МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра програмування
(повна назва кафедри)

Магістерська робота

СТВОРЕННЯ ВЕБ ЗАСТОСУНКІВ У СЕРЕДОВИЩІ PHARO

Виконав: студент групи ПМІМ-21
спеціальності
122 – комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Рібій В. В.
(прізвище та ініціали)

Керівник (підпис) Ярошко С.А.
(прізвище та ініціали)

Рецензент (підпис) Музичук Ю.А.
(прізвище та ініціали)

ДЕКАН

Львів – 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *програмування*

Освітньо-кваліфікаційний рівень *магістр*

Спеціальність *122 – комп'ютерні науки*

«ЗАТВЕРДЖУЮ»
Зав. кафедрою *доц. Ярошко С.А.*
« 13 » *вересня* 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Рібій Віталій Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи
Створення веб застосунків в середовищі Pharo

керівник роботи *доц. Ярошко С.А.*
затверджена Вченою радою факультету від « *13* » *вересня* 20 *22* р., № *15*

2. Строк подання студентом роботи *12.12.2022 р.*

3. Вхідні дані до роботи
Література та інтернет-ресурси за тематикою роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вивчити фреймворк Seaside. Розробити веб-сайт притулку для тварин. На основі розробленого веб-сайту сформувати інструкцію для початківців. Дослідити можливі проблеми з які можуть виникнути у початківця та знайти описати їх вирішення у інструкції.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *13.09.22 р.*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд існуючих систем та технологій*	*Вересень*	
2.	*Постановка задачі*	*Вересень*	
3.	*Розробка веб сайту*	*Жовтень*	
4.	*Розробка веб сайту*	*Листопад*	
5.	*Розробка інструкції на базі веб сайту*	*Листопад*	
6.	*Оформлення роботи*	*Грудень*	

Студент _____ *Рібій В.В.*
підпис

Керівник роботи _____ *доц. Ярошко С.А.*
підпис

ЗМІСТ

Вступ 7
1 Мета 7
2 Актуальність 8
3 Побудова веб-сайту 8
3.1 Загальний огляд стеку технологій 8
3.2 Створення віртуального середовища та огляд інтерфейсу 9
3.3 Створення репозиторію. 12
3.4 Створення основних моделей 16
3.4.1 Модель даних застосунку 16
3.4.2 Клас Animal 17
3.4.3 Клас Breed 23
3.5 Тестування коду 26
3.6 Зберігання моделей у базі даних 29
3.6.1 Інсталювання Voyage 29
3.6.2 Конфігурація Voyage для збереження даних 30
3.6.3 Зберігаємо дані 31
3.6.4 Надсилаємо запити до бази даних 32
3.7 Вступ до Seaside 33
3.7.1 Додавання необхідних пакетів 33
3.7.2 Запуск Zinc Server 33
3.7.3 Оголошення точки входу веб застосунку 35
3.7.4 Візуалізація компонент 36
3.7.5 Архітектура веб-сайту 36
3.8 Створення веб компонент головної сторінки 37
3.8.1 AHScreenComponent 37

3.8.2 AHHeaderComponent..... 39
3.8.3 Ієрархія та відношення компонентів у Seaside. 41
3.8.4 AHMainScreenComponent..... 41
3.8.5 AHAnimalsGrid..... 42
3.8.6 Фільтрація в галереї. 50
3.8.7 Карусель фотографій 54
3.8.8 Форма – повідомити про тварину. 55
3.9 Автентифікація та сесії..... 59
3.9.1 Перехід на сторінку адміністратора..... 59
3.9.2 Повернення на головну сторінку..... 61
3.9.3 Автентифікація..... 62
3.9.4 Відображення помилок логування 65
3.9.5 Модель користувача 66
3.9.6 Сесія..... 69
3.9.7 Спрощена навігація..... 70
3.9.8 Реєстрація користувача 71
3.10 Створення компонент сторінки адміністратора 74
3.10.1 AHFoundAnimalCard..... 75
3.10.2 AHWantAnimalCard 77
3.10.3 Відображення вмісту AHAdminComponent 80
3.11 Подальша доля тварин..... 82
3.11.1 Статус тварини 82
3.11.2 AHTakenAnimalsComponent 83
3.11.3 *AHTakenAnimalsGrid* та *AHTakenAnimalCard* 85
3.11.4 Додавання звітів про тварину 87
3.11.5 Статистика забраних тварин..... 91
3.12 Відгуки 92

3.12.1 Модель відгуку 92
3.12.2 AHReviewsComponent 93
3.12.3 Форма відгуку на притулок 94
3.12.4 Відгук на користувача 97
4 Демонстрація програми 99
Висновок 109
Список літератури 110

ВСТУП

Pharo – це сучасна повнофункціональна реалізація середовища мовою програмування *Smalltalk* з відкритим вихідним кодом. *Pharo* пропонує гнучку платформу для професійної розробки програмного забезпечення, а також надійну і стабільну основу для наукових досліджень і розробки в галузі динамічних мов і середовищ. Багато спеціалістів у сфері комп'ютерних технологій обирають *Pharo*, як альтернативу *Python* та *Javascript*, адже синтаксис є набагато легшим, а написання коду є зручнішим та простішим.

У сфері розробки веб застосунків, *Pharo* показує себе як дуже надійний інструмент, що з допомогою своїх особливостей, а саме можливості налагоджування та редагування коду та розмітки сайту «на льоту» дає змогу дуже легко діагностувати та виправляти помилки. А не статична розмітка веб сайту дає змогу робити сторінки динамічними, бо розмітка генерується в процесі виконання.

1 МЕТА

Мета магістерської роботи – розглянути та продемонструвати:

- додавання сторонніх пакетів до образу *Pharo;*
- роботу із системами контролю версій;
- інтеграцію з базами даних;
- розробку веб інтерфейсів через *Seaside* та *Bootstrap;*
- керування сесіями та автентифікацію;

А результатом цієї роботи має бути описана покрокова інструкція по реалізації задачі створення сайту «притулку для тварин».

2 АКТУАЛЬНІСТЬ

На даний момент найновіша інструкція по розробці веб застосунків на Pharo застаріла, оскільки активно випускалися нові версії, внаслідок чого, багато чого стало недоступним у нових версіях, або ж з’явились нові, кращі підходи та інструменти для виконання поставлених задач.

Також інструкція опублікована англійською та французькою мовами, що додає складнощів для українських розробників та може відбити бажання продовжувати вивчення даної перспективної технології.

3 ПОБУДОВА ВЕБ-САЙТУ

У цій інструкції ми розробимо веб-сайт притулку для тварин та розглянемо як:

- Створити та зберегти моделі даних.
- Створювати та відображати веб компоненти.
- Стилізувати компоненти за допомогою *css* та *bootstrap*.
- Змінювати компоненти, що відображаються.
- Налаштувати автентифікацію та сесійність.

3.1 Загальний огляд стеку технологій

У сучасній розробці веб-сайтів в середовищі *Pharo* використовують:

- *Zinc* для створення локального сервера.
- *Seaside-REST* для обробки *REST* запитів.
- *Seaside* для генерації та рендерингу веб компонентів.
- *Voyage* для комунікації з *NoSql* базами даних.
- *Magritte* для автоматичної генерації форм на основі опису моделі.
- *Bootstrap* для швидкого використання та стилізації компонент.

3.2 Створення віртуального середовища та огляд інтерфейсу

Після встановлення, запускаємо *Pharo Launcher* та через вкладку «new» створюємо новий образ середовища, обираємо «Official distributions/*Pharo 10.0–64 bit* (*stable*)» і вводимо назву «*AnimalsHouse*».

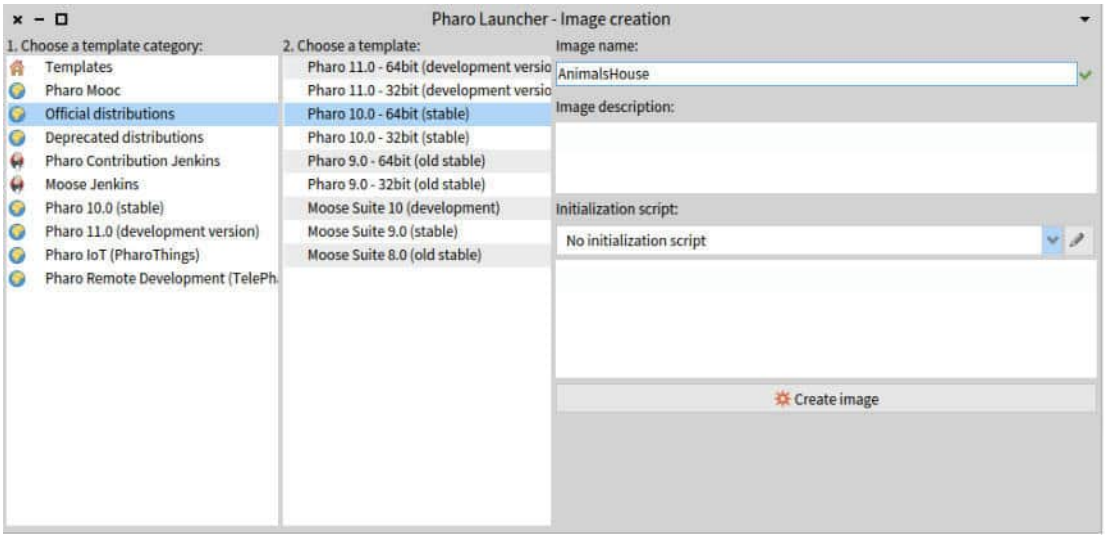

Рис. 3.1 Створення образу Pharo

Після цього в списку образів – вибираємо щойно створений та запускаємо.

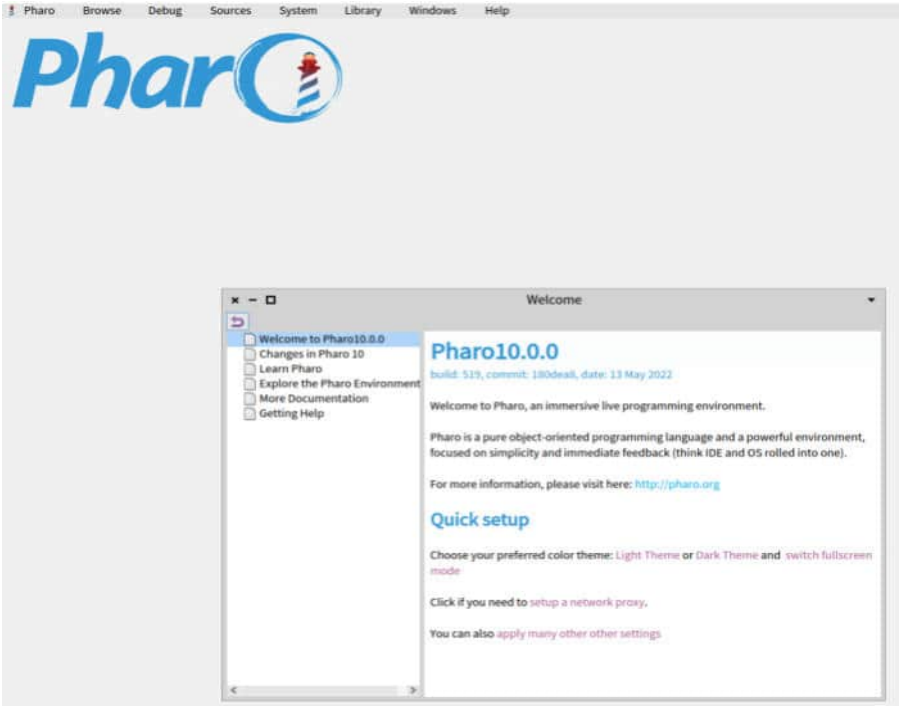

Рис. 3.2 Стартове вікно середовища Pharo

У відкритому вікні можна ознайомитись із корисними посиланнями на вивчення синтаксису мови та іншої документації. Клацання лівою кнопкою миші в довільному місці вікна середовища програмування відкриває World-меню – головне меню системи:

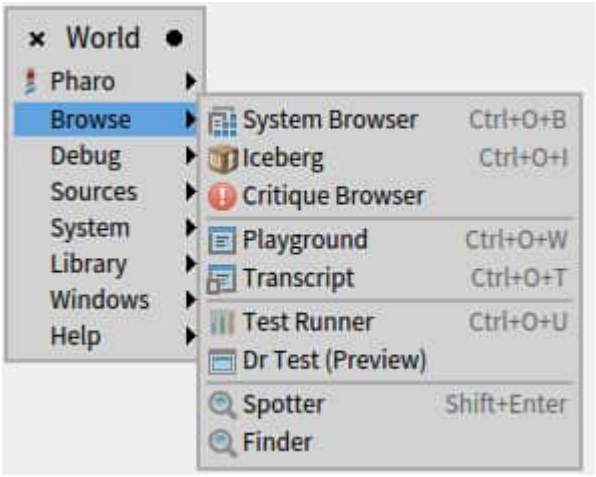

Рис. 3.3 World-меню

У пункті «*browse*» можемо бачити кілька корисних команд, які ми будемо часто використовувати, для кожного з них є свої гарячі клавіші.

System Browser – програма, що дає змогу переглядати та додавати до системи різноманітні пакети та класи. У цьому вікні власне ми проведемо найбільше часу.

Iceberg – система контролю версій. З її допомогою можна зберігати код у локальному та віддаленому сховищі, імпортувати та експортувати пакети розроблені в середовищі Pharo.

Playground – це вікно для написання коду в першу чергу для тестових цілей, або прописування скриптів

Transcript – аналог консолі в інших середовищах, часто використовується для налагодження.

Spotter – інструмент пошуку в середовищі, дуже корисний, коли необхідно знайти певний клас, або того, хто надсилає певне повідомлення.

Також дуже важливим елементом інтерфейсу є збереження образу, оскільки у випадку виходу з образу, усі зміни внесені до нього, не будуть збережені якщо

ми не зберегли їх перед виходом, або ж під час виходу. Для цього необхідно використати спеціальну опцію у World-меню. А саме “*Save and quit*” та “*Save*”.

У випадку якщо ви вийшли не зберігши зміни, їх можна відновити якщо запускати образ з обраного в файловій системі файлу, що автоматично генерується, зазвичай цей файл знаходиться у папці *pharo-local* потрібного образу.

Ще однією корисною функцією є припинення виконання вже запущеного коду. Особливо це корисно у випадку коли потік виконання потрапляє у вічний цикл. Для того щоб його перервати, нам необхідно натиснути комбінацію клавіш: *CTRL+*”.”

І останньою важливою особливістю середовища розробки *Pharo*, є можливість додавати/міняти код під час налагодження. Усе що для цього потрібно, це під час появи вікна налагодження редагувати у ньому рядки коду та зберегти. Або, якщо шуканий метод не знайдений, ми можемо натиснути кнопку “*Create”* та обрати куди додати необхідний нам метод, та який функціонал буде у ньому. Після того як усі зміни були внесені, можна натиснути кнопку *Proceed* і відновити роботу потоку з того ж місця.

3.3 Створення репозиторію.

Для початку, потрібно створити репозиторій. Створимо його на платформі *github.com* і відкриємо *Iceberg*.

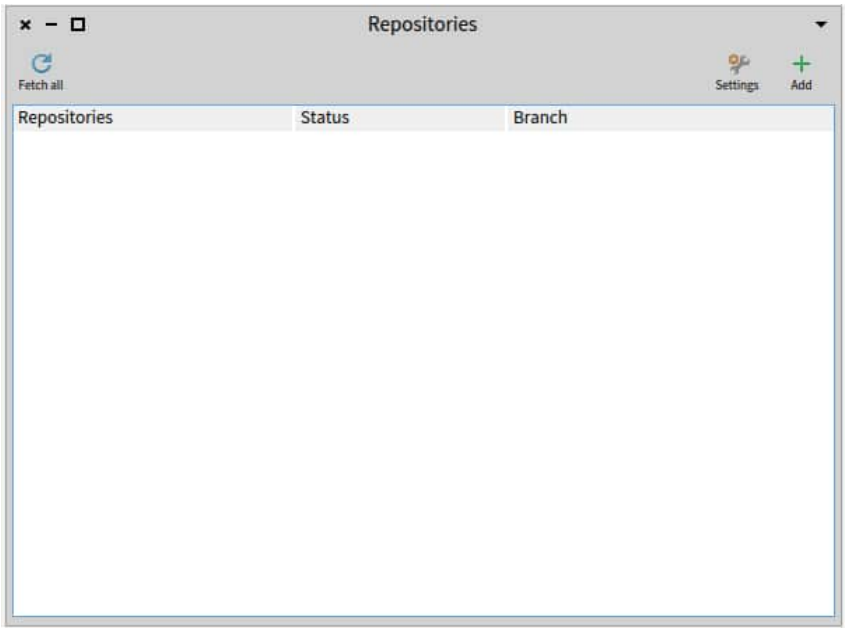

Рис. 3.4 Вікно репозиторіїв

Тут будуть відображатись усі «стягнуті» репозиторії. Поки що тут немає нічого, тому давайте додамо наш репозиторій. Натискаємо *Add* та обираємо *Clone* from github.com, справа вводимо автора проекту, назву проекту, обираємо локацію для репозиторію та протокол. Рекомендую вибрати HTTPS.

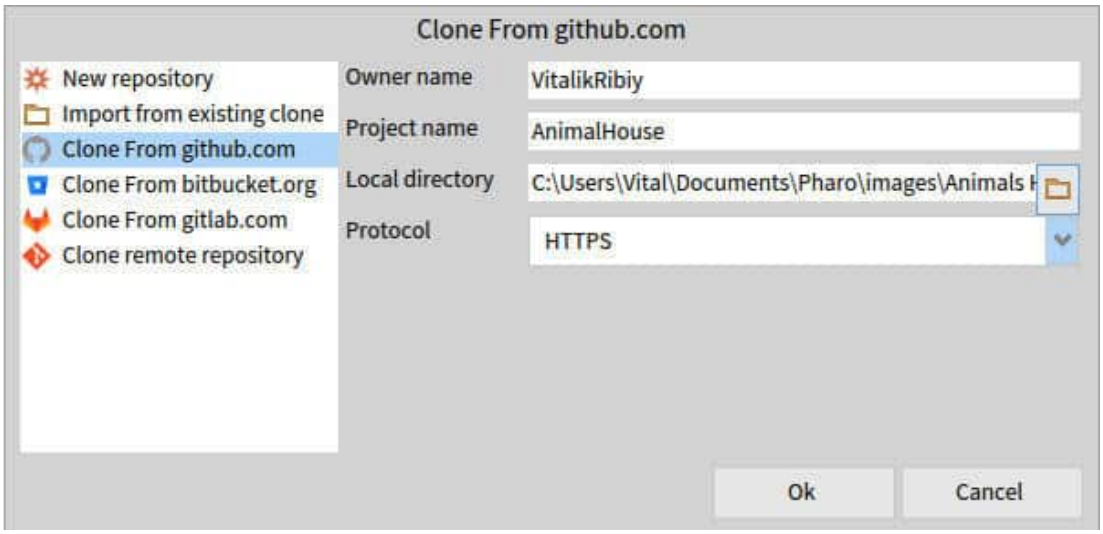

Рис. 3.5 Вікно клонування репозиторію

Для логування нам необхідно створити токен для автентифікації. Для цього, в налаштуваннях акаунту на *github.com* необхідно зайти в *developer settings* та створити тимчасовий токен. Далі, під час логування, у Pharo, замість паролю – просто вставте цей токен. Або ж відкрийте Playground та пропишіть наступні рядки коду:

```
IceCredentialStore current
    storeCredential: (IceTokenCredentials new
        username: 'your@email.com';
        token: 'token value';
        yourself)
    forHostname: 'github.com'.
```

Після логування бачимо, що в списку репозиторіїв з’явився наш.

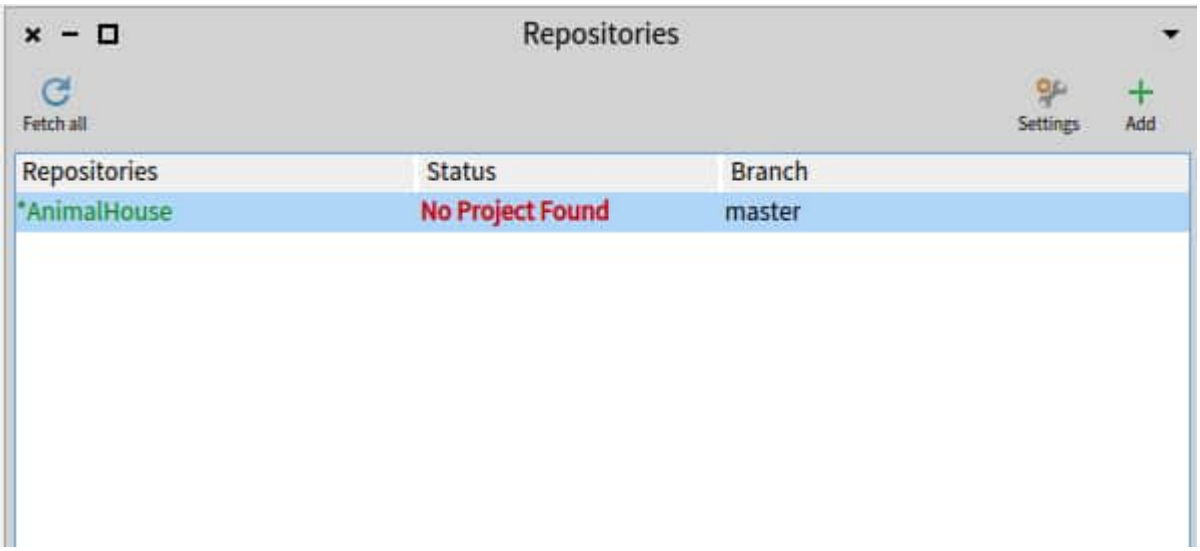

Рис. 3.6 Вікно репозиторіїв

Як бачимо статус репозиторію вказує на відсутність проекту, давайте це виправимо. Натискаємо правою кнопкою на репозиторій та вибираємо «*repair*» після чого бачимо наступне вікно:

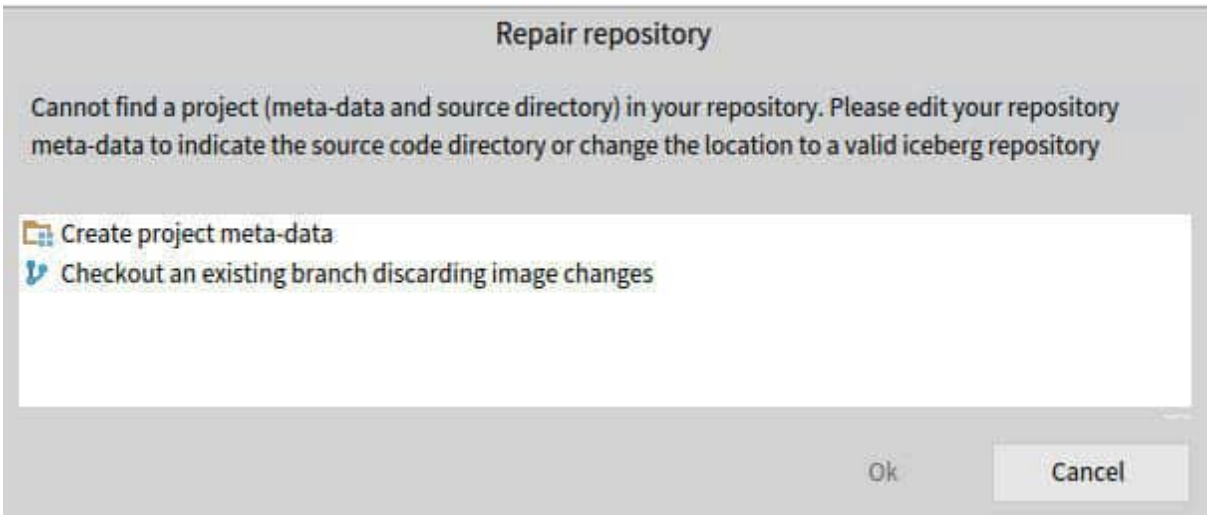

Рис. 3.7 Вікно ремонту репозиторію

Вибираємо перший пункт та додаємо в кореневу папку, папку src.

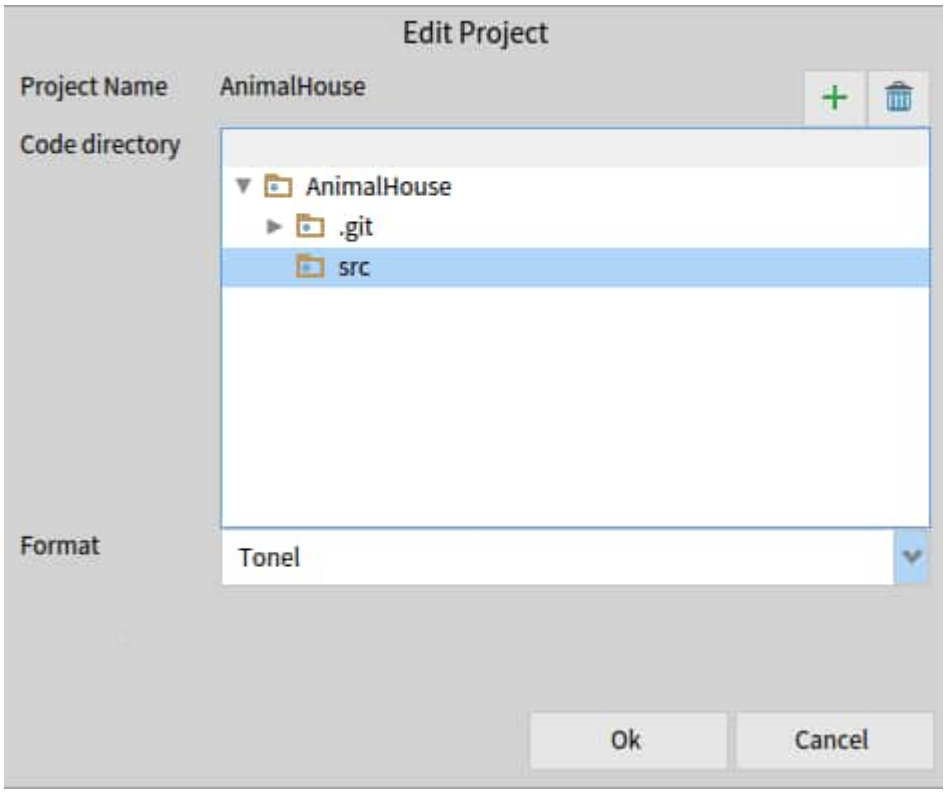

Рис. 3.8 Вікно редагування проекту

Натискаємо *Ok* та бачимо, що статус змінився на «*Not loaded*». Двічі натискаємо на наш репозиторій та переходимо в наступне вікно:

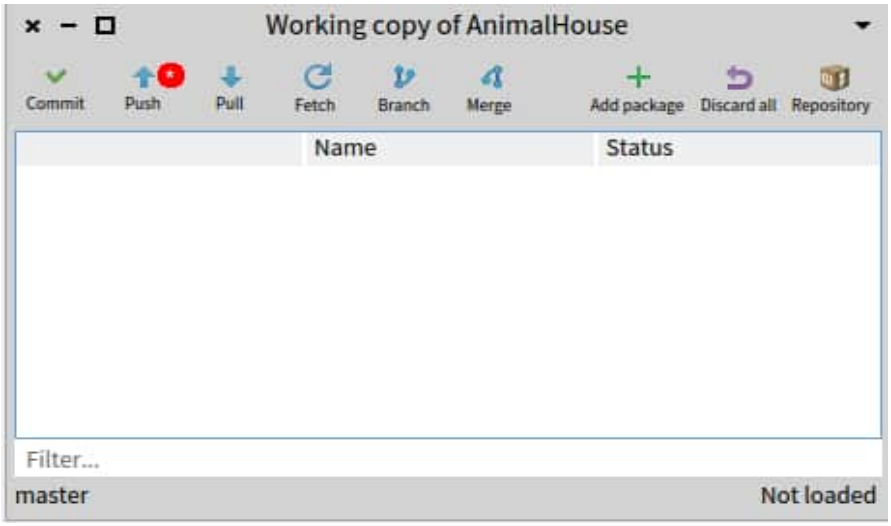

Рис. 3.9 Вікно контролю над репозиторієм

Тут ми маємо доступ до стандартних команд системи контролю версій *Github*. Для початку зробимо свій перший запис до сховища. Натиснемо на кнопку «commit» та побачимо, що з'явилось наступне вікно:

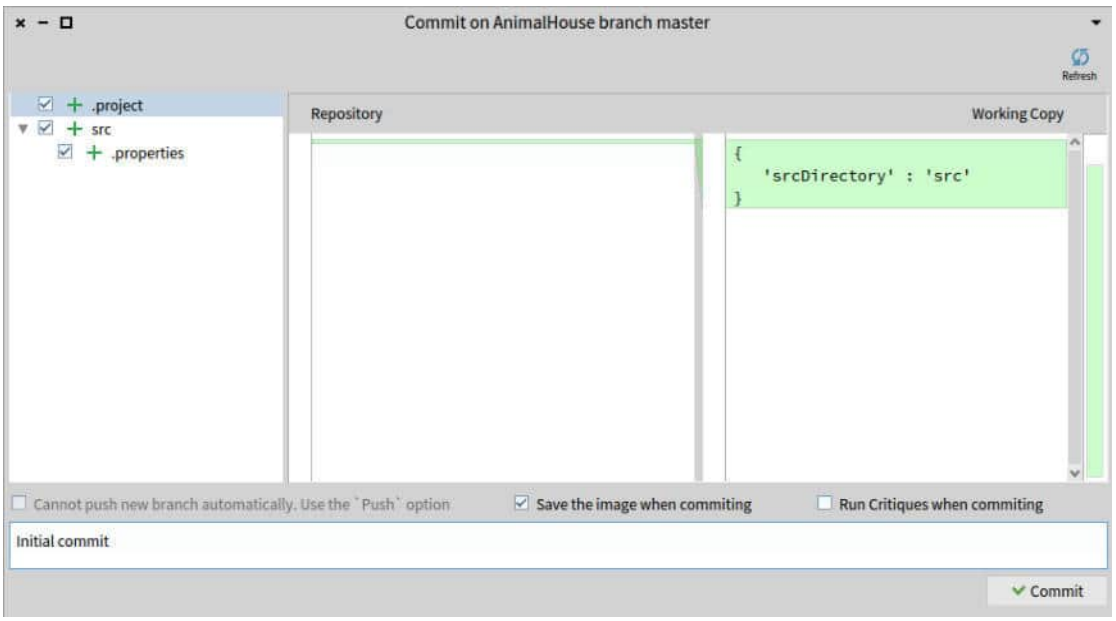

Рис. 3.10 Вікно створення запису

Тут ми можемо побачити всі локальні зміни, що відбулись відносно віддаленої версії гілки. Дописуємо коментар для запису, нехай це буде «Initial commit» та натискаємо «Commit».

Після цього натискаємо «Push». Вітаю, ви створили та вивантажили свій перший запис!

3.4 Створення основних моделей

3.4.1 Модель даних застосунку

Для початку потрібно спланувати структуру та відношення між моделями даних. Основними будуть модель тварини *Animal* та користувача *AHUser*, також у них є свої додаткові моделі які містять додаткову інформацію. Такими є моделі породи *Breed* та інформації про користувача *AHUserInfo*.

Також для реалізації функціоналу сайту нам необхідні додаткові моделі, такі як *AHReview* – модель відгуку, *AHReport* – модель звіту про забрану тварину, *AHWantAnimal* – модель заявки на отримання та *AHFoundAnimal* – модель повідомлення про знайдену тварину.

Відношення моделей та список колонок кожної моделі зображені на схемі нижче.

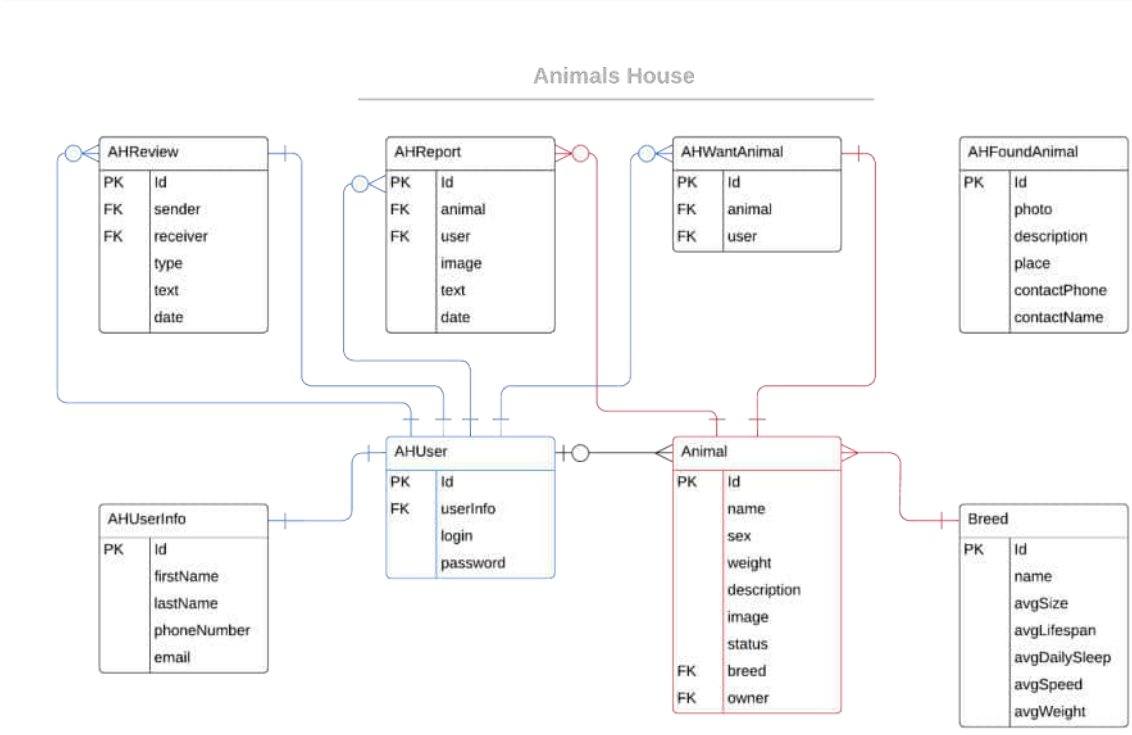

Рис. 3.11 Схема відношення моделей

3.4.2 Клас Animal

У цьому розділі ми розробимо наші доменні моделі сайту. Основна модель у нашій системі буде тварина. Відкриваємо *System Browser* та в полі для коду створюємо клас *Animal*.

```
Object subclass: #Animal
    instanceVariableNames: 'name age sex animalSpecies breed weight
     healthStatus description image'
    classVariableNames: ''
    package: 'AnimalsHouse'
```

Рис. 3.12 Створення класу Animal

Після збереження можна помітити, що в списку пакетів з'явився *AnimalsHouse*, натиснувши на нього можна пройти до щойно створеного класу *Animal*.

Поруч із класом ви можете побачити знак оклику, це означає, що класу необхідно додати коментар. Це хороша практика коментувати свій код, тож давайте напишемо, для чого цей клас.

```
Class: Animal
_____________________________________

This class represents abstract class of an animal

Instance Variables
age:  <Number>
breed:  <Breed>
description:  <ByteString>
animalSpecies:  <ByteString>
healthStatus:  <ByteString>
name:  <ByteString>
sex:  <ByteString>
weight:  <Number>
```

Рис. 3.13 коментар до класу Animal

Таким чином ми вказали, що *Animal* це лише абстрактний клас, для якого ми створимо нащадків та вказати типи для змінних. Серед них ми можемо побачити нестандартний тип Breed, ми повернемось до нього пізніше.

Наступне, що нам потрібно зробити, це додати методи, що дають змогу присвоювати та отримувати значення змінних екземпляру нашого об’єкту Animal. Середовище Pharo надає дуже зручний інструментарій для цього. Усе, що нам потрібно зробити, це натиснути правою кнопкою на наш клас та знайти опцію «Generate accessors»

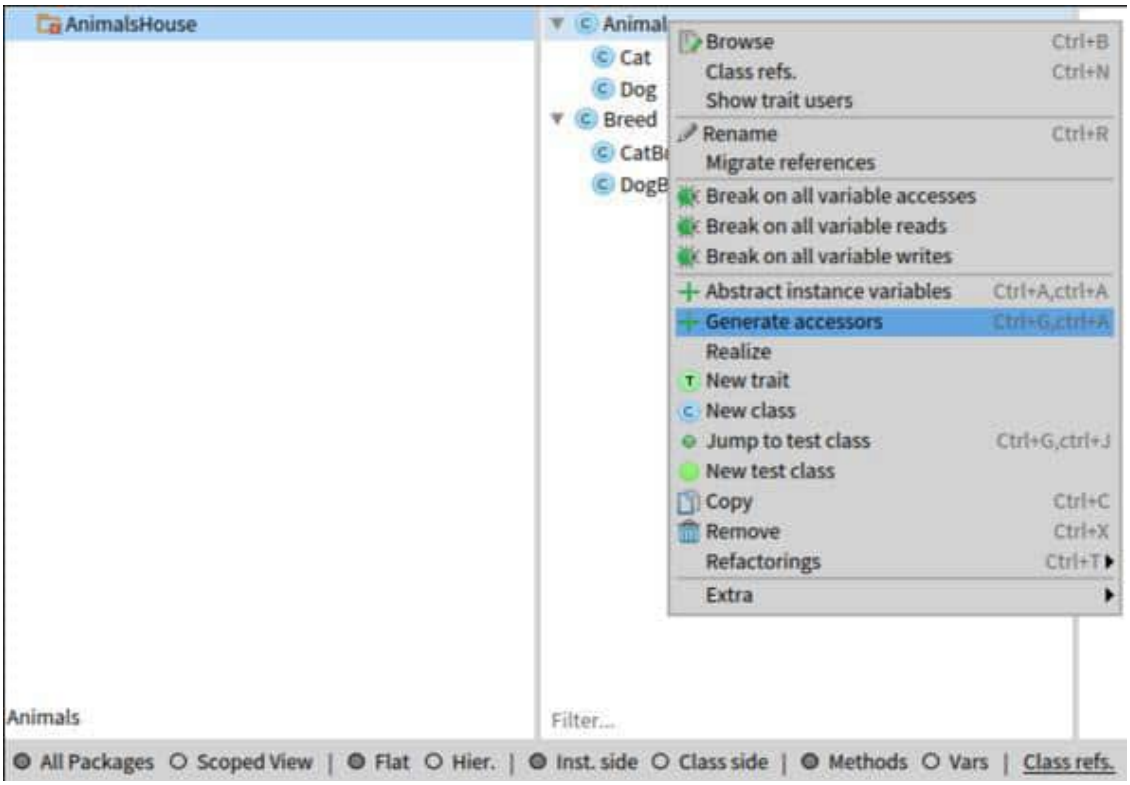

Рис. 3.14 Генерація методів доступу

Після цього побачимо, що в нас з’явилося по два методи доступу до змінних екземпляра: селектор і модифікатор

Давайте модифікуємо ці методи , щоб забезпечити відповідність змінної та типу даних. Розглянемо на прикладі методу для присвоєння віку тварини, за замовчуванням метод виглядає так:

```
age: anObject

  age := anObject
```

Рис. 3.15 Метод присвоєння віку тварини за замовчуванням

Однак такий метод дозволяє присвоїти текстове значення змінній, яка має бути числовим. Тоді, додаймо перевірки, які не дадуть цього зробити.

```
age: anObject
  anObject isNumber
  ifTrue: [ age := anObject ]
  ifFalse: [ self error: 'age should be an instance of number' ]
```

Рис. 3.16 Модифікований метод присвоєння віку тварини

За аналогією змінимо наступні методи:

```
description: desc
  desc isByteString
  ifTrue: [ description := desc ]
  ifFalse: [ self error: 'description should be an instance of ByteString' ]
healthStatus: status
  status isByteString
  ifTrue: [ healthStatus := status ]
  ifFalse: [ self error: 'healthStatus should be an instance of ByteString' ]
name: nameValue
  nameValue isByteString
  ifTrue: [ name := nameValue ]
  ifFalse: [ self error: 'name should be an instance of ByteString' ]
sex: sexValue
  sexValue isByteString
  ifTrue: [ sex := sexValue ]
  ifFalse: [ self error: 'sex should be an instance of ByteString' ]
weight: anObject
  anObject isNumber
  ifTrue: [ weight := anObject ]
  ifFalse: [ self error: 'weight should be an instance of number' ]
```

Рис. 3.17 Методи присвоєння значень для змінних тварини

Оскільки *Animal* це абстрактний клас, нам потрібно заборонити створення об’єкту цього класу. Для цього, додаємо методи на стороні класу.

```
new
  self == Animal ifTrue: [
      ^ self error: 'Animal is an abstract class.  Make a concrete subclass.' ].
    ^ super new
```

Рис. 3.18 Модифікований метод new

Тепер, якщо спробуємо створити новий об’єкт класу *Animal*, отримуємо повідомлення про помилку:

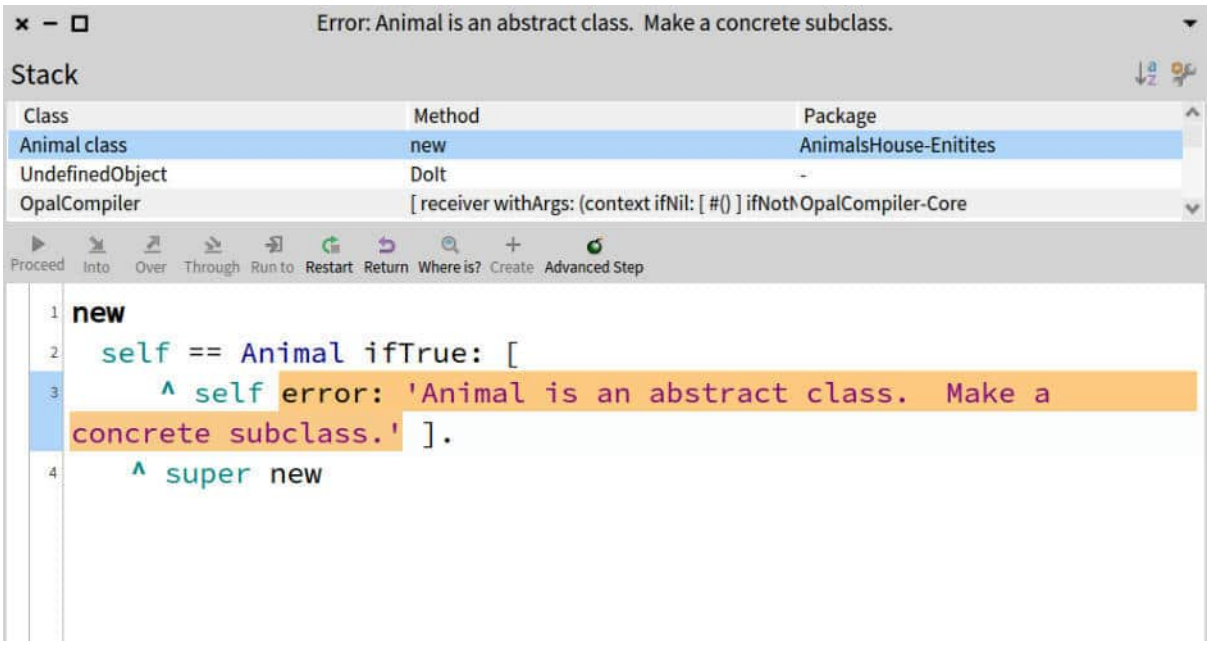

Рис. 3.19 Вікно налагоджування

Наступне що ми зробимо – створимо класи, які будуть наслідувати *Animal*, а саме *Cat* та *Dog*.

```
Animal subclass: #Cat
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'AnimalsHouse-Enitites'
```

Рис. 3.20 Створення класу Cat

```
Animal subclass: #Dog
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'AnimalsHouse-Enitites'
```

Рис. 3.21 Створення класу Dog

Оскільки ці класи наслідують *Animal,* для них доступні всі методи та змінні, що оголошені в цьому класі. Давайте для зручності додаємо повідомлення *isCat* та *isDog* у клас *Animal* при цьому обидва методи повертатимуть значення *false.*

```
isCat        isDog

    ^ false      ^ false
```

Рис. 3.22 Методи isCat та isDog для класу Animal

Можна побачити, що тепер нащадки класу *Animal* мають зобов'язання реалізувати це повідомлення. Його код буде дуже простим, але цей метод дозволить нам відрізняти об'єкти класу *Animal* один від одного.

```
isCat

    ^ true
```

Рис. 3.23 Реалізація для класу *Cat*,

```
isDog

    ^ true
```

Рис. 3.24 Реалізація для класу Dog

Також, для створення нових об'єктів, додаймо метод на рівні класу *Animal*:

```
withName: nameValue age: ageValue sex: sexValue breed: breedValue weight: weightValue healthStatus:
healthStatusValue description: descriptionValue image: imageValue
    | instance |
    instance := self new.
    instance name: nameValue;
    age: ageValue;
    sex: sexValue;
    breed: breedValue;
    weight: weightValue;
    healthStatus: healthStatusValue;
    description: descriptionValue;
    image: imageValue.
    ^ instance.
```

Рис. 3.25 Метод для створення об'єктів класу Animal

Оголосивши цей метод, ми можемо використати його до конкретного класу, *Cat* чи *Dog* і отримати об'єкт цього класу. У свою чергу надіславши це повідомлення до класу *Animal*, отримаємо помилку як у випадку з повідомленням *new*.

Для зручності, створимо класи *Sex* та *Species* які будуть допомагати нам отримувати рядок, для створення об'єктів.

```
Object subclass: #Sex
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'AnimalsHouse-Enums'
```

```
Object subclass: #Species
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'AnimalsHouse-Enums'
```

Рис. 3.26 Створення класів Sex та Species

Для класу *Sex* додаємо повідомлення *female* та *male*:

```
female
  ^ 'Самка'.
```

```
male
  ^ 'Самець'.
```

Рис. 3.27 Реалізація методів female та male

Для класу Species додаємо повідомлення *cat* та *dog*:

```
cat
  ^ 'Кіт'.
```

```
dog
  ^ 'Пес'
```

Рис. 3.28 Реалізація методів cat та dog

Перевизначимо метод *initialization* для класів *Cat* та *Dog* на рівні об’єктів.

```
initialize
  animalSpecies := Species cat.
```

```
initialize
  animalSpecies := Species dog.
```

Рис. 3.29

Тепер спробуємо створити об’єкт класу *Cat* з нашими новими методами.

```
Cat withName: 'Ally' age: 3 sex: Sex female
  breed: 'Husky' weight: 25 healthStatus: 'Healthy'
  description: 'description' image: nil.
```

Рис. 3.30 Приклад створення об’єкту кота

У результаті отримуємо об’єкт із заданими значеннями змінних.

3.4.3 Клас Breed

Важливою частиною класу *Animal* є змінна екземпляру *breed*, давайте створимо цей клас.

```
Object subclass: #Breed
   instanceVariableNames: 'name avgSize avgLifespan
   avgDailySleep avgSpeed avgWeight animalSpecies'
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'
```

Рис. 3.31 Створення класу Breed

За аналогією з *Animal*, генеруємо метод для змінних екземпляру. Додаємо метод *isBreed*, для майбутніх перевірок чи об’єкт належить до класу *Breed*.

```
isBreed
    ^ true.
```

Рис. 3.32 Реалізація методу isBreed

Також додаємо перевизначимо метод new на рівні класу, за аналогією як у *Animal.*

```
new
self == Breed ifTrue: [
     ^ self error: 'Breed is an abstract class.  Make a concrete subclass.' ].
  ^ super new
```

Рис. 3.33 Модифікація методу new для класу Breed

Оскільки порода має бути унікальною, необхідно мати можливість переглядати список усіх порід та знаходити об’єкт за назвою породи.

```
breeds
    ^ self allInstances.
```

Рис. 3.34 Метод отримання списку порід

```
getByName: nameOfBreed
    | result |
    result := (self breeds select: [ :e | e name = nameOfBreed ]).
    result isNotEmpty ifTrue: [ ^ result at: 1 ]
    ifFalse: [ ^ nil ]
```

Рис. 3.35 Метод отримання об’єкту породи за ім’ям

Тепер можна додати метод для створення об’єкту з заданими значеннями та у випадку, якщо така порода вже існує, вертати вже існуючий об’єкт:

```
withName: nameValue avgSize: avgSizeValue avgWeight: avgWeightValue avgLifespan: avgLifeSpanValue
avgDailySleep: avgDailySleepValue avgSpeed: avgSpeedValue
    | instance |
    self == Breed ifTrue: [
        ^ self error: 'Breed is an abstract class.  Make a concrete subclass.' ].
    instance := self getByName: nameValue.
    instance isNotNil
        ifFalse: [
            instance := self new.
            instance name: nameValue;
                avgSize: avgSizeValue;
                avgWeight: avgWeightValue;
                avgLifespan: avgLifeSpanValue;
                avgDailySleep: avgDailySleepValue;
                avgSpeed: avgSpeedValue.
        ].
    ^ instance.
```

Рис. 3.36 Метод створення об’єкту класу Breed

Так як Breed це абстрактний клас, пора додати його класи нащадки: *CatBreed* та *DogBreed*.

```
Breed subclass: #DogBreed
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Enitites'
Breed subclass: #CatBreed
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Enitites'
```

Рис. 3.37 Створення класів CatBreed та DogBreed

Для розрізнення, у класі *Breed* ми додали змінну *animalsBreed* і перевизначили метод initialize на рівні об’єкту класу.

```
initialize
    animalSpecies := self animalSpecies.
```

Рис. 3.38 Метод initialize

Повідомлення *self* відноситься до первинного класу об’єкту, тож коли ми створимо об’єкт класу *CatBreed* в методі initialize виконається метод *animalSpecies* реалізація, якого буде записана в *CatBreed*. Давайте додаємо ці методи в класи *CatBreed* та *DogBreed*.

```
animalSpecies
    ^ Species cat.
```

```
animalSpecies
    ^ Species dog.
```

Рис. 3.39 Реалізація методу animalSpecies для класів Cat та Dog

А на стороні класу у *CatBreed* та *DogBreed* перевизначимо метод *initialize*. Таким чином, у нас будуть окремі списки порід котів та собак.

```
initialize
    breeds := OrderedCollection new.
```

Рис. 3.40 Метод initialize

Останнє, що нам потрібно зробити – модифікувати створення тварин, так щоби змінна *breed* відповідав класу *breed* та виду тварини.

```
breed: anObject
    (anObject isBreed)
    ifTrue: [
        ((self isCat and: anObject class == CatBreed) or: (self isDog and: anObject class == DogBreed))
        ifTrue: [ breed := anObject ]
        ifFalse: [ self error: 'breed object do not correspond to animal specie' ]
        ]
    ifFalse: [ self error: 'breed: should be an instance of Breed' ]
```

Рис. 3.41 Реалізація методу breed:

На цьому ми завершуємо роботу над моделями та переходимо до способів тестування нашого коду.

3.5 Тестування коду

Дуже важливою частиною розробки програмного забезпечення є написання тестів, вони слугують гарантією того що функціонал працює коректно.

Для початку, створимо тестовий клас. Назва тестового класу має складатись з назви класу що буде тестуватись та слова “Test”. Створимо клас *AnimalTest* у новому пакеті AnimalsHouse-Entities-Tests додавши відповідний текст в оголошенні класу.

```
TestCase subclass: #AnimalTest
   instanceVariableNames: 'testAnimal testBreed'
   classVariableNames: ''
   package: 'AnimalsHouse-Entities-Tests'
```

Рис. 3.42 Створення класу AnimalTest

Кожен тест потребує початкових даних, тому для того, щоб не створювати ці дані вручну для кожного тесту, додамо метод *setUp*, що буде автоматично виконуватись перед кожним запуском тестування і надавати потрібні дані для тестування.

```
setUp
   testBreed := CatBreed withName: 'British' avgSize: 13 avgWeight: 5 avgLifespan: 15 avgDailySleep: 16 avgSpeed: 7.
   testAnimal := Cat withName: 'TestCatName' age: 1 sex: Sex female breed: testBreed weight: 5 healthStatus: 'Healthy' description: 'test description' image: nil.
   super setUp.
```

Рис. 3.43 Метод setUp

Також важливою умовою написання тестів є те, що вони мають бути незалежними один від одного. Тож після кожного виконання тесту нам необхідно очищувати результати його виконання. Для цього додамо метод *tearDown.*

```
tearDown
   testAnimal := nil.
   testBreed := nil.
```

Рис. 3.44 Метод tearDown

Для того, щоб тестувати метод класу, нам необхідно додати метод до тестового класу з префіксом test. Спробуємо додати тестовий метод для отримання віку тварини. Він має містити повідомлення *self assert: equals*: для перевірки відповідності двох значень, очікуваного та реального.

```
testAge

    self assert: testAnimal age equals: 1
```

Рис. 3.45 Метод для тестування віку тварини

Після створення цього методу, можемо побачити, що біля нього та біля назви класу у System Browser з’явився сірий значок. Натиснувши на нього, ми можемо запустити тестування методу, або ж усього класу. Також такі круги з’являються біля методів які були протестовані у класі який ми тестували. Це дозволяє нам перевірити правильність виконання методу, не переходячи до тестового класу.

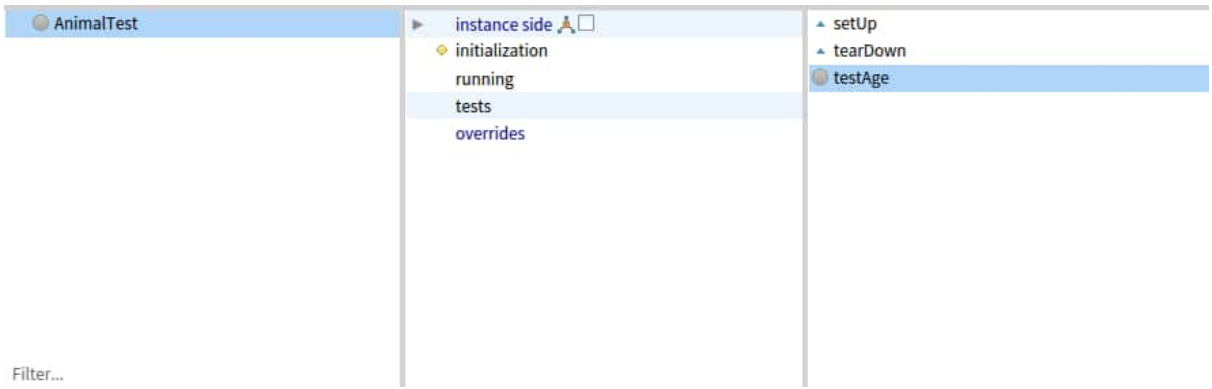

Рис. 3.46 Створений метод з значком для тестування

Рис. 3.47 Методи доступу до віку тварини

Запустивши тестування цього методу ми можемо побачити повідомлення про результати тестування, а методи, які було протестовано, підсвітились зеленим кольором.

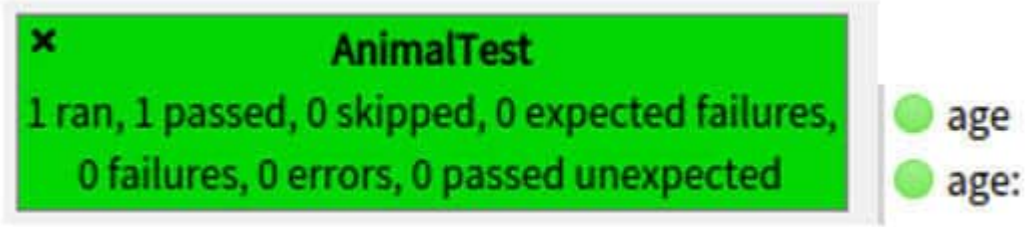

Рис. 3.48 Результат тестування

Створимо ще кілька тестових методів та перевіримо їх виконання.

```
testBreed
    self assert: testAnimal breed equals: testBreed.
testName
    self assert: testAnimal name equals: 'TestCatName'.
testSex
    self assert: testAnimal sex equals: Sex female.
testWeight
    self assert: testAnimal weight equals: 5.
testHealthStatus
    self assert: testAnimal healthStatus equals: 'Healthy'.
testDescription
    self assert: testAnimal description equals: 'test description'.
testIsCat
    self assert: testAnimal isCat equals: true.
testIsDog
    self assert: testAnimal isDog equals: false.
```

Рис. 3.49 Методи для тестування методів доступу

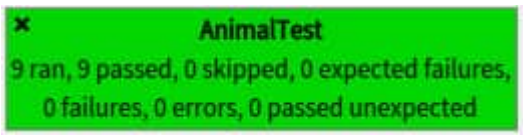

Рис. 3.50 Результат тестування

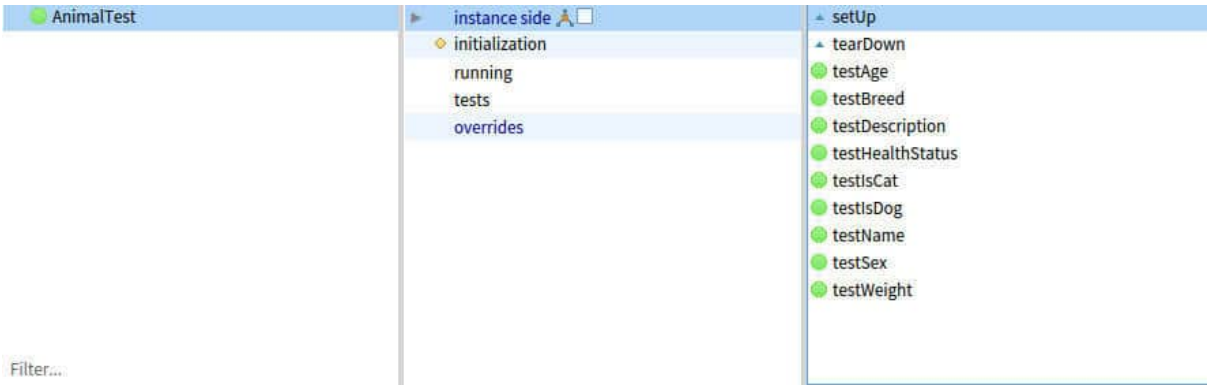

Рис. 3.51 Тестові методи із зеленим значком

Таким чином у нас є можливість легко протестувати наш функціонал у випадку внесення змін до нього.

3.6 Зберігання моделей у базі даних

До цього часу наші об’єкти зберігались у пам’яті та після перезапуску середовища зникали. Для веб сайту це недопустимо, нам необхідно мати можливість зберігати дані не у оперативній пам’яті, оскільки після перезапуску програми, ці дані зникнуть. Таким місцем у нашому випадку буде *NoSQL* база даних *MongoDB*.

Найпопулярнішим інструментом роботи з базами даних у *Pharo* є *Voyage* framework.

3.6.1 Інсталювання Voyage

Для того, щоб використовувати функціонал цього фреймворку, нам необхідно додати відповідний пакет у образ. Для цього у Playground нам необхідно запустити наступні рядки коду:

```
Metacello new
    repository: 'github://pharo-nosql/voyage:pharo10-ready/mc';
    baseline: #Voyage;
    onConflictUseLoaded;
    load.
```

Після цього, відкривши Iceberg, ви зможете побачити, що серед репозиторіїв з’явився новий, з назвою voyage,

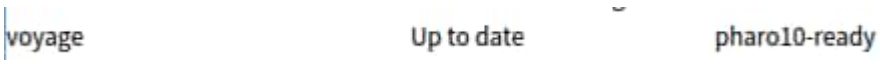

Рис. 3.52 Доданий репозиторій Voyage

Також необхідно встановити власне *MongoDB* сервер, завантажити його можна на сайті https://www.mongodb.com/.

3.6.2 Конфігурація Voyage для збереження даних

Оголосивши метод *isVoyageRoot* на рівні класу, ми вказуємо, що цей об’єкт має бути збережений у базі даних як кореневий об’єкт. Це означає, що кожен екземпляр цього класу буде збережений як окремий запис у базі.

Кореневим об’єктом потрібно робити такі об’єкти, що будуть часто потрібні як окремі сутності. У нашому випадку таких дві, *Animal* та *Breed* тож оголошувати метод ми будемо в обидвох класах.

```
isVoyageRoot
    ^ true
```

Рис. 3.53 Метод isVoyageRoot

Для початку можна не використовувати існуючу базу даних, для цього є клас *VOMemoryRepository*, однак надалі я буду використовувати реальне підключення через *VOMongoRepository.*

Для підключення нашого додатку до бази даних необхідно в одному з класів оголосити метод *initializeVoyageDB* на рівні класу:

```
initializeVoyageDB
    | repository |
    repository := VOMongoRepository database: 'AnimalsHouse'.
    repository enableSingleton.
```

Рис. 3.54 метод initializeVoyageDB

Оскільки ми хочемо мати лише одне підключення, для цього використовуємо метод *enableSingleton*.

Метод *reset* заново ініціалізує базу даних, це необхідно робити кожного разу, коли міняється «схема даних» у базі, до прикладу змінились змінні в класі Animal.

Оголосимо два метод на рівні класу:

```
reset
    self initializeVoyageDB

initialize
    self reset.
```

Рис. 3.55 Методи reset та initialize

3.6.3 Зберігаємо дані

Для того щоби зберегти наші об’єкти в базі нам необхідно зробити лише одне – надіслати повідомлення *save* до екземпляру нашого класу.

Модифікуємо методи для створення об’єктів Animal та Breed і додамо методи без додавання до бази, для тестування.

```
withName: nameValue age: ageValue sex: sexValue breed: breedValue weight: weightValue healthStatus:
healthStatusValue description: descriptionValue image: imageValue
   | instance |
   instance := self new.
   instance name: nameValue;
   age: ageValue;
   sex: sexValue;
   breed: breedValue;
   weight: weightValue;
   healthStatus: healthStatusValue;
   description: descriptionValue;
   image: imageValue;
   save.
   ^ instance.
```

Рис. 3.56 Модифікований метод для створення об’єкту класу Animal

```
withName: nameValue avgSize: avgSizeValue avgWeight: avgWeightValue avgLifespan: avgLifeSpanValue
avgDailySleep: avgDailySleepValue avgSpeed: avgSpeedValue
   | instance |
   self == Breed ifTrue: [
      ^ self error: 'Breed is an abstract class.  Make a concrete subclass.' ].
   instance := self getByName: nameValue.
   instance isNotNil
      ifFalse: [
         instance := self new.
         instance name: nameValue;
            avgSize: avgSizeValue;
            avgWeight: avgWeightValue;
            avgLifespan: avgLifeSpanValue;
            avgDailySleep: avgDailySleepValue;
            avgSpeed: avgSpeedValue.
         instance save.
         ].
      ^ instance.
```

Рис. 3.57 Модифікований метод для створення об’єкту класу Breed

```
testWithName: nameValue age: ageValue sex: sexValue breed: breedValue weight: weightValue healthStatus:
healthStatusValue description: descriptionValue image: imageValue
   | instance |
   instance := self new.
   instance name: nameValue;
   age: ageValue;
   sex: sexValue;
   breed: breedValue;
   weight: weightValue;
   healthStatus: healthStatusValue;
   description: descriptionValue;
   image: imageValue.
   ^ instance.
```

Рис. 3.58 Тестовий метод для створення об’єкту класу Animal

```
testWithName: nameValue avgSize: avgSizeValue avgWeight: avgWeightValue avgLifespan: avgLifeSpanValue
avgDailySleep: avgDailySleepValue avgSpeed: avgSpeedValue
   | instance |
   self == Breed ifTrue: [
      ^ self error: 'Breed is an abstract class.  Make a concrete subclass.' ].
   instance := self getByName: nameValue.
   instance isNotNil
      ifFalse: [
         instance := self new.
         instance name: nameValue;
            avgSize: avgSizeValue;
            avgWeight: avgWeightValue;
            avgLifespan: avgLifeSpanValue;
            avgDailySleep: avgDailySleepValue;
            avgSpeed: avgSpeedValue.
         ].
      ^ instance.
```

Рис. 3.59 Тестовий метод для створення об’єкту класу Breed

Для подальшої зручності рекомендую додати метод, що буде генерувати набір тестових даних та зберігати їх у базу.

3.6.4 Надсилаємо запити до бази даних

Щоб перевірити, чи існують у нас якісь дані, ми можемо надіслати повідомлення *count* до класу Animal чи Breed, що поверне нам кількість об’єктів даного типу в базі даних.

Для того, щоб їх отримати, необхідно надіслати повідомлення *selectAll* – для отримання усіх даних з таблиці, *selectOne: aBlock* – для отримання одного запису що підходить за умовою заданою у блоці коду, *selectMany: aBlock* – отримати набір даних які відповідають умові заданій в блоці.

Якщо вам необхідно видалити об’єкт із бази, для цього достатньо надіслати повідомлення *remove* до екземпляру об’єкта.

3.7 Вступ до Seaside

3.7.1 Додавання необхідних пакетів

Для повноцінної роботи з Seaside фреймворком, нам необхідно додати його до образу, це можна зробити додавши його окремо, або ж разом з кількома іншими пакетами, а саме bootstrap та bootstrap4, під час їх додавання Seaside та необхідні йому залежності будуть додані автоматично.

Для цього у Playground введіть наступні команди:

```
Metacello new
      baseline: #Bootstrap;
      repository:'github://astares/Seaside-Bootstrap;master/src';
      onConflictUseLoaded;
      load.

Metacello new
      baseline: #Bootstrap4;
      repository:'github://astares/Seaside-Bootstrap4;master/src';
      onConflictUseLoaded;
      load.
```

3.7.2 Запуск Zinc Server

У цьому розділі ми ознайомимось з Seaside та створимо наш перший компонент.

Seaside використовує *ZnZincServer* для того щоби запускати наш сайт локально. Для того щоби запустити його, необхідно відкрити вікно *Seaside Control Panel* у пункті *Library*.

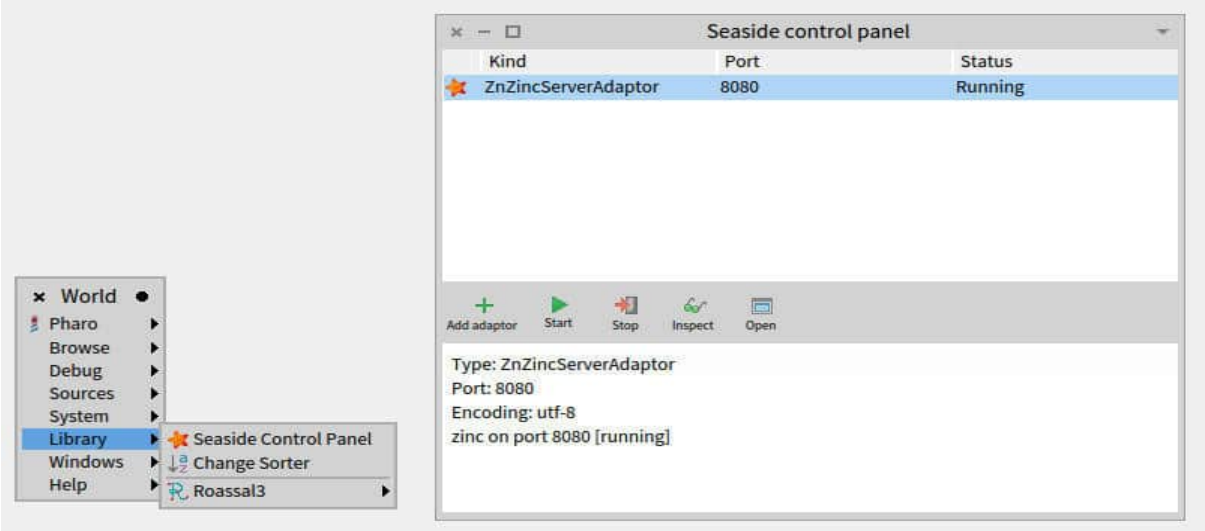

Рис. 3.60 Вікно контролю над сервером

Обравши перший запис натискаємо старт. Якщо ж у вас немає жодного адаптера в списку, створіть його та запустіть на потрібному вам порті, за замовчуванням це 8080.

Коли ви відкриєте браузер на сторінці localhost:8080 то побачите стандартне вікно запуску Seaside з набором прикладів та посиланнями на книжку, де ви можете дізнатись більше про Seaside.

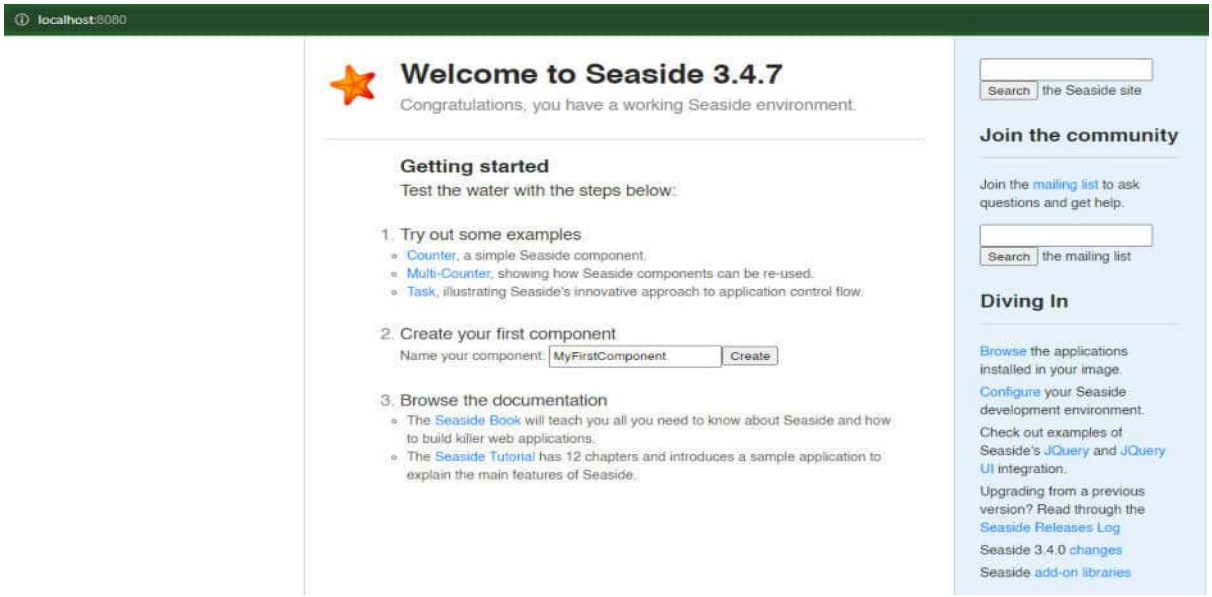

Рис. 3.61 Вступна сторінка Seaside

Також прописавши localhost:8080/browse ви попадете на сторінку з корисними посиланнями, включно з прикладами використання *Bootstrap*, це нам знадобиться в майбутньому.

3.7.3 Оголошення точки входу веб застосунку

Клас *AHApplicationRootComponent* буде точкою входу нашого веб-сайту, він буде кореневим компонентом застосунку та міститиме решту компонентів. Екземпляр цього класу містить змінну *main*, що є основним компонентом нашого застосунку. Наслідуємо ж ми нашу точку входу від класу *SBSComponent*, що дає нам можливість використовувати функціонал *Bootstrap4*. Для зручності всі веб компоненти розміщуємо в окремому пакеті, *AnimalsHouse-Components*

```
SBSComponent subclass: #AHApplicationRootComponent
   instanceVariableNames: 'main'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.62 Створення кореневого компонента сайту

Для того щоб кореневий компонент виконував свою функцію, нам потрібно перевизначити метод initialize на рівні класу. Зареєструємо наш додаток та додамо необхідні бібліотеки та залежності.

```
initialize
   | app |
   app := WAAdmin register: self asApplicationAt: 'AnimalsHouse'.
   app addLibrary: JQDeploymentLibrary;
   addLibrary: JQUiDeploymentLibrary;
   addLibrary: TBSDeploymentLibrary;
   addLibrary: SBSDevelopmentLibrary;
   addLibrary: SBSDeploymentLibrary;
   addLibrary: SBSFileLibrary.
```

Рис. 3.63 Додавання необхідних бібліотек до кореневого компоненту

Також на рівні класу необхідно додати метод *canBeRoot*, щоби вказати Seaside що цей компонент не є звичайним, а цілим застосунком.

```
canBeRoot
    ^ true.
```

Рис. 3.64 Метод визначення кореневого компоненту

Перевірити правильність виконаних дій можна прописавши localhost:8080/AnimalsHouse у адресний рядок браузера.

3.7.4 Візуалізація компонент

Основою відображення кожного компоненту є метод *RenderContentOn: html*, що вказує фреймворку, що саме цей компонент має відображати.

Оголосимо цей метод для *AHApplicationRootComponent*:

```
renderContentOn: html
    html text: 'Animals House'
```

Рис. 3.65 Метод renderContentOn для AHApplicationRootComponent

Якщо перезагрузити сторінку, то можна побачити, що на сайті з’явився щойно доданий напис.

Для того щоби веб сайт підтримував html5 нам необхідно перевизначити ще один метод, *updateRoot*:

```
updateRoot: anHtmlRoot
    super updateRoot: anHtmlRoot.
    anHtmlRoot beHtml5.
    anHtmlRoot title: 'AnimalsHouse'.
```

Рис. 3.66 Метод updateRoot

3.7.5 Архітектура веб-сайту

Наш веб-сайт буде складатись із головної сторінки, сторінки забраних тварин, сторінки відгуків та панелі адміністратора.

На головній сторінці ми розмістимо карусель із фото, форму про знаходження бездомної тварини, список тварин, що живуть у притулку, та інструменти фільтрування списку.

На сторінці забраних тварин ми розмістимо список тварин, що були забрані з притулку, статистику про кількість тварин, які успішно знайшли власників.

На сторінці відгуків буде список відгуків від користувачів про цей притулок та форма для подачі відгуку.

На панелі адміністратора ми розмістимо список поданих заявок про знаходження тварини та додаймо можливість реєстрації цієї тварини в базі нашого притулку.

Також спільним елементом у нас буде заголовок сайту з кнопками переходу на інші сторінки, логуванням та реєстрацією.

3.8 Створення веб компонент головної сторінки

Тож перейдемо до процесу створення наших компонент і почнемо ми з *AHScreenComponent*.

3.8.1 AHScreenComponent

Цей компонент буде основою для всіх наших сторінок та міститиме спільні компоненти такі, як *header*, що буде наявний на всіх сторінках.

```
SBSComponent subclass: #AHScreenComponent
   instanceVariableNames: 'header'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.67 Створення компоненти екрану

Для відображення вмісту компоненти нам необхідно оголосити метод *renderContentOn*:

```
renderContentOn: html
   html text: 'This is Screen Component'.
```

Рис. 3.68 Реалізація методу renderContentOn

Оновимо нашу сторінку та побачимо, що нічого не змінилось. Це через те, що *ScreenComponent* ще не підключений до нашого застосунку. Давайте це змінимо.

У *AHApplicationRootComponent* оголосимо initialization метод, та присвоюємо змінній main значення *AHScreenComponent*.

Після цього змінимо метод *renderContentOn* кореневого компонента до наступного вигляду:

```
renderContentOn: html
   html div
      style: 'background-color: #EAF6FF; height: 100%; width: 100%;';
      with: main.
```

Рис. 3.69 Модифікований метод renderContentOn кореневого компоненту

Таким чином, ми створили блок із заданими стилями, та зобразили компоненту *AHScreenComponent* у ньому.

В подальшому ми хочемо мати постійну структуру сторінки, і щоб уникнути постійного дублювання коду, змінимо реалізацію методу *renderContentOn* компоненти *AHScreenComponent* наступним чином:

```
renderContentOn: html

   html column
      style: 'display: flex; flex-direction: column; min-height: 100vh';
      with: [
         html row
            style: 'min-height: 8vh';
            with: [ html render: header ].
         html row
            style: 'min-height: 86vh';
            with: [
               html column
                  style: 'justify-content: center;';
                  extraLargeSize: 1;
                  with: [ self renderAdOn: html ].
               html column
                  extraLargeSize: 10;
                  with: [ self renderMainContentOn: html ].
               html column
                  style: 'justify-content: center;';
                  extraLargeSize: 1;
                  with: [ self renderAdOn: html ] ].
         html row
            style: 'min-height: 6vh';
            with: [ self renderFooterOn: html ] ]
```

Рис. 3.70 Модифікований метод renderContentOn компоненти екрану

Тут ми використовуємо доступні нам методи стилізації які були додані пакетом bootstrap, щоб зробити сторінку гнучкою та мати різні розміри елементів на екранах різного розміру.

Структура нашого сайту буде наступна: спочатку заголовок з набором кнопок для навігації, низ сторінки, та основна частина, де по боках будуть розміщені банери для реклами, а посередині власне вміст сторінки, яка буде відкрита.

Таким чином для усіх подальших компонент-сторінок нам не потрібно буде прописувати цю структуру, а лише реалізувати метод *renderMainContentOn* з вмістом сторінки.

Реалізуємо необхідні методи для цього *renderMainContentOn, renderAdOn* та *renderFooterOn*.

```
renderMainContentOn: html

   html text: 'Main content'

renderAdOn: html

   html container
      style: 'height: 100%; background-color: #6F9DC8;';
      with: [ html label with: 'Реклама' ]

renderFooterOn: html

   html row
      style:
         'width: 100%;background-color: #355070; margin-top: 1%; min-height: 25px;';
      with: [ html label: 'footer' ]
```

Рис. 3.71 Методи для відображення вмісту у компоненті екрану

В даний момент нам у нас не створено об’єкту header який ми намагаємось відобразити, тож у наступному розділі ми це виправимо.

3.8.2 AHHeaderComponent.

Цей компонент є доволі простим та буде використано у всіх наших сторінках, тож після його оголошення нам потрібно додати трішки коду до *AHScreenComponent*.

```
SBSComponent subclass: #AHHeaderComponent
    instanceVariableNames: 'component'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.72 Створення компоненти заголовку

Додамо метод *initialize* та *createHeaderComponent* до *AHScreenComponent*, для створення заголовоку змінна component буде вказувати на те, який компонент створює цей заголовок. До AHHeaderComponent додамо метод на рівні класу *from:* який приймає компонент як аргумент.

```
initialize
    super initialize.
    header := self createHeaderComponent.

createHeaderComponent
    ^ AHHeaderComponent from: self

from: aComponent
    ^ self new
        component: aComponent;
        yourself
```

Рис. 3.73 Методи необхідні для роботи компоненти заголовку

Для того щоби відобразити заголовок, нам традиційно необхідно оголосити метод *renderContentOn* та *renderBrandOn*.

```
renderContentOn: html

    html navigationBar
        style:
            'width:100%; font: 65px Montserrat; color: white; background-color: #355070';
        with: [
            self renderBrandOn: html.
    ]
```

Рис. 3.74 Метод renderContentOn для заголовку

```
renderBrandOn: html
    html tbsNavbarHeader: [
        html tbsNavbarBrand
            style: 'font-size: large;
                color: white;
                padding: 15px;
                text-align: center;';
            url: self application url;
            with: 'Дім Тварин' ].
```

Рис. 3.75 Метод для відображення бренду

Ці два методи дозволяють нам відобразити заголовок, а з допомогою стилів ми можемо зробити його більш приємним на вигляд.

Після цього ми можемо побачити, що наш заголовок успішно відобразився.

3.8.3 Ієрархія та відношення компонентів у Seaside.

Для того, щоб фреймворк знав, що *header* є частиною компоненти *AHScreenComponent*, нам необхідно додати метод *children*, що вертає список із компонентами, у даному випадку – header, такий ж метод необхідно додати до кореневого компоненту, такі методи необхідно буде додавати для усіх наших компонент.

```
children
    ^ { header }

children
    ^ { main }
```

Рис. 3.76 Методи children для позначення ієрархії компонент

3.8.4 AHMainScreenComponent

Оскільки *AHScreenComponent* задумувався як абстрактний предок усіх сторінок застосунку, пора створити конкретну реалізацію однієї зі сторінок, назвемо її *AHMainScreenComponent*. Назва компоненти говорить сама за себе, ця компонента буде відповідати за основну головну сторінку сайту, де буде розміщено більшість функціоналу.

```
AHScreenComponent subclass: #AHMainScreenComponent
    instanceVariableNames: 'grid'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.77 Створення компоненти головного екрану

Для початку оголосимо клас зі змінною *grid*, який буде містити наш список тварин, який ми створимо в наступному розділі. Також варто помітити, що так як *AHMainScreenComponent* унаслідується від *AHScreenComponent* то він автоматично отримує змінну *header*.

Створимо повідомлення *renderMainContentOn*:

```
renderMainContentOn: html

   html column
      style:
         'display: inline-flex; flex-direction: column; vertical-align: top;';
      extraLargeSize: 12;
      with: [
         html row with: [
            html column
               extraLargeSize: 10;
               extraLargeOffset: 1;
               mediumSize: 10;
               mediumOffset: 1;
               with: grid ] ]
```

Рис. 3.78 Метод для відображення вмісту сторінки

Тут ми використали інструменти, які надає нам bootstrap, а саме grid систему, яка дозволяє створювати інтерфейси, що реагують на розмір екрану, та підлаштовуються для кращого відображення.

Для цього ми спочатку створили одну загальну колонку та додали в неї перший рядок, що міститиме наш список.

3.8.5 AHAnimalsGrid

Одним із найважливіших компонентів нашого застосунку буде список тварин у форматі «галереї карток». Оскільки таких галерей буде кілька, створимо абстрактний клас галереї *AHGeneralGrid* та оголосимо два методи, що будуть спільними для всіх галерей: *getData* та *renderContentOn*:, а також метод на рівні класу для створення об’єкту *from*: аналогічно до *AHHeaderComponent*

```
SBSComponent subclass: #AHGeneralGrid
   instanceVariableNames: 'sorting component'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.79 Створення компоненти галереї

Метод *getData* буде реалізовано кожною галереєю по-різному, отримуючи різні дані та картки, однак у кожній він точно буде, тож перекладемо відповідальність визначення цього методу на класи нащадки:

```
getData
    self subclassResponsibility.
```

Рис. 3.80 Метод для отримання даних в галереї

В свою чергу метод *renderContentOn* буде однаковий для всіх. Компонент містить колонку, що буде розтягнута на всю доступну ширину, та містить багато «карток» тварин, кожна картка займатиме четвертину, третину чи половину рядка, залежно від розміру екрану.

```
renderContentOn: html

   html column
      style:
         'background: #6F9DC8; paddign: 0px; height: inherit; border: 1px solid #355070;';
      extraLargeSize: 12;
      mediumSize: 12;
      smallSize: 12;
      with: [
         html row
            style: 'overflow-y:auto; overflow-x: hidden; max-height: 800px;';
            with: [
               self getData collect: [ :a |
                     html column
                        extraLargeSize: 3;
                        largeSize: 3;
                        mediumSize: 4;
                        smallSize: 6;
                        with: a ] ] ]
```

Рис. 3.81 Метод renderContentOn для компоненти галереї

Тепер, створимо клас *AHAnimalsGrid*, та визначимо в ньому метод *getData*, що буде отримувати всіх тварин із бази та створювати картки для них.

```
getData
   ^ Animal selectAll collect: [ :animal | AHAnimalCard new animal: animal ]
```

Рис. 3.82 Реалізація методу отримання даних для галереї тварин

Однак для правильної роботи нам бракує класу карток для відображення тварин. Тож давайте це виправляти. Створимо клас *AHBasicCard* з змінними *component* та *data.* Він буде виконувати роль базового класу для усіх карток.

```
SBSComponent subclass: #AHBasicCard
   instanceVariableNames: 'component data'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.83 Створення компоненти картки

Додамо метод from: withData: на рівні класу, який буде створювати картку з даними які будуть передані в картку.

```
from: aComponent withData: data

    ^ self new
        component: aComponent;
        data: data yourself
```

Рис. 3.84 Метод створення картки відносно компоненти

Також нам необхідно відображати фото тварини, отримуючи його з файлової системи, за збереженим шляхом у класі Animal. Для цього нам необхідно створити наступні методи: *readImageFrom:, saveImage:, renderAnimalImageOn: with:.*

```
readImageFrom: fullPath

    | file content stream |
    file := fullPath asFileReference.
    stream := file binaryReadStream.
    content := ''
              , (ZnBase64Encoder new encode: stream contents).
    stream close.
    ^ content
```

Рис. 3.85 Метод для зчитування зображення за переданим шляхом

```
saveImage: imageObject

    | filepath |
    filepath := FileSystem disk workingDirectory / imageObject fileName.
    filepath binaryWriteStream
       nextPutAll: imageObject rawContents;
       close.
    ^ filepath pathString
```

Рис. 3.86 Метод збереження зображення

```
renderAnimalImageOn: html with: image

   html tbsImage
      beResponsive;
      style:
         'display: flex; max-width: 100%; width: auto; height: 200px; object-fit: cover;';
      url: (self readImageFrom: image)
```

Рис. 3.87 Метод для відображення зображення

Далі створимо новий клас, *AHAnimalCard зі зм*інною *animal*, який містить у собі інформацію про конкретну тварину, та двома змінними, що будуть потрібні у випадку, коли тварину хочуть забрати.

```
AHBasicCard subclass: #AHAnimalCard
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.88 Створення компоненти картки тварини

Тепер можна додати відображення картки через метод *renderContentOn*.

```
renderContentOn: html

    html card
        style: 'margin: 15px; border: 2px solid #355070;';
        with: [
            self renderAnimalImageOn: html with: data image.
            html cardBody: [
                  html cardTitle level5 with: data name.
                  html cardSubtitle
                      level6;
                      mutedText;
                      with: data sex , ',' , data age printString.
                  html formButton bePrimary
                      dataToggle: 'modal';
                      dataTarget: '#TakeAnimalModal' , data id;
                      with: 'Забрати додому' ].
            self renderModalOn: html ]
```

Рис. 3.89 Метод renderContentOn для картки тварини

Тут ми використали метод *card* що доступний у *Bootstrap4* для генерації картки з фото та описом. Також на картку ми додали кнопку, що буде викликати модальне вікно, та просити заповнити форму для того, щоб стати власником тварини. Власне реалізацію форми, що буде в модальному вікні, ми винесли в окремий метод, *renderModalOn*.

```
renderModalOn: html

    html modal
        id: 'TakeAnimalModal' , data id;
        with: [
            html modalDialog with: [
                    html modalContent: [
                            html modalHeader: [
                                    html modalTitle
                                        level5;
                                        with: 'Забрати тварину'.
                                    html modalCloseButton ].
                            self renderModalBodyOn: html ] ] ]
```

Рис. 3.90 Метод для відображення модального вікна

```
renderModalBodyOn: html

    html modalBody
        style: 'color: #000000; font-size: 16px';
        with: [
            html tbsImage
                style:
                    'border: 3px solid black; max-width: 50%; margin-left: 25%; margin-right: 25%; margin-bottom: 3%;';
                url: (self readImageFrom: data image).
            self renderAnimalDataOn: html ].
    html modalFooter
        style: 'justify-content: space-evenly;';
        with: [
            html form: [
                html formButton
                    beSecondary;
                    dataDismiss: 'modal';
                    with: 'Закрити'.
                html formButton
                    bePrimary;
                    callback: [ self save ];
                    with: 'Забрати' ] ]
```

Рис. 3.91 Метод для відображення тіла модального вікна

```
renderAnimalDataOn: html

    html container
        style: 'margin-left: 10%; width:50%';
        with: [
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Кличка:'.
                    html paragraph: data name ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Порода:'.
                    html paragraph: data breed name ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Стать:'.
                    html paragraph: data sex ].
            html row
                style: 'justify-content: start; ';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Вік:'.
                    html paragraph: data age ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Вага:'.
                    html paragraph: data weight ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Здоров'я:'.
                    html paragraph: data healthStatus ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Опис:'.
                    html paragraph: data description ] ]
```

Рис. 3.92 Метод для відображення інформації про тварину

Також додамо метод *id* до класу Animal

```
id

    ^ self voyageId value printStringHex
```

Рис. 3.93 Метод для отримання унікального ідентифікатора тварини

Для того, щоб зберегти заявку на отримання тварини, нам потрібно створити певний запис у базі даних, для цього створимо нову модель даних – *AHWantAnimal*, зі змінними екземпляру *animal, contactName* та *contactPhone.* Додамо методи *isVoyageRoot*, та метод для створення цього об’єкту.

```
Object subclass: #AHWantAnimal
   instanceVariableNames: 'contactName contactPhone animal'
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'
```

Рис. 3.94 Створення моделі заявки на отримання тварини

```
withAnimal: animal contactName: nameValue contactPhone: phoneValue
   | instance |
   instance := self new.
   instance animal: animal;
            contactName: nameValue;
            contactPhone: phoneValue;
            save.
   ^ instance.
```

Рис. 3.95 Метод для створення об’єкту заявки

Тепер додамо до *AHAnimalCard*, два методи: *reset*, що буде видаляти дані зі змінних, якщо форма не була надіслана, та save, який власне буде створювати об’єкт класу *AHWantAnimal* з отриманих даних.

```
reset
   contactName := ''.
   contactPhone := ''.
```

Рис. 3.96 Метод для очищення даних

```
save
   AHWantAnimal withAnimal: animal contactName: contactName contactPhone: contactPhone.
   self reset.
```

Рис. 3.97 Метод для збереження заповненої форми

Таким чином наша картка повністю готова, у результаті в нас має вийти така картка, та модальне вікно:

Рис. 3.98 Форма для отримання тварини

Рис. 3.99 Вигляд картки

Після цього редагуємо метод initialize класу *MainScreenComponent* та отримаємо готову галерею тварин.

```
initialize

    super initialize.
    grid := AHAnimalsGrid from: self
```

Рис. 3.100 Модифікований метод initialize

3.8.6 Фільтрація в галереї.

Для того, щоб відфільтрувати картки в галереї, нам необхідно задати відповідні критерії та створити інтерфейс вибору цих критеріїв. Для цього створюємо клас *AHSortingComponent* зі змінними, що відповідають нашим критеріям, а саме: вид, стать та порода.

```
SBSComponent subclass: #AHSortingComponent
   instanceVariableNames: 'selectedSpeciesName selectedSex selectedBreedName'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.101 Створення компоненти сортування галереї

Першим ділом створюємо методи для отримання та присвоєння значення змінним об’єкту. Усі вони стандартні за винятком задання породи

```
selectSpecies: species
   selectedSpeciesName = species ifTrue: [ ^ self ]
      ifFalse: [
         selectedSpeciesName := species.
         selectedBreedName := (Species breedsOf: species) asArray at: 1.
      ].
```

Рис. 3.102 Метод присвоєння вибраного виду

Цей метод використовує нове повідомлення класу *Species*, що дозволяє отримати список порід відповідно до виду.

```
breedsOf: aSpecies

   | speciesClass |
   speciesClass := aSpecies = self dog
                     ifTrue: [ DogBreed ]
                     ifFalse: [ CatBreed ].
   ^ speciesClass selectAll collect: [ :breed | breed name ]
```

Рис. 3.103 Метод отримання порід в залежності від вибраного виду

Далі визначимо ще кілька методів:

```
initialize

    super initialize.
    selectedSpeciesName := 'Вид'.
    selectedSex := 'Стать'.
    selectedBreedName := 'Порода'
```

Рис. 3.104 Метод initialize для компоненти сортування

```
renderContentOn: html

    html row
        style:
            'display: flex; margin: 0; justify-content: space-between; width: 100%';
        with: [
            html div
                style:
                    'display: inline-flex; justify-content: flex-start; padding: 10px; width: 50%';
                with: [
                    html dropdown
                        style: 'margin-right: 2%;';
                        with: [
                            html formButton
                                bePrimary;
                                beLarge;
                                dropdown;
                                dataToggle: 'dropdown';
                                with: selectedSpeciesName.
                            html dropdownMenu
                                attributeAt: 'aria-labelledby' put: 'navbarDropdown';
                                with: [
                                    (Species list collect: [ :species |
                                            self renderSpeciesLinkOn: html with: species ]) addFirst:
                                        'Вид' ] ].
                    html dropdown
                        style: 'margin-right: 2%;';
                        with: [
                            html formButton
                                bePrimary;
                                beLarge;
                                dropdown;
                                dataToggle: 'dropdown';
                                with: selectedSex.
                            html dropdownMenu
                                attributeAt: 'aria-labelledby' put: 'navbarDropdown';
                                with: [
                                    (Sex list collect: [ :sex |
                                            self renderSexLinkOn: html with: sex ]) addFirst: 'Стать' ] ].
                    html dropdown
                        style: 'margin-right: 2%;';
                        with: [
                            html formButton
                                class: 'disabled' if: selectedSpeciesName = 'Вид';
                                bePrimary;
                                beLarge;
                                dropdown;
                                dataToggle: 'dropdown';
                                with: selectedBreedName.
                            html dropdownMenu
                                attributeAt: 'aria-labelledby' put: 'navbarDropdown';
                                with: [
                                    ((Species breedsOf: selectedSpeciesName) collect: [ :breed |
                                            self renderBreedLinkOn: html with: breed ]) addFirst:
                                        'Порода' ] ] ].
            html div
                style:
                    'justify-content: end; display: inline-flex;padding: 10px; width: 50%';
                with: [
                    html form: [
                            html formButton
```

Рис. 3.105 Метод renderContentOn для компоненти сортування

```
bePrimary;
beLarge;
callback: [ self clearFilters ];
with: ' Очистити фільтри' ] ] ]
```

Рис. 3.106 Продовження методу renderContentOn для компоненти сортування

Та три методи, що створюють випадаюче меню для кожної зі змінних:

```
renderBreedLinkOn: html with: breed
   html dropdownItem
      class: 'active' if: breed = self selectedBreedName;
      callback: [ self selectBreed: breed ]; with: breed.
renderSexLinkOn: html with: sex
   html dropdownItem
      class: 'active' if: sex = self selectedSex;
      callback: [ self selectSex: sex ]; with: sex.

renderSpeciesLinkOn: html with: species
   html dropdownItem
      class: 'active' if: species = self selectedSpeciesName;
      callback: [ self selectSpecies: species ]; with: species.
```

Рис. 3.107 Методи для відображення елементів вибору

Також додамо метод для повернення фільтру до початкового стану *clearFilters.*

```
clearFilters

   selectedSpeciesName := 'Вид'.
   selectedSex := 'Стать'.
   selectedBreedName := 'Порода'
```

Рис. 3.108 Метод для очищення вибраних параметрів для сортування

Після цього додамо компоненту сортування до змінних компоненти загальної галереї *AHGeneralGrid* та змінюємо метод *renderContentOn* цього ж класу, додаючи сортування як опціональний компонент галереї.

```
SBSComponent subclass: #AHGeneralGrid
   instanceVariableNames: 'sorting component'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.109 Модифікування компоненти галереї

```
renderContentOn: html

    html column
        style:
            'background: #6F9DC8; paddign: 0px; height: inherit; border: 1px solid #355070;';
        extraLargeSize: 12;
        mediumSize: 12;
        smallSize: 12;
        with: [
            sorting isNotNil ifTrue: [
                  html row
                      style: 'background: #2e608f';
                      with: sorting ].
            html row
                style: 'overflow-y:auto; overflow-x: hidden; max-height: 800px;';
                with: [
                    self getData collect: [ :a |
                          html column
                              extraLargeSize: 3;
                              largeSize: 3;
                              mediumSize: 4;
                              smallSize: 6;
                              with: a ] ] ]
```

Рис. 3.110 Модифікований метод renderContentOn для галереї

Далі потрібно змінити метод *getData* у *AHAnimalsGrid,* застосувавши фільтрацію з вибраними параметрами в компоненті сортування.

```
getData

    ^ (Animal selectAll select: [ :animal |
          (animal sex = sorting selectedSex or:
              sorting selectedSex = 'Стать') and:
             ((sorting selectedSpeciesName = animal animalSpecies or:
                  sorting selectedSpeciesName = 'Вид') and:
                 (sorting selectedBreedName = animal breed name or:
                     sorting selectedBreedName = 'Порода')) ]) collect: [
        :filteredAnimal |
        AHAnimalCard from: self component withData: filteredAnimal ]
```

Рис. 3.111 Реалізація методу getData з врахуванням сортування

І останній крок – створити об’єкт *sorting* на етапі ініціалізації *AHAnimalsGrid*:

```
initialize
    super initialize.
    sorting := AHSortingComponent new.
```

Рис. 3.112 Метод initialize для галереї тварин

3.8.7 Карусель фотографій

Bootstrap дає можливість легко додавати вже готові, складні компоненти. Розглянемо це на прикладі каруселі для фото. Створимо компоненту *AHCarouselPhotos* та оголосимо метод renderContentOn

```
SBSComponent subclass: #AHCarouselPhotos
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.113 Створення компоненти каруселі

```
renderContentOn: html

   html carousel
      style: 'height: 80%; width: 50%;';
      id: 'example';
      slide;
      with: [
         html carouselInner with: [
               self renderImageOn: html with: self firstPhoto active: true.
               self renderImageOn: html with: self secondPhoto active: false.
               self renderImageOn: html with: self thirdPhoto active: false ].
         html carouselControlPreviousFor: 'example'.
         html carouselControlNextFor: 'example' ].
   html script: '$(".carousel").carousel()'
```

Рис. 3.114 Метод renderContentOn для відображення каруселі

```
renderImageOn: html with: url active: boolean

   ^ html carouselItem
        style: 'display:flex; justify-content: center; background: white;';
        class: 'active' if: boolean;
        with: [
           html tbsImage
              style: 'width: auto; height: 300px;';
              url: url ]
```

Рис. 3.115 Метод для відображення зображення каруселі

Методи *firstPhoto*, *secondPhoto*, *thirdPhoto* мають повертати посилання на фото, записане рядком символів. Якщо ж у вас більше ніж 3 фото, рекомендується замість окремих методів для другого та третього фото повертати масив посилань.

І це все, що потрібно зробити на стороні компоненти, доволі просто. Тепер давайте відобразимо цю компоненту у *AHMainScreenComponent*, додамо змінну

carousel та створимо його в методі *initialize* та додамо рядок із каруселлю в методі *renderContentOn*.

```
initialize

    super initialize.
    carousel := AHCarouselPhotos new.
    grid := AHAnimalsGrid from: self
```

Рис. 3.116 Модифікований метод initialize для головного екрану

```
renderMainContentOn: html

    html column
        style:
            'display: inline-flex; flex-direction: column; vertical-align: top;';
        extraLargeSize: 12;
        with: [
            html row
                style: 'justify-content: center; max-height: 300px';
                with: carousel.
            html row with: [
                html column
                    extraLargeSize: 10;
                    extraLargeOffset: 1;
                    mediumSize: 10;
                    mediumOffset: 1;
                    with: grid ] ]
```

Рис. 3.117 Модифікований метод renderMainContentOn для головного екрану

3.8.8 Форма – повідомити про тварину.

Останнім необхідним елементом на головному екрані є форма, яка дозволить повідомити притулок про бездомну тварину. Розмістимо цю форму в модальному вікні, що буде відкриватись після натискання кнопки на головному екрані.

Для початку створимо компоненту AHBasicModal що буде базовим для модальних вікон та міститиме вже знайомий нам метод from: для створення. Наступним створимо *AHFoundAnimalModal зі зм*інними екземпляру: контактний номер, ім’я контактної особи, фото тварини, вид тварини, опис, та місцезнаходження.

```
AHBasicModal subclass: #AHFoundAnimalModal
    instanceVariableNames: 'contactName contactPhone photo animalSpecies description place'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис 3.118 Створення компоненти модального вікна про знайдену тварину

У повідомленні *renderContentOn* пропишемо кнопку, та модальне вікно, що має відкриватись на її натискання, основне наповнення модального вікна виносимо в окремий метод *renderFormBodyOn*

```
renderContentOn: html

   html container with: [
      html formButton
         style: 'height: 4vh; font-size: 16px; border-radius: 10px';
         bePrimary;
         dataToggle: 'modal';
         dataTarget: '#FoundAnimalModal';
         with: 'Повідомити про тварину'.
      html modal
         id: 'FoundAnimalModal';
         with: [
            html modalDialog with: [
                  html modalContent: [
                        html modalHeader: [
                              html modalTitle
                                 level5;
                                 with: 'Повідомити про тварину'.
                              html modalCloseButton ].
                        self renderFormBodyOn: html ] ] ] ]
```

Рис. 3.119 Метод renderContentOn для модального вікна про знайдену тварину

```
renderFormBodyOn: html

   html modalBody: [
      html form
         style:
            'color: black; font-size: medium; display: flex; flex-direction: column';
         multipart;
         with: [
            html formGroup with: [
                  html label: 'Photo'.
                  html fileUpload callback: [ :value | photo := value ] ].
            html formGroup with: [
                  html label: 'Place'.
                  html textInput
                     formControl;
                     callback: [ :value | place := value ] ].
            html formGroup with: [
               html label: 'Description'.
               html textInput
                  formControl;
                  callback: [ :value | description := value ] ].
            html formGroup with: [
               html label: 'Name'.
               html textInput
                  formControl;
                  callback: [ :value | contactName := value ] ].
            html formGroup with: [
               html label: 'Contact Phone'.
               html telephoneInput
                  formControl;
                  callback: [ :value | contactPhone := value ] ].
            html modalFooter: [
               html tbsButton
                  beDefault;
                  dataDismiss: 'modal';
                  callback: [ self reset ];
                  with: 'Close'.
               html tbsSubmitButton
                  bePrimary;
                  callback: [ self save ];
                  with: 'Send application' ] ] ]
```

Рис. 3.120 Метод відображення вмісту модального вікна

У методі *reset* ми видаляємо значення, що були збережені в змінних. А в методі save ми створюємо новий екземпляр класу *AHFoundAnimal* що буде зберігатись у базі даних.

```
Object subclass: #AHFoundAnimal
   instanceVariableNames: 'photo description place contactName contactPhone animalSpecies'
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'
```

Рис. 3.121 Створення моделі про знайдену тварину

Генеруємо методи доступу до змінних екземпляру та додаємо метод *isVoyageRoot* на рівні класу для того щоби зберігати ці дані окремою таблицею. Також додаємо метод для створення об’єктів цього класу та збереження в базу.

```
withSpecies: species contactName: contactName contactPhone: phone animalPhoto: photoValue
animalDescription: description animalPlace: place
   | instance |
   instance := self new.
   instance animalSpecies: species; contactName: contactName; contactPhone: phone;
   photo: photoValue; description: description; place: place.
   instance save.
   ^ instance.
```

Рис. 3.122 Метод для створення об’єкту класу AHFoundAnimal

Власне його ми й будемо використовувати в методі *save* класу *FoundAnimalModal*.

```
save

   AHFoundAnimal
      withSpecies: animalSpecies
      contactName: contactName
      contactPhone: contactPhone
      animalPhoto: (self saveImage: photo)
      animalDescription: description
      animalPlace: place
```

Рис. 3.123 Метод для зберігання заповненої форми

```
saveImage: image

    | filepath |
    filepath := FileSystem disk workingDirectory / image fileName.
    filepath binaryWriteStream
       nextPutAll: image rawContents;
       close.
    ^ filepath pathString
```

Рис. 3.124 Метод для зберігання зображення

Фото, що буде надіслано користувачем, ми записуємо на диск, а шлях до нього зберігаємо в колонці *photo* класу *AHFoundAnimal*.

Останнє, що варто зробити, це додати кнопку на головний екран та кілька написів, щоб допомогти користувачу зорієнтуватися у функціоналі компонент.

У результаті метод *renderContentOn* класу *AHMainScreenComponent* має виглядати наступним чином

```
renderMainContentOn: html

   html column
      style:
         'display: inline-flex; flex-direction: column; vertical-align: top;';
      extraLargeSize: 12;
      with: [
         html row
            style: 'justify-content: center; max-height: 300px';
            with: carousel.
         html row
            style:
               'display:flex; flex-direction: column; align-items: center;';
            justifyContentCenterVeryLarge;
            with: [
               html heading
                  style: 'padding-top: 10px; marigin-bottom: 0px;';
                  level2;
                  with: 'Знайшли тварину? Допоможіть нам знайти їй домівку:'.
               html column
                  style: 'text-align: center';
                  extraLargeSize: 2;
                  with: (AHFoundAnimalModal from: self).
               html heading
                  level2;
                  with: 'Станьте власником чотирилапого чуда!' ].
         html row with: [
            html column
               extraLargeSize: 10;
               extraLargeOffset: 1;
               mediumSize: 10;
               mediumOffset: 1;
               with: grid ] ]
```

Рис. 3.125 Модифікований метод для відображення вмісту головної сторінки

Такий вигляд має наша головна сторінка в результаті.

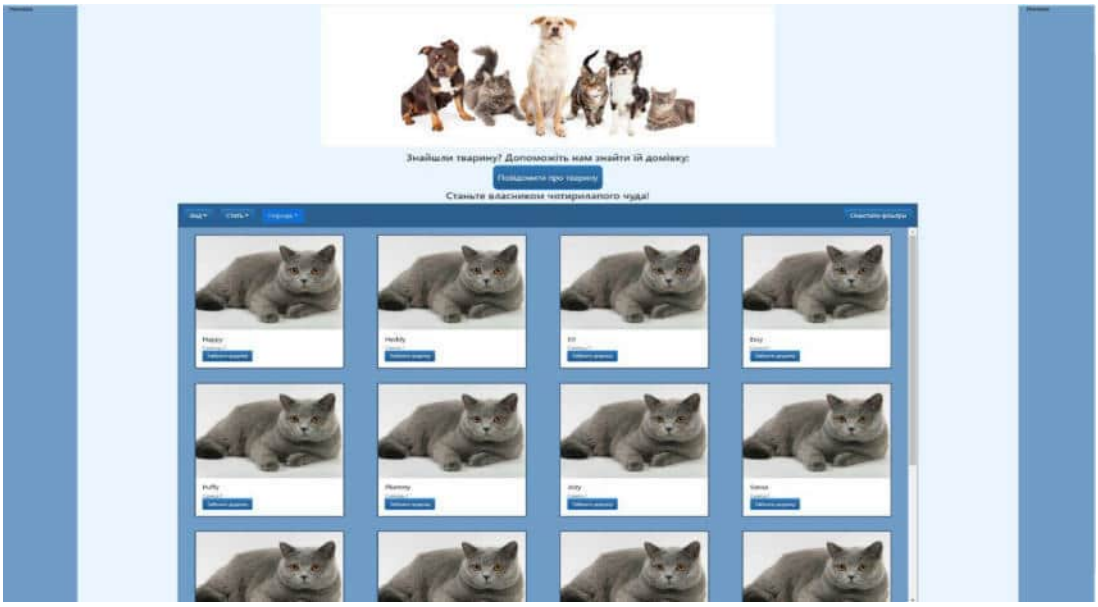

Рис. 3.126 Вигляд головної сторінки

3.9 Автентифікація та сесії

3.9.1 Перехід на сторінку адміністратора

Для повноцінної роботи сайту необхідна роль адміністратора, який буде реагувати на форми, які заповнює користувач. Для цього додамо спосіб логування в систему як адміністратор. Першим нашим кроком буде створення сторінки, яку він буде бачити. Такою сторінкою буде компонент *AHAdminComponent*, що унаслідується від компоненту *AHScreenComponent.*

```
AHScreenComponent subclass: #AHAdminComponent
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.127 Створення компоненти адміністратора

Додаємо метод *renderMainContentOn*, поки що вона містить лише заголовок.

```
renderMainContentOn: html

   html column
      style: 'min-height: 800px';
      extraLargeSize: 12;
      with: [
         html heading: 'Сторінка адміністратора'.
         html horizontalRule ]
```

Рис. 3.128 Метод відображення вмісту сторінки адміністратора

Для того щоб могти залогуватись як адміністратор, необхідно додати кнопку в заголовку сайту. Для цього змінимо метод *renderContentOn* в класі *AHHeaderComponent,* додавши рядок із методом *renderButtonsOn*.

```
renderContentOn: html

    html navigationBar
        expandLarge;
        style:
            'width:100%; font: 65px Montserrat; color: white; background-color: #355070';
        with: [
            self renderBrandOn: html.
            self renderButtonsOn: html ]
```

Рис. 3.129 Модифікований метод renderContentOn заголовку сайту

```
renderButtonsOn: html
    self renderAdminButtonOn: html.
```

Рис. 3.130 Метод для відображення кнопок заголовку

```
renderAdminButtonOn: html

    html form: [
        html tbsNavbarButton
            tbsPullRight;
            style: self buttonStyle;
            callback: [ component goToAdminView ];
            with: [
                html tbsGlyphIcon iconListAlt.
                html text: ' Admin View' ] ]
```

Рис. 3.131 Метод для відображення кнопки переходу на сторінку адміністратора

Також додамо метод *buttonStyle* до *AHHeaderComponent*, який використовуватимемо для стилізації усіх кнопок заголовку.

```
buttonStyle

    ^ 'border-radius: 5px;height: 30px; width: fit-content;margin: 3px;'
```

Рис. 3.132 Метод для отримання стилю кнопок

Після оновлення сторінки ми побачимо, що кнопка з’явилась, однак її натискання видає помилку, адже нам необхідно визначити метод *goToAdminView* в компоненті, яка створила цей заголовок.

Метод *goToAdminView* має бути визначений у класі *AHMainScreenComponent*. Для цього спочатку створимо цей метод у класі AHScreenComponent який буде делегувати реалізацію цього методу, своїм нащадкам.

```
goToAdminView

    self subclassResponsibility
```

Рис. 3.133 Метод для переходу на сторінку адміністратора

А у AHMainScreenComponent реалізація методу буде наступна:

```
goToAdministrationView
    self call: AHAdminComponent new.
```

Рис. 3.134 Реалізація методу переходу на сторінку адміністратора

Перевіряємо й бачимо, що при натисканні кнопки в нас змінюється компонента, що відображається в даний момент.

3.9.2 Повернення на головну сторінку

В даний момент, ми маємо можливість перейти на сторінку адміністратора, однак не можемо повернутися назад. Для цього нам потрібно додати додаткову кнопку, однак вона нам необхідна лише тоді, коли відображається сторінка адміністратора. Саме тому нам необхідно створити клас, що походить від класу *AHHeaderCompopnent*, та перевизначити метод *renderButtonsOn, щоби м*ати інший перелік кнопок на цій сторінці.

```
AHHeaderComponent subclass: #AHAdminHeaderComponent
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.135 Створення компоненти заголовку для сторінки адміністратора

```
renderButtonsOn: html
    html form: [
        self renderDisconnectButtonOn: html.]
```

Рис. 3.136 Метод для відображення кнопок

```
renderDisconnectButtonOn: html

    html tbsNavbarButton
        tbsPullRight;
        style: self buttonStyle;
        callback: [
            self session reset.
            component goToMainView ];
        with: [
            html text: 'Вийти'.
            html tbsGlyphIcon iconLogout ]
```

Рис. 3.137 Метод для відображення кнопки виходу

У класі *AHAdminComponent* перевизначимо метод *createHeaderComponent* так, щоби він створював об’єкт класу *AHAminHeaderComponent*. Та додаємо метод *goToMainView,* який буде відповідати за повернення відображення головної компоненти.

```
createHeaderComponent
    ^ AHAdminHeaderComponent from: self

goToMainView
    self answer
```

Рис. 3.138 Методи для створення заголовку та переходу на головну сторінку

Тепер у нас є повністю функціональні кнопки для переміщення між головною сторінкою та сторінкою адміністратора.

3.9.3 Автентифікація

Для автентифікації нам знадобиться окремий компонент *AHAuthenticationModal,* який буде нащадком *AHBasicModal*. Створюємо його та генеруємо відповідні методи доступу до змінних екземпляру.

```
AHBasicModal subclass: #AHAuthenticationModal
    instanceVariableNames: 'password account'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.139 Створення компоненти модального вікна для автентифікації

Додамо метод *renderContentOn,* розділивши вміст модального вікна на різні методи для зручності.

```
renderContentOn: html

    html modal
        id: 'authModalDialog';
        with: [
            html modalDialog with: [
                html modalContent: [
                    self renderHeaderOn: html.
                    self renderBodyOn: html ] ] ]
```

Рис. 3.140 Метод для відображення модального вікна автентифікації

```
renderHeaderOn: html
    html modalHeader: [
        html modalTitle
            level5;
            style: 'color: black';
            with: 'Authentication'.
            html modalCloseButton. ]
```

Рис. 3.141 Метод для відображення заголовку модального вікна

```
renderBodyOn: html

    html modalBody: [
        html form
            multipart;
            with: [
                html formGroup
                    style:
                        'color: black; font-size: medium; display: flex; flex-direction: column';
                    with: [
                        self renderAccountFieldOn: html.
                        self renderPasswordFieldOn: html ].
                html modalFooter: [ self renderButtonsOn: html ] ] ]
```

Рис. 3.142 Метод для відображення вмісту модального вікна

Методи *renderAccountFieldOn* та *renderPasswordFieldOn* містять поля для вводу логіну та паролю.

```
renderAccountFieldOn: html
   html
   formGroup: [
      html label with: 'Account'.
      html textInput
         formControl;
         attributeAt: 'autofocus' put: 'true';
         callback: [ :value | account := value ];
         value: account ]
```

Рис. 3.143 Метод для відображення поля логіну

```
renderPasswordFieldOn: html
   html tbsFormGroup: [
   html label with: 'Password'.
   html passwordInput
      tbsFormControl;
      callback: [ :value | password := value ];
      value: password ]
```

Рис. 3.144 Метод для відображення поля паролю

А метод *renderButtonsOn* відображає кнопки, що будуть розміщені внизу модального вікна, одна з кнопок відповідає за виклик методу validate для валідації введених даних.

```
renderButtonsOn: html
   html tbsButton
      attributeAt: 'type' put: 'button';
      attributeAt: 'data-dismiss' put: 'modal';
      beDefault;
      value: 'Cancel'.
   html tbsSubmitButton
      bePrimary;
      callback: [ self validate ];
      value: 'SignIn'
```

Рис. 3.145 Метод для відображення кнопок модального вікна

Метод validate містить повідомлення до компоненти, *tryConnectionWithLogin.*

```
validate
   ^ component tryConnectionWithLogin: self account andPassword: self password
```

Рис. 3.146 Метод валідації введених даних

Необхідно додати цей метод до компоненти AHScreenComponent, щоб користувач міг залогуватись з усіх сторінок сайту. Поки що правильні логін та пароль у нас будуть збережені в коді.

```
tryConnectionWithLogin: login andPassword: password
   (login = 'admin' and: [ password = 'topsecret' ])
      ifTrue: [ self goToAdministrationView ]
      ifFalse: [ self loginErrorOccurred ]
```

Рис. 3.147 Метод для логування

Залишилось інтегрувати компонент автентифікації в програму, додавши кнопку для логування на заголовок сайту.

```
renderButtonsOn: html
   self renderModalLoginButtonOn: html.
```

Рис. 3.148 Метод відображення кнопок в заголовку

```
renderModalLoginButtonOn: html

   html render: (AHAuthenticationModal from: component).
   html tbsNavbarButton
      tbsPullRight;
      style: self buttonStyle;
      attributeAt: 'data-target' put: '#authModalDialog';
      attributeAt: 'data-toggle' put: 'modal';
      with: [
         html tbsGlyphIcon iconLock.
         html text: 'Залогуватись' ]
```

Рис. 3.149 Метод відображення кнопки логування

Після оновлення сторінки можемо побачити кнопку *“Залогуватись”* та модальне вікно, яке відкривається при натисканні.

3.9.4 Відображення помилок логування

Для того щоби відобразити помилку при логуванні, нам необхідно додати її на сторінку. Для цього додамо нову змінну екземпляру до компоненти *AHScreenComponent*, *showLoginError* та відповідні методи для зміни значення цієї змінної.

```
loginErrorOccurred
    showLoginError := true.

hasLoginError
    ^ showLoginError ifNil: [ false ]
```

Рис. 3.150 Допоміжні методи для відображення помилок логування

Також додаємо метод для отримання тексту помилки.

```
loginErrorMessage
    ^ 'Incorrect login or password'
```

Рис. 3.151 Повідомлення про помилку логування

Додаємо відображення помилки на початку методу *renderMainContentOn* об'єкту класу *AHMainScreenComponent*.

Метод *renderLoginErrorMessageIfAny*, відображає, якщо значення метод *hasLoginError* повертає значення *true*.

```
renderLoginErrorMessageIfAnyOn: html
    self hasLoginError ifFalse: [ ^ self ].
    showLoginError := false.
    html alert
        beDanger;
        with: self loginErrorMessage
```

Рис. 3.152 Відображення повідомлення про помилку логування

```
renderMainContentOn: html

    html column
        style:
            'display: inline-flex; flex-direction: column; vertical-align: top;';
        extraLargeSize: 12;
        with: [
            self renderLoginErrorMessageIfAnyOn: html.
            html row
                style: 'justify-content: center; max-height: 300px';
                with: carousel.
```

Рис. 3.153 Модифікований метод renderMainContentOn головного екрану

3.9.5 Модель користувача

Так як зберігати логін та пароль у коді – це дуже погана практика, нам необхідний механізм безпечного зберігання таких даних. Найпростіший варіант вирішення цього питання – зберегти модель користувача в базі даних.

Створимо класи користувача та адміністратора який унаслідується від нього зі змінними екземпляру, логіном, паролем та інформацією про користувача. В цьому випадку нам не потрібно створювати окрему таблицю під кожен клас, нам достатньо додати метод *isVoyageRoot* який вертатиме значення true лише до *AHUser*, так в таблиці будуть зберігатись як звичайні користувачі, так і адміністратори, а інформація про користувача буде зберігатись напряму в користувачі, без додаткової таблиці. Після цього генеруємо методи доступу до змінних, однак нам потрібно буде змінити метод, що задає значення змінній *password*.

```
Object subclass: #AHUser
   instanceVariableNames: 'login password userInfo'
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'

Object subclass: #AHUserInfo
    instanceVariableNames: 'firstName lastName phoneNumber email'
    classVariableNames: ''
    package: 'AnimalsHouse-Enitites'

AHUser subclass: #AHAdministrator
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'
```

Рис. 3.154 Створення класів користувача, адміністратора та інформації про користувача

Оскільки зберігати такі чутливі дані у відкритому вигляді – небезпечно, тож перед тим, як зберегти значення в базу, нам необхідно зашифрувати пароль наступним чином.

```
password: anObject

    password := SHA256 hashMessage: anObject
```

Рис. 3.155 Метод шифрування паролю

Тепер усі паролі, які будуть зберігатись у базі даних, будуть у безпеці. Також додаємо метод для створення об’єктів цього класу, а також метод для створення інформації про користувача.

```
withLogin: login password: password userInfo: userInfoObject

    | instance |
    instance := self new
                    login: login;
                    password: password;
                    userInfo: userInfoObject.
    instance save.
    ^ instance
```

Рис. 3.156 Метод створення обб'єкту користувача

```
withFirstName: firstNameValue lastName: lastNameValue phoneNumber: phoneNumberValue email: emailValue

    | instance |
    instance := self new.
    instance
        firstName: firstNameValue;
        lastName: lastNameValue;
        phoneNumber: phoneNumberValue;
        email: emailValue.
    ^ instance
```

Рис. 3.157 Метод створення об'єкту інформації про користувача

Також додамо метод для ідентифікації ролі користувача *isAdministrator,* який вертатиме значення *false* для звичайного користувача та *true* для адміністратора.

```
isAdministrator
    ^ false.

isAdministrator
    ^ true.
```

Рис. 3.158 Реалізація методу isAdministrator для класів AHUser та AHAdministrator

Тепер пора інтегрувати наших користувачів. Для початку спробуємо модифікувати метод *tryConnectionWithLogin: andPassword* щоб дати можливість користувачам залогуватись.

Для цього додаємо метод *userWithLogin:* до класу *AHScreenComponent*, що буде знаходити в базі даних користувача з введеним логіном, якщо такого користувача не знайдено – виводимо на екран помилку, якщо ж такий існує перевіряємо чи логін та пароль такі ж як введені у формі. Далі перевіримо чи користувач є адміністратором – якщо так то відкриваємо сторінку адміністратора.

```
userWithLogin: login

    ^ AHUser selectOne: [ :e | e login = login ]
```

Рис. 3.159 Метод отримання користувача за введеним логіном

```
tryConnectionWithLogin: login andPassword: password

    | user |
    user := self userWithLogin: login.
    user ifNil: [ ^ self loginErrorOccurred ].
    (login = user login and: [
        (SHA256 hashMessage: password) = user password asByteArray ])
       ifTrue: [
          user isAdministrator ifTrue: [
             self resetAdminView.
             self goToAdminView ] ]
       ifFalse: [ ^ self loginErrorOccurred ]
```

Рис. 3.160 Модифікований метод логування

3.9.6 Сесія

Тепер у нас є процес логування з реальним логіном і паролем. Відображення помилки про неправильний логін та пароль, однак є один недолік, щоразу як ми хочемо переключитись на сторінку адміністратора, нам необхідно заново вводити логін та пароль. Щоби виправити це, існує механізм сесій.

Створимо клас *AHSession* зі змінною екземпляра *currentUser* та генеруємо методи доступу до змінної.

```
WASession subclass: #AHSession
    instanceVariableNames: 'currentUser'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.161 Створення класу сесії

Після цього створимо метод *isLogged,* що буде вказувати на те, чи в цій сесії ми вже логувались як користувач. Також додаємо метод reset, що буде вилоговувати користувача та повертати до головної сторінки.

```
isLogged

    ^ self currentUser notNil
```

```
reset

    currentUser := nil.
    self requestContext redirectTo: self application url.
    self unregister
```

Рис. 3.162 Додаткові методи класу AHSession

Зареєструємо сесію в кореневому компоненті веб застосунку AHApplicationRootComponent.

```
initialize
    | app |
    app := WAAdmin register: self asApplicationAt: 'AnimalsHouse'.
    app preferenceAt: #sessionClass put: AHSession.
    app addLibrary: JQDeploymentLibrary;
    addLibrary: JQUiDeploymentLibrary;
    addLibrary: TBSDeploymentLibrary;
    addLibrary: SBSDevelopmentLibrary;
    addLibrary: SBSDeploymentLibrary;
    addLibrary: SBSFileLibrary.
```

Рис. 3.163 Модифікований метод initialize кореневого компоненту

Тепер у випадку успішного логування необхідно присвоїти змінній *currentUser* екземпляр класу *AHUser*.

```
tryConnectionWithLogin: login andPassword: password

    | user |
    user := self userWithLogin: login.
    user ifNil: [ ^ self loginErrorOccurred ].
    (login = user login and: [
        (SHA256 hashMessage: password) = user password asByteArray ])
       ifTrue: [
          self session currentUser: user.
          user isAdministrator ifTrue: [
             self resetAdminView.
             self goToAdminView ] ]
       ifFalse: [ ^ self loginErrorOccurred ]
```

Рис. 3.164 Модифікований метод логування

Таким чином ми зберігаємо залогованого користувача у сесії та маємо доступ до нього з будь-якої точки програми.

3.9.7 Спрощена навігація

Наступним пунктом розробки нашого процесу логування є покращення навігації між сторінками. Для цього в методі *renderButtonsOn* компоненти *AHHeaderComponent* змінимо логіку поведінки залежно від того чи користувач залогований.

```
renderButtonsOn: html
    self session isLogged
        ifTrue: [ self renderAdminButtonOn: html ]
        ifFalse: [ self renderModalLoginButtonOn: html. ].
```

Рис. 3.165 Метод для відображення кнопок

Таким чином, якщо адміністратор уже залогований, відображається кнопка для переходу на сторінку адміністратора, якщо ж ні, то буде відображатись логування.

Також, використовуючи метод *reset*, що ми додали в клас сесії, ми можемо модифікувати поведінку кнопки *Disconnect.*

```
renderDisconnectButtonOn: html
    html tbsNavbarButton
        tbsPullRight;
        callback: [ self session reset ];
        with: [
            html text: 'Disconnect '.
            html tbsGlyphIcon iconLogout ]
```

Рис. 3.166 Метод для відображення кнопки від’єднання

Наступним покращенням є додавання кнопки для переходу зі сторінки адміністратора на головну сторінку без потреби розлоговуватись.

3.9.8 Реєстрація користувача

В даний момент у нас немає механізму створення користувача через веб-сторінку, лише наперед створені дані в коді чи базі даних. Тож в цьому розділі додамо компоненту реєстрації.

Створимо клас *AHRegisterModal* з логіном та паролем, а також усією необхідною для нас інформацією про користувача.

```
AHBasicModal subclass: #AHRegisterModal
    instanceVariableNames: 'login password firstName lastName phoneNumber email'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.167 Створення компоненти модального вікна для реєстрації

Реалізуємо метод *renderContentOn* для нього та усі супутні методи.

```
renderContentOn: html

   html modal
      id: 'registerModalDialog';
      with: [
         html modalDialog with: [
               html modalContent: [
                     self renderHeaderOn: html.
                     self renderBodyOn: html ] ] ]
```

```
renderHeaderOn: html

   html modalHeader: [
      html modalTitle
         level5;
         style: 'color: black';
         with: 'Реєстрація'.
      html modalCloseButton ]
```

```
renderBodyOn: html

   html modalBody: [
      html form
         multipart;
         style:
            'color: black; font-size: medium; display: flex; flex-direction: column';
         with: [
            self renderUserInfoOn: html.
            self renderLoginFieldOn: html.
            self renderPasswordFieldOn: html.
            html modalFooter: [ self renderButtonsOn: html ] ] ]
```

Рис. 3.168 Методи відображення для модального вікна реєстрації

```
renderUserInfoOn: html

   html formGroup: [
      html label with: 'Ім`я'.
      html textInput
         formControl;
         required;
         attributeAt: 'autofocus' put: 'true';
         callback: [ :value | self firstName: value ];
         value: firstName.
      html label with: 'Прізвище'.
      html textInput
         formControl;
         required;
         attributeAt: 'autofocus' put: 'true';
         callback: [ :value | self lastName: value ];
         value: lastName.
      html label with: 'Номер телефону'.
      html textInput
         formControl;
         required;
         attributeAt: 'autofocus' put: 'true';
         callback: [ :value | self phoneNumber: value ];
         value: phoneNumber.
      html label with: 'Електронна пошта'.
      html textInput
         formControl;
         required;
         attributeAt: 'autofocus' put: 'true';
         callback: [ :value | self email: value ];
         value: email ]
```

Рис. 3.169 Метод для відображення форми

```
renderLoginFieldOn: html

    html formGroup: [
        html label
            with: 'Логін';
            style: 'color: #000000'.
        html textInput
            formControl;
            required;
            attributeAt: 'autofocus' put: 'true';
            callback: [ :value | self login: value ];
            value: login ]
```

```
renderPasswordFieldOn: html

    html formGroup: [
        html label with: 'Пароль'.
        html passwordInput
            formControl;
            required;
            callback: [ :value | self password: value ];
            value: password ]
```

Рис. 3.170 Методи для відображеня полів логіну та паролю

```
renderButtonsOn: html

    html tbsButton
        attributeAt: 'type' put: 'button';
        attributeAt: 'data-dismiss' put: 'modal';
        beDefault;
        value: 'Скасувати'.
    html tbsSubmitButton
        bePrimary;
        callback: [ self validate ];
        value: 'Зареєструватись'
```

Рис. 3.171 Метод для відображеня кнопок модального вікна

І останнім є метод validate, який власне створить запис користувача у базі даних, та автоматично залогує в систему.

```
validate

    AHUser withLogin: login password: password userInfo: (AHUserInfo
            withFirstName: firstName
            lastName: lastName
            phoneNumber: phoneNumber
            email: email).
    component tryConnectionWithLogin: login andPassword: password
```

Рис. 3.172 Метод валідації введених даних

Наступним потрібно додати кнопку реєстрації на заголовку сайту, тож додамо метод *renderModalRegisterButtonOn:* до *AHHeaderComponent.*

```
renderModalRegisterButtonOn: html

    html render: (AHRegisterModal from: component).
    html tbsNavbarButton
        tbsPullRight;
        style: self buttonStyle;
        attributeAt: 'data-target' put: '#registerModalDialog';
        attributeAt: 'data-toggle' put: 'modal';
        with: [
            html tbsGlyphIcon iconLock.
            html text: 'Реєстрація' ]
```

Рис. 3.173 Метод для відображення кнопки реєстрації

Модифікуємо метод renderButtonsOn: в *AHHeaderComponent* наступним чином, щоб він містив різний набір кнопок, в залежності від того чи залогований користувач.

```
renderButtonsOn: html

    html container
        style: 'justify-content: flex-end; margin-right: 7%';
        with: [
            self session isLogged
                ifTrue: [
                    self session currentUser isAdministrator
                        ifTrue: [ html form: [ self renderAdminButtonOn: html ] ]
                        ifFalse: [ html form: [ self renderDisconnectButtonOn: html ] ] ]
                ifFalse: [
                    html form: [
                        self
                            renderModalLoginButtonOn: html;
                            renderModalRegisterButtonOn: html ] ] ]
```

Рис. 3.174 Модифікований метод відображення кнопок у заголовку

3.10 Створення компонент сторінки адміністратора

Для сторінки адміністратора нам необхідно ще додати компоненти для роботи з заявками, які надсилають користувачі.

Оскільки в нас є два типи заявок: про знаходження тварини та бажання забрати тварину, нам необхідно мати два списки. Давайте використаємо загальний компонент *AHGeneralGrid* та створимо на його основі дві галереї, *AHFoundAnimalGrid* та *AHWantAnimalGrid*

```
AHGeneralGrid subclass: #AHFoundAnimalGrid
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Components'

AHGeneralGrid subclass: #AHWantAnimalGrid
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.175 Створення компонент знайдених тварин та тих які хочуть забрати

Ці два класи не потребують ніяких доповнень, окрім визначення методу getData який є індивідуальним залежно від галереї.

Також нам необхідно створити класи карток для відповідних галерей

3.10.1 AHFoundAnimalCard

```
AHBasicCard subclass: #AHFoundAnimalCard
   instanceVariableNames: 'name age sex species breed weight healthStatus description image'
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.176 Створення картки знайденої тварини

Генеруємо всі необхідні методи доступу до змінних екземпляра та модифікуємо один з них, щоб змінні відповідали тим даним, які необхідні для створення об’єкту тварини:

```
breed: breedName
   breed := (Breed selectAll select: [ :breedObj | breedObj name = breedName ]) at: 1
```

Рис. 3.177 Метод присвоєння значення змінній breed

Наступним кроком додаємо методи *renderContentOn* для відображення картки, та *renderFormBodyOn* для відображення форми в модальному вікні.

```
renderContentOn: html

   html card
      style: 'margin: 15px';
      with: [
         self renderAnimalImageOn: html with: data photo.
         html cardBody: [
               html cardTitle level5 with:
                     data contactName , ',' , data contactPhone.
               html cardSubtitle
                  level6;
                  mutedText;
                  with: data place , ',' , data description.
               html formButton bePrimary
                  dataToggle: 'modal';
                  dataTarget: '#AddAnimalModal' , data id;
                  with: 'Зареєструвати' ] ].
   html modal
      id: 'AddAnimalModal' , data id;
      with: [
         html modalDialog with: [
               html modalContent: [
                     html modalHeader: [
                           html modalTitle
                              level5;
                              with: 'Зареєструвати тварину'.
                           html modalCloseButton ].
                     self renderFormBodyOn: html ] ] ]
```

Рис. 3.178 Метод для відображення картки знайденої тварини

```
renderFormBodyOn: html

   self reset.
   html modalBody: [
      html form
         style:
            'color: black; font-size: medium; display: flex; flex-direction: column';
         multipart;
         with: [
            html formGroup: [
                  html label: 'Фото'.
                  html fileUpload callback: [ :file | self image: file ] ].
            html formGroup: [
                  html label
                     style: 'margin-right: 5px';
                     with: 'Кличка'.
                  html textInput callback: [ :value | name := value ] ].
            html formGroup: [
               html label
                  style: 'margin-right: 5px';
                  with: 'Стать'.
               html select formControl
                  list: Sex list;
                  callback: [ :value | sex := value ] ].
            html formGroup: [
               html label
                  style: 'margin-right: 5px';
                  with: 'Порода'.
               html select
                  list: (Species breedsOf: data animalSpecies);
                  callback: [ :value | self breed: value ] ].
            html formGroup: [
               html label
                  style: 'margin-right: 5px';
                  with: 'Вік'.
               html textInput callback: [ :value | age := value asNumber ] ].
            html formGroup: [
               html label
                  style: 'margin-right: 5px';
                  with: 'Вага'.
               html textInput callback: [ :value | weight := value asNumber ] ].
            html formGroup: [
               html label
                  style: 'margin-right: 5px';
                  with: 'Здоров`я'.
               html textInput callback: [ :value | healthStatus := value ] ].
            html formGroup: [
               html label
                  style: 'margin-right: 5px';
                  with: 'Опис'.
               html textInput callback: [ :value | description := value ] ].
            html modalFooter: [
               html formButton
                  beSecondary;
                  dataDismiss: 'modal';
                  callback: [ self reset ];
                  with: 'Закрити'.
               html formButton
                  bePrimary;
                  callback: [ self save ];
                  with: 'Зареєструвати' ] ]
```

Рис. 3.179 Метод для відображення форми в модальному вікні

Для правильної роботи модального вікна необхідно додати методи reset та save, один для скидання змінних до початкового стану, інший – для створення та зберігання нової тварини в базі даних.

```
reset

    name := nil.
    age := nil.
    sex := nil.
    species := data animalSpecies.
    breed := (Species breedsOf: species) asArray at: 1.
    weight := nil.
    healthStatus := nil.
    description := nil.
    image := nil
```

Рис. 3.180 Метод повернення значень змінних до початкових

```
save

    | class |
    class := species = Species cat
                ifTrue: [ Cat ]
                ifFalse: [ Dog ].
    image ifNil: [ image := data image ].
    class
        withName: name
        age: age
        sex: sex
        breed: breed
        weight: weight
        healthStatus: healthStatus
        description: description
        image: (self saveImage: image).
    data remove
```

Рис. 3.181 Метод збереження тварини у базі даних з введених даних

3.10.2 AHWantAnimalCard

Компонента AHWantAnimalCard має набагато менше змінних, оскільки їй не потрібно створювати нових даних. Усе, що необхідне, це об’єкт класу AHWantAnimal, який містить контактні номер та ім’я людини, що бажає забрати тварину, та власне об’єкт тварини, яку бажають забрати.

```
AHBasicCard subclass: #AHWantAnimalCard
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.182 Створення картки тварини яку хочуть забрати

У цьому класі нам необхідні лише кілька методів. Метод для видалення запиту в разі, якщо потенційний власник не підходить – *delete,* та метод для підтвердження передачі тварини новому власнику, що видалить цю заявку та забере картку тварини із загального списку – *approve*.

```
delete
    request remove.
```

```
approve
    request animal remove.
    request remove.
```

Рис. 3.183 Метод видалення та підтвердження заявки

І обов’язково методи для відображення картки та вмісту модального вікна. Серед них данні про користувача, який хоче забрати тварину.

```
renderContentOn: html

    html card
        style: 'margin: 15px';
        with: [
            self renderAnimalImageOn: html with: data animal image.
            html cardBody: [
                    html cardTitle level5 with:
                            data user userInfo firstName , ','
                            , data user userInfo phoneNumber.
                    html cardSubtitle
                        level6;
                        mutedText;
                        with:
                            data animal name printString , ','
                            , data animal age printString , ','
                            , data animal breed name printString.
                    html formButton bePrimary
                        dataToggle: 'modal';
                        dataTarget: '#GiveAnimalModal' , data id;
                        with: 'Віддати' ] ].
    html modal
        id: 'GiveAnimalModal' , data id;
        with: [
            html modalDialog
                style: 'max-width: 70vh';
                with: [
                    html modalContent: [
                            html modalHeader: [
                                    html modalTitle
                                        level5;
                                        with: 'Віддати тварину'.
                                    html modalCloseButton ].
                            self renderModalBodyOn: html ] ] ]
```

Рис. 3.184 Методи для відображення картки

```
renderModalBodyOn: html

    html modalBody: [
        self renderUserInfoOn: html.
        html form
            style:
                'color: black; font-size: medium; display: flex; flex-direction: column';
            multipart;
            with: [
                html modalFooter: [
                        html label
                            style: 'justify-content: start;';
                            with: 'Віддати тварину новому власнику?'.
                        html formButton
                            beSecondary;
                            callback: [ self delete ];
                            with: 'Ні'.
                        html formButton
                            bePrimary;
                            callback: [ self approve ];
                            with: 'Так' ] ] ]
```

Рис. 3.185 Методи для відображення вмісту модального вікна

```
renderUserInfoOn: html

    html container
        style:
            'margin-left: 5%; width:80%;color: black; font-size: medium; display: flex; flex-direction: column';
        with: [
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Ім`я:'.
                    html paragraph: data user userInfo firstName ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Прізвище:'.
                    html paragraph: data user userInfo lastName ].
            html row
                style: 'justify-content: start;';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Електронна адреса:'.
                    html paragraph: data user userInfo email ].
            html row
                style: 'justify-content: start; ';
                with: [
                    html label
                        style: 'font-size: bold; margin-right: 3%;';
                        with: 'Номер телефону:'.
                    html paragraph: data user userInfo phoneNumber ].
```

Рис. 3.186 Метод для відображення інформації про користувача

Також модифікуємо метод *renderModalOn* класу AHAnimalCard для того, щоб обмежити можливість надіслати заявку користувачам, що не залоговані в систему.

```
renderModalOn: html

   html modal
      id: 'TakeAnimalModal' , data id;
      with: [
         html modalDialog with: [
               html modalContent: [
                     self session isLogged
                        ifTrue: [
                           html modalHeader: [
                                 html modalTitle
                                    level5;
                                    with: 'Забрати тварину'.
                                 html modalCloseButton ].
                           self renderModalBodyOn: html ]
                        ifFalse: [
                           html modalHeader: [
                              html modalTitle
                                 level5;
                                 with: 'Доступно лише зареєстрованим користувачам'.
                              html modalCloseButton ] ] ] ] ]
```

Рис. 3.187 Метод для відображення модального вікна

3.10.3 Відображення вмісту AHAdminComponent

Для того, щоб сторінка адміністратора відобразила два списки заявок, нам необхідно зробити ще кілька кроків.

В першу чергу необхідно додати реалізацію метода *getData* для *AHFoundAnimalGrid* та *AHWantAnimalGrid* відповідно:

```
getData
   ^ AHFoundAnimal selectAll collect: [ :report | AHFoundAnimalCard withReport: report ].

getData
   ^ AHWantAnimal selectAll collect: [ :request | AHWantAnimalCard withRequest: request ].
```

Рис. 3.188 Методи отримання даних для *AHFoundAnimalGrid* та *AHWantAnimalGrid*

Після цього необхідно додати змінні *foundAnimalGrid* та *wantAnimalGrid* на рівні екземпляру класу *AHAdminComponent* та створити їх у ініціалізації екземплярів цього класу.

```
initialize

    super initialize.
    foundAnimalGrid := AHFoundAnimalGrid from: self.
    wantAnimalGrid := AHWantAnimalGrid from: self
```

Рис. 3.189 Метод initialize для сторінки адміністратора

Останнім кроком буде додавання методу *renderMainContentOn* до екземпляру класу *AHAdminComponent*.

```
renderMainContentOn: html

    html column
        style: 'min-height: 800px';
        extraLargeSize: 12;
        with: [
            html row with: [
                    html column
                        extraLargeSize: 1;
                        with: [ html heading: 'Знайдені тварини' ].
                    html column
                        extraLargeSize: 10;
                        with: [
                            html column
                                extraLargeSize: 10;
                                extraLargeOffset: 1;
                                mediumSize: 10;
                                mediumOffset: 1;
                                with: foundAnimalGrid ] ].
            html horizontalRule.
            html row with: [
                html column
                    extraLargeSize: 1;
                    with: [ html heading: 'Заявки на отримання' ].
                html column
                    extraLargeSize: 10;
                    with: [
                        html column
                            extraLargeSize: 10;
                            extraLargeOffset: 1;
                            mediumSize: 10;
                            mediumOffset: 1;
                            with: wantAnimalGrid ] ] ]
```

Рис. 3.190 Модифікований метод для відображення вмісту сторінки адміністратора

Таким чином, ми закінчили створення сторінки адміністратора.

3.11 Подальша доля тварин

Наступним нашим кроком буде створення сторінки, де можна переглянути усіх тварин, що були віддані новим власникам, також необхідно буде реалізувати можливість новому власнику завантажити звіт, про те, що зараз з цією тваринкою. А також відобразимо статистику щодо відсотку тварин, які знайшли нових власників.

3.11.1 Статус тварини

Для початку нам потрібно додати нові характеристики тварини – а саме її статус та власника. Для цього додамо до класу *Animal* змінні екземпляру *status* та *owner,* де *status* буде текстом, який ми будемо проставляти через додатковий клас *AnimalStatus,* який міститиме два методи на рівні класу: *inShelter* та *taken,* за замовчуванням статус тварини буде – «В притулку»

```
Object subclass: #AnimalStatus
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Enums'
```

Рис. 3.191 Створення класу AnimalStatus

```
inShelter
    ^ 'В притулку'

taken
    ^ 'Забрана'
```

Рис. 3.192 Додаткові методи для класу

В свою чергу змінна *owner* буде класу *AHUser*, за замовчуванням це поле буде мати значення *nil.*

Проставимо значення статусу в методі *initialze* в класі *Animal* та додамо відповідні методи для доступу та отримання нових полів.

```
initialize

   super initialize.
   status := AnimalStatus inShelter
```

Рис. 3.193 Метод initialize для класу Animal

Після цього модифікуємо процес передачі тварини користувачу та замість видалення тварини з бази даних змінимо її статус і вкажемо користувача, який надіслав запит як власника тварини.

```
approve

    data animal
        status: AnimalStatus taken;
        owner: data user;
        save.
    data remove
```

Рис. 3.194 Модифікований метод погодження передачі тварини

Також потрібно модифікувати метод *getData* у *AHAnimalsGrid* та додати фільтрацію по статусу, адже в цьому списку нас цікавлять лише тварини, що знаходяться в притулку.

```
getData

    ^ (Animal selectAll select: [ :animal |
          animal status = AnimalStatus inShelter and:
              ((animal sex = sorting selectedSex or:
                    sorting selectedSex = 'Стать') and:
                  ((sorting selectedSpeciesName = animal animalSpecies or:
                        sorting selectedSpeciesName = 'Вид') and:
                      (sorting selectedBreedName = animal breed name or:
                          sorting selectedBreedName = 'Порода'))) ]) collect: [
        :filteredAnimal |
        AHAnimalCard from: self component withData: filteredAnimal ]
```

Рис. 3.195 Модифікований метод отриманння даних у AHAnimalsGrid

3.11.2 AHTakenAnimalsComponent

Для початку давайте створимо компоненту сторінки забраних тварин. Для цього створимо нащадка класу *AHScreenComponent AHTakenAnimalsComponent* з змінною екземпляру *takenAnimalsGrid*, це буде наш список забраних тварин.

```
AHScreenComponent subclass: #AHTakenAnimalsComponent
    instanceVariableNames: 'takenAnimalsGrid'
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.196 Створення компоненти забраних тварин

Реалізуємо метод *renderMainContentOn* для цієї компоненти, в ній ми розмістимо список забраних тварин та повідомлення про помилку логування, якщо така станеться.

```
renderMainContentOn: html

    html column
        style: 'min-height: 86vh';
        extraLargeSize: 10;
        extraLargeOffset: 1;
        mediumSize: 8;
        mediumOffset: 2;
        with: [
            self renderLoginErrorMessageIfAnyOn: html.
            html row
                style: 'min-height: inherit;';
                with: takenAnimalsGrid ]
```

Рис. 3.197 Метод відображення вмісту сторінки забраних тварин

Також необхідно додати кнопку переходу на цю сторінку у заголовку. Тож створимо метод *renderTakenAnimalsButtonOn*: у *AHHeaderComponent,* також надішлемо повідомлення disabled: до об’єкту кнопки, щоб задати умови за якої вона буде заблокована, а саме, тоді коли ця сторінка вже відкрита.

```
renderTakenAnimalsButtonOn: html

    html tbsNavbarButton tbsPullLeft
        style: self buttonStyle;
        disabled: component class = AHTakenAnimalsComponent;
        callback: [ component goToTakenAnimalsView ];
        with: [ html text: 'Забрані тварини' ]
```

Рис. 3.198 Метод відображення кнопки забраних тварин у заголовку

Як бачимо, нам необхідно додати метод *goToTakenAnimalsView.* Для цього додамо цей метод до класу *AHScreenComponent*.

```
goToTakenAnimalsView

    self call: AHTakenAnimalsComponent new
```

Рис. 3.199 Метод переходу на сторінку забраних тварин

Також змінимо трохи логіку розміщення кнопок у заголовку, тепер поділимо заголовок на дві частини. В першій, з лівого краю, будуть розміщені кнопки для переходу на інші сторінки, з правого краю, будуть розміщені кнопки логування, реєстрації та виходу.

```
renderButtonsOn: html

   html container
      style: 'justify-content: flex-start; margin-left: 1%';
      with: [
         html form: [
               self renderTakenAnimalsButtonOn: html. ] ].
   html container
      style: 'justify-content: flex-end; margin-right: 7%';
      with: [
         self session isLogged
            ifTrue: [
               self session currentUser isAdministrator
                  ifTrue: [ html form: [ self renderAdminButtonOn: html ] ]
                  ifFalse: [ html form: [ self renderDisconnectButtonOn: html ] ] ]
            ifFalse: [
               html form: [
                     self
                        renderModalLoginButtonOn: html;
                        renderModalRegisterButtonOn: html ] ] ]
```

Рис. 3.200 Модифікований метод відображення кнопок у заголовку

3.11.3 *AHTakenAnimalsGrid* та *AHTakenAnimalCard*

Тепер потрібно створити список забраних тварин, для початку створимо клас *AHTakenAnimalsGrid* унаслідуємо його від *AHGeneralGrid* та реалізуємо метод *getData*.

```
AHGeneralGrid subclass: #AHTakenAnimalsGrid
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.201 Створення компоненти галереї забраних тварин

```
getData

   ^ (Animal selectAll select: [ :animal |
        animal status = AnimalStatus taken ]) collect: [ :filteredAnimal |
      AHTakenAnimalCard from: self component withData: filteredAnimal ]
```

Рис. 3.202 Метод отримання даних для галереї забраних тварин

І це все, що потрібно реалізувати для цього класу, адже решта функціоналу, уже доступна через *AHGeneralGrid*

Наступним кроком є створення класу *AHTakenAnimalCard*, унаслідуємо його від AHAnimalCard, оскільки деякі методи цього класу нам знадобляться у картці забраних тварин.

```
AHAnimalCard subclass: #AHTakenAnimalCard
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'AnimalsHouse-Components'
```

Рис. 3.203 Створення картки забраних тварин

Реалізуємо методи *renderContentOn, renderModalOn* та *renderModalBodyOn.*

```
renderContentOn: html

   html card
      style: 'margin: 15px; border: 2px solid #355070;';
      with: [
         self renderAnimalImageOn: html with: data image.
         html cardBody: [
               html cardTitle level5 with: data name.
               html cardSubtitle
                  level6;
                  mutedText;
                  with: data sex , ',' , data age printString.
               html formButton bePrimary
                  dataToggle: 'modal';
                  dataTarget: '#TakenAnimalModal' , data id;
                  with: 'Переглянути' ].
         self renderModalOn: html.]
```

Рис. 3.204 Метод відображення картки забраної тварини

```
renderModalOn: html

   html modal
      id: 'TakenAnimalModal' , data id;
      with: [
         html modalDialog with: [
               html modalContent: [
                     html modalHeader: [
                           html modalTitle
                              level5;
                              with: 'Забрана тварина'.
                           html modalCloseButton ].
                     self renderModalBodyOn: html ] ] ]
```

Рис. 3.205 Метод відображення модального вікна про забрану тварину

```
renderModalBodyOn: html

   html modalBody
      style:
         'color: #000000; font-size: 16px; overflow-x: hidden; overflow-y: auto;';
      with: [
         html tbsImage
            style:
               'border: 3px solid black; max-width: 50%; margin-left: 25%; margin-right: 25%; margin-bottom: 3%;';
            url: (self readImageFrom: data image).
         self renderAnimalDataOn: html. ].
   html modalFooter
      style: 'justify-content: space-evenly;';
      with: [
         html form: [
               html formButton
                  beSecondary;
                  dataDismiss: 'modal';
                  with: 'Закрити'. ] ]
```

Рис. 3.206 Метод відображення вмісту модального вікна

Тепер у нас є готова сторінка зі списком де користувачі можуть переглянути усіх тварин, що знайшли нових господарів.

3.11.4 Додавання звітів про тварину

Важливою частиною цієї сторінки є можливість переглянути, що з ними сталось пізніше. Для цього додамо їхнім новим власникам можливість завантажити звіт про цю тварину.

Спочатку додамо нову сутність звіту *AHReport* з змінними екземпляру *animal* – тварина про яку лишають звіт, *user* – користувач який лишає звіт, *image* – зображення завантажене у звіті, *comment* – коментар до звіту, *date* – дата подання звіту. Також додамо метод на рівні класу для створення запису звіту.

```
Object subclass: #AHReport
   instanceVariableNames: 'animal user image comment date'
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'
```

Рис. 3.207 Створення моделі звіту про тварину

```
withAnimal: animalObject user: userObject image: imageObject comment: commentObject date: dateObject

   | instance |
   instance := self new.
   instance
      animal: animalObject;
      user: userObject;
      image: imageObject;
      comment: commentObject;
      date: dateObject.
   ^ instance
```

Рис. 3.208 Метод створення об’єкту класу AHReport

Також необхідно додати змінну *reports* до класу Animal. Модифікуємо метод *initialize* та додамо метод *addReports*: щоб додавати нові звіти.

```
Object subclass: #Animal
   instanceVariableNames: 'name age sex animalSpecies breed weight healthStatus description image status owner reports'
   classVariableNames: ''
   package: 'AnimalsHouse-Enitites'
```

Рис. 3.209 Модифікація класу Animal

```
initialize

   super initialize.
   reports := OrderedCollection new.
   status := AnimalStatus inShelter
```

```
addReport: report

   reports addLast: report
```

Рис. 3.210 Модифікований метод initialize та метод додавання звітів про тварину

Тепер нам необхідно додати можливість залишати звіти. Для цього ми створимо нове модальне вікно, яке буде доступне для відкриття лише, якщо залогований користувач – власник цієї тварини.

Додамо методи відображення модального вікна для завантаження звіту renderReportModalOn, renderReportModalBodyOn.

```
renderReportModalOn: html

   html modal
      id: 'ReportModal' , data id;
      with: [
         html modalDialog with: [
               html modalContent: [
                     html modalHeader: [
                           html modalTitle
                              level5;
                              with: 'Завантажити звіт'.
                           html modalCloseButton ].
                     self renderReportModalBodyOn: html ] ] ]
```

Рис. 3.211 Метод відображення модального вікна для завантаження звітів

```
renderReportModalBodyOn: html

   self reset.
   html modalBody: [
      html form
         multipart;
         with: [
            html formGroup: [
                  html label: 'Фото'.
                  html fileUpload callback: [ :file | image := file ] ].
            html formGroup: [
                  html label: 'Коментар'.
                  html textInput callback: [ :value | comment := value ] ].
            html modalFooter: [
                  html formButton
                     beSecondary;
                     dataDismiss: 'modal';
                     callback: [ self reset ];
                     with: 'Закрити'.
                  html formButton
                     bePrimary;
                     callback: [ self save ];
                     with: 'Зберегти' ] ] ]
```

Рис. 3.212 Метод для відображення вмісту модального вікна

Також додамо методи *reset* та *save*, де ми створюємо звіт з ведених в формі даних і додаємо його до тварини, використовуючи створений заздалегідь метод *addReport*.

```
save

   | report |
   report := AHReport
                withAnimal: data
                user: data owner
                image: (self saveImage: image)
                comment: comment
                date: DateAndTime now asUTC.
   data
      addReport: report;
      save
```

Рис. 3.213 Метод для створеного звіту з введених даних

```
reset

   image := nil.
   comment := nil
```

Рис. 3.214 Метод повернення значень змінних до початкових значень

Після цього лишається лише модифікувати методи *renderContentOn,* щоб додати модальне вікно для подачі звітів та *renderModalBodyOn,* щоб додати кнопку, яка буде відкривати це модальне вікно, яка буде видима лише власнику тварини.

```
self renderModalOn: html.
self renderReportModalOn: html.
```

Рис. 3.215 Модифікований метод renderContentOn

```
html form: [
      html formButton
         beSecondary;
         dataDismiss: 'modal';
         with: 'Закрити'.

      self session currentUser voyageId = data owner voyageId ifTrue: [
         html formButton
            bePrimary;
            dataToggle: 'modal';
            dataTarget: '#ReportModal' , data id;
            dataDismiss: 'modal';
            with: 'Додати звіт' ] ] ]
```

Рис. 3.216 Модифікований метод renderModalBodyOn

Тепер ми маємо можливість додавати звіти, однак досі не можемо їх побачити, давайте це виправимо. Для початку додамо метод *renderReportsDataOn* для відображення звітів

```
renderReportsDataOn: html

   data reports isEmpty ifTrue: [ ^ false ].
   html heading
      style: 'margin: 5%';
      level: 4;
      with: 'Моя подальша доля:'.

   data reports do: [ :report |
      html horizontalRule.
      html row
         style: 'display: inline-flex; justify-content: center';
         with: [
            html tbsImage
               style:
                  'border: 3px solid black; max-width: 50%; margin-left: 25%; margin-right: 25%; margin-bottom: 3%;';
               url: (self readImageFrom: report image).
            html paragraph
               style: 'margin: 3%; word-break: break-word';
               with: report comment ] ]
```

Рис. 3.217 Метод для відображення звітів

Після цього модифікуємо тіло модального вікна в методі *renderModalBodyOn* наступним чином:

```
html modalBody
   style:
      'color: #000000; font-size: 16px; overflow-x: hidden; overflow-y: auto;';
   with: [
      html tbsImage
         style:
            'border: 3px solid black; max-width: 50%; margin-left: 25%; margin-right: 25%; margin-bottom: 3%;';
         url: (self readImageFrom: data image).
      self renderAnimalDataOn: html.
      html div
         style: 'overflow-x: hidden; overflow-y: auto; max-height: 600px';
         with: [ self renderReportsDataOn: html ] ].
```

Рис. 3.218 Модифікований методо renderModalBodyOn

3.11.5 Статистика забраних тварин

Останнім пунктом цього розділу буде додавання статистики, яка б відобразила відсоток тварин, які знайшли нових власників. Для отримання даних щодо відсотку забраних тварин використаємо новий метод *takenPercent* на рівні класу *Animal.*

```
takenPercent

   ^ ((100 / self count) asFloat
      *
         (self selectMany: [ :animal | animal status = AnimalStatus taken ])
            size) rounded
```

Рис. 3.219 Метод для отримання відсотку забраних тварин

Для відображення використаємо доступний в пакеті *Bootstrap* елемент progress, змінимо *renderMainContentOn* в компоненті *AHTakenAnimalsComponent*.

```
renderMainContentOn: html

    html column
        style: 'min-height: 86vh';
        extraLargeSize: 10;
        extraLargeOffset: 1;
        mediumSize: 8;
        mediumOffset: 2;
        with: [
            self renderLoginErrorMessageIfAnyOn: html.
            html row
                style: 'justify-content: center;';
                with: [
                    html label
                        style: 'font-size: 24px';
                        with: 'Тварин прилаштовано'.
                    html progress
                        style: 'height: 40px; width: 100%';
                        with: [
                            | percent |
                            percent := Animal takenPercent.
                            html progressBar
                                infoBackground;
                                valueNow: percent;
                                with: percent asString , '%' ] ].
            html row
                style: 'min-height: inherit;';
                with: takenAnimalsGrid ]
```

Рис. 3.220 Модифікований метод renderMainContentOn

Таким чином ми завершили створення сторінки подальшої долі тварин.

3.12 Відгуки

Останньою частиною нашого сайту є можливість додавати та переглядати відгуки про притулок, та про користувача.

3.12.1 Модель відгуку

Першим ділом додамо сутність відгуку до нашої бази даних. Для цього створимо клас *AHReview* з змінними екземпляру *receiverUser* – користувач, про якого залишили відгук (у випадку якщо відгук про притулок, цей користувач буде адміністратором), *sender* – користувач, який залишив відгук, *text* – текст відгуку, *type* – тип відгуку, позитивний чи негативний, *date* – дата створення відгуку.

```
Object subclass: #AHReview
    instanceVariableNames: 'receiverUser sender text type date'
    classVariableNames: ''
    package: 'AnimalsHouse-Enitites'
```

Рис. 3.221 Створення моделі відгуку

Для зручності створимо допоміжний клас *ReviewType,* який матиме 3 методи на рівні класу, *positive*, *negative* та *list*.

```
Object subclass: #ReviewType
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Enums'
```

Рис. 3.222 Створення класу ReviewType

```
positive
    ^ 'Позитивний'

negative
    ^ 'Негативний'

list
    ^ { self positive.
        self negative }
```

Рис. 3.223 Додаткові методи для класу ReviewType

Також у класі AHReview на рівні класу додамо методи *isVoyageRoot,* щоб вказати необхідність створення окремої таблиці для відгуків, та метод для створення відгуку *withText: receiver: sender: type: .*

```
withText: textObject receiver: receiverObject sender: senderObject type: typeObject

    | instance |
    instance := self new
                    text: textObject;
                    receiverUser: receiverObject;
                    sender: senderObject;
                    type: typeObject;
                    date: DateAndTime now;
                    save.
    ^ instance
```

Рис. 3.224 Метод для створення об’єктів класу AHReview

3.12.2 AHReviewsComponent

Створимо сторінку, де будуть відображатись відгуки на притулок, залишені користувачами. Вона міститиме кнопку для надсилання відгуку та список відгуків.

```
AHScreenComponent subclass: #AHReviewsComponent
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'AnimalsHouse-Components'
```

Рис. 3.225 Створення компонента сторінки відгуків

Реалізуємо метод *renderMainContentOn*, розмістимо на сторінці кнопку, яка буде викликати модальне вікно для заповнення відгуку.

```
renderMainContentOn: html

 html column
   style: 'min-height: 86vh;';
   extraLargeSize: 10;
   extraLargeOffset: 1;
   mediumSize: 8;
   mediumOffset: 2;
   with: [
      self renderLoginErrorMessageIfAnyOn: html.
      html row
         style: 'justify-content: start; height: 6vh; margin: 5px';
         with: [
            html formButton bePrimary
               style: 'height: 4vh; font-size: 16px; border-radius: 10px';
               dataToggle: 'modal';
               dataTarget: '#ReviewModal';
               with: 'Залишити відгук' ] ]
```

Рис. 3.226 Метод для відображення вмісту сторінки відгуків

3.12.3 Форма відгуку на притулок

Для надсилання відгуку нам необхідно створити клас *AHReviewModal* та реалізувати методи *renderContentOn, renderFormBodyOn*.

```
AHBasicModal subclass: #AHReviewModal
  instanceVariableNames: 'type text receiverUser sender'
  classVariableNames: ''
  package: 'AnimalsHouse-Components'
```

Рис. 3.227 Створення компоненти модального вікна для відгуків

```
renderContentOn: html

  html modal
     id: 'ReviewModal';
     with: [
        html modalDialog with: [
             html modalContent: [
                  self session isLogged
                     ifTrue: [
                        html modalHeader: [
                             html modalTitle
                                level5;
                                with: 'Залиште свій відгук'.
                             html modalCloseButton ].
                        self renderFormBodyOn: html ]
                     ifFalse: [
                        html modalHeader: [
                           html modalTitle
                              level5;
                              with: 'Доступно лише зареєстрованим користувачам'.
                           html modalCloseButton ] ] ] ] ]
```

Рис. 3.228 Метод для відображення модального вікна

```
renderFormBodyOn: html

  html modalBody: [
    html form
      style:
        'color: black; font-size: medium; display: flex; flex-direction: column';
      multipart;
      with: [
        html formGroup with: [
            html label: 'Ваше враження:'.
            html select
              list: ReviewType list;
              callback: [ :value | type := value ] ].
        html formGroup with: [
          html label: 'Відгук:'.
          html textInput
            formControl;
            callback: [ :value | text := value ] ].
        html modalFooter: [
          html tbsButton
            beDefault;
            dataDismiss: 'modal';
            callback: [ self reset ];
            with: 'Закрити'.
          html tbsSubmitButton
            bePrimary;
            callback: [ self save ];
            with: 'Надіслати відгук' ] ] ]
```

Рис. 3.229 Метод для відображення вмісту модального вікна

```
save

  AHReview
    withText: text
    receiver: receiverUser
    sender: sender
    type: type
```

```
reset

    type := ReviewType positive.
    text := nil
```

Рис. 3.230 Методи зберігання та очищення форми

Також потрібно додати модальне вікно на сторінку, для цього модифікуємо метод *renderMainContentOn*

```
renderMainContentOn: html

  html column
    style: 'min-height: 86vh;';
    extraLargeSize: 10;
    extraLargeOffset: 1;
    mediumSize: 8;
    mediumOffset: 2;
    with: [
      self renderLoginErrorMessageIfAnyOn: html.
      html row
        style: 'justify-content: start; height: 6vh; margin: 5px';
        with: [
          html formButton bePrimary
            style: 'height: 4vh; font-size: 16px; border-radius: 10px';
            dataToggle: 'modal';
            dataTarget: '#ReviewModal';
            with: 'Залишити відгук' ].
      html row: ((AHReviewModal from: self)
          sender: self session currentUser;
          receiverUser: AHAdministrator getInstance) ]
```

Рис. 3.231 Модифікований метод для відображення сторінки відгуків

Тепер, коли користувач може лишити відгук на притулок, потрібно їх відобразити на сторінці. Для цього додамо методи для відображення відгуків *renderReview: on:* та *renderReviewsOn:*

```
renderReview: review on: html

  html row with: [
    html tbsPanel
      style: 'width: 100%';
      class: 'panel-danger' if: review type = ReviewType negative;
      class: 'panel-success' if: review type = ReviewType positive;
      with: [
        html tbsPanelHeading: [
            html tbsPanelTitle
              level: 3;
              with: review sender userInfo firstName , ' '
                , review sender userInfo lastName , '-'
                , review date asDate greaseString ].
        html horizontalRule.
        html tbsPanelBody: review text ] ]
```

Рис. 3.232 Метод для відображення окремого відгуку

```
renderReviewsOn: html

  ^ ((AHReview selectAll select: [ :review |
        review receiverUser isAdministrator ]) sort: [ :a :b |
      a date > b date ]) collect: [ :filteredReview |
    self renderReview: filteredReview on: html ]
```

Рис. 3.233 Метод для відображення відгуків

Останнім кроком додамо відобразимо відгуки на сторінці.

```
renderMainContentOn: html

  html column
    style: 'min-height: 86vh;';
    extraLargeSize: 10;
    extraLargeOffset: 1;
    mediumSize: 8;
    mediumOffset: 2;
    with: [
      self renderLoginErrorMessageIfAnyOn: html.
      html row
        style: 'justify-content: start; height: 6vh; margin: 5px';
        with: [
          html formButton bePrimary
            style: 'height: 4vh; font-size: 16px; border-radius: 10px';
            dataToggle: 'modal';
            dataTarget: '#ReviewModal';
            with: 'Залишити відгук' ].
      html div
        style:
          'height: 80vh; overflow-x: hidden; overflow-y: auto; border: 3px solid #355070; border-radius: 12px; padding: 5px';
        with: [ self renderReviewsOn: html ].
      html row: ((AHReviewModal from: self)
            sender: self session currentUser;
            receiverUser: AHAdministrator getInstance) ]
```

Рис. 3.234 Метод для відображення сторінки відгуків

Тепер у нас є готова сторінка з відгуками на притулок, які можуть переглядати усі користувачі та залоговані користувачі можуть їх лишати.

3.12.4 Відгук на користувача

Останнім необхідним функціоналом сайту є можливість адміністратору залишати відгук про користувача та переглянути їх під час розгляду заявки на отримання тварини.

Тож для початку додамо можливість залишати коментарі на користувача, для цього додамо кнопку до модального вікна на сторінці забраних тварин, зробити це можна модифікувавши в класі *AHTakenAnimalCard* методи *renderModalBodyOn* та *renderContentOn* – додавши модальне вікно відгуку до картки.

```
html modalFooter
  style: 'justify-content: space-evenly;';
  with: [
    html form: [
        html formButton
          beSecondary;
          dataDismiss: 'modal';
          with: 'Закрити'.

        self session isAdministrator ifTrue: [
          html formButton
            bePrimary;
            dataDismiss: 'modal';
            dataToggle: 'modal';
            dataTarget: '#ReviewModal';
            with: 'Залишити відгук' ].

        self session currentUser voyageId = data owner voyageId ifTrue: [
          html formButton
            bePrimary;
            dataToggle: 'modal';
            dataTarget: '#ReportModal' , data id;
            dataDismiss: 'modal';
            with: 'Додати звіт' ] ] ]
```

Рис. 3.235 Модифікований метод renderModalBodyOn

```
renderContentOn: html

  html card
     style: 'margin: 15px; border: 2px solid #355070;';
     with: [
        self renderAnimalImageOn: html with: data image.
        html cardBody: [
             html cardTitle level5 with: data name.
             html cardSubtitle
               level6;
               mutedText;
               with: data sex , ',' , data age printString.
             html formButton bePrimary
               dataToggle: 'modal';
               dataTarget: '#TakenAnimalModal' , data id;
               with: 'Переглянути' ].
        self renderModalOn: html.
        self renderReportModalOn: html.
        html div with: ((AHReviewModal from: component)
              sender: self session currentUser;
              receiverUser: data owner) ]
```

Рис. 3.236 Модифікований метод renderContentOn картки забраних тварин

Тепер адміністратор може залишити позитивний чи негативний відгук щодо людини яка забрала тварину, але нам потрібно десь відображати цю інформацію.

Для початку, додамо кілька методів до класу *AHUser* а саме: *hasReviews* та *getReviews.* Вони дозволять нам перевірити, чи є в користувача відгуки і отримати їх.

```
hasReviews

  ^ AHReview count: [ :e | e receiverUser login = self login ]
getReviews

  ^ AHReview selectMany: [ :review | review receiverUser = self ]
```

Рис. 3.237 Методи перевірки наявності відгуків та списку відгуків

Наступним кроком є відображення цієї інформації в модальному вікні AHWantAnimalCard, для цього модифікуємо метод *renderUserInfoOn,* додавши вкінці наступні стрічки коду:

```
data user hasReviews ifTrue: [
  html div
    style:
      'justify-content: start; overflow-x: hidden; overflow-y: auto; border: 3px solid #355070; border-radius: 12px; padding: 5px; max-height: 50vh';
    with: [ self renderReviewsOn: html ] ]
```

Рис. 3.238 Модифікований метод renderUserInfoOn

Також додамо ще два методи *renderReviewsOn* та *renderReview: on:*

```
renderReviewsOn: html

  (data user getReviews sort: [ :a :b | a date > b date ]) do: [
    :filteredReview | self renderReview: filteredReview on: html ]
```

Рис. 3.239 Метод renderReviewsOn для AHWantAnimalCard

```
renderReview: review on: html

  html row with: [
    html tbsPanel
      style: 'width: 100%';
      class: 'panel-danger' if: review type = ReviewType negative;
      class: 'panel-success' if: review type = ReviewType positive;
      with: [
        html tbsPanelHeading: [
            html tbsPanelTitle
              level: 3;
              with: review sender userInfo firstName , ' '
                , review sender userInfo lastName , '-'
                , review date asDate asString ].
        html horizontalRule.
        html tbsPanelBody: review text ] ]
```

Рис. 3.240 Метод renderReview: on: для AHWantAnimalCard

І це все, тепер адміністратор може побачити відгуки на користувача якщо вони присутні.

4 ДЕМОНСТРАЦІЯ ПРОГРАМИ

В результаті в нас вийшов сайт із трьома публічними сторінками та сторінкою адміністратора.

На головній сторінці в нас відображені кілька основних блоків:

- Заголовок із назвою сайту, кнопками для переходу на інші сторінки та можливістю залогуватись/зареєструватись.
- Банери для реклами
- Карусель із фото
- Кнопка з можливістю заповнити форму про знайдену тварину
- Список тварин, з можливістю фільтрації, що шукають нового господаря.

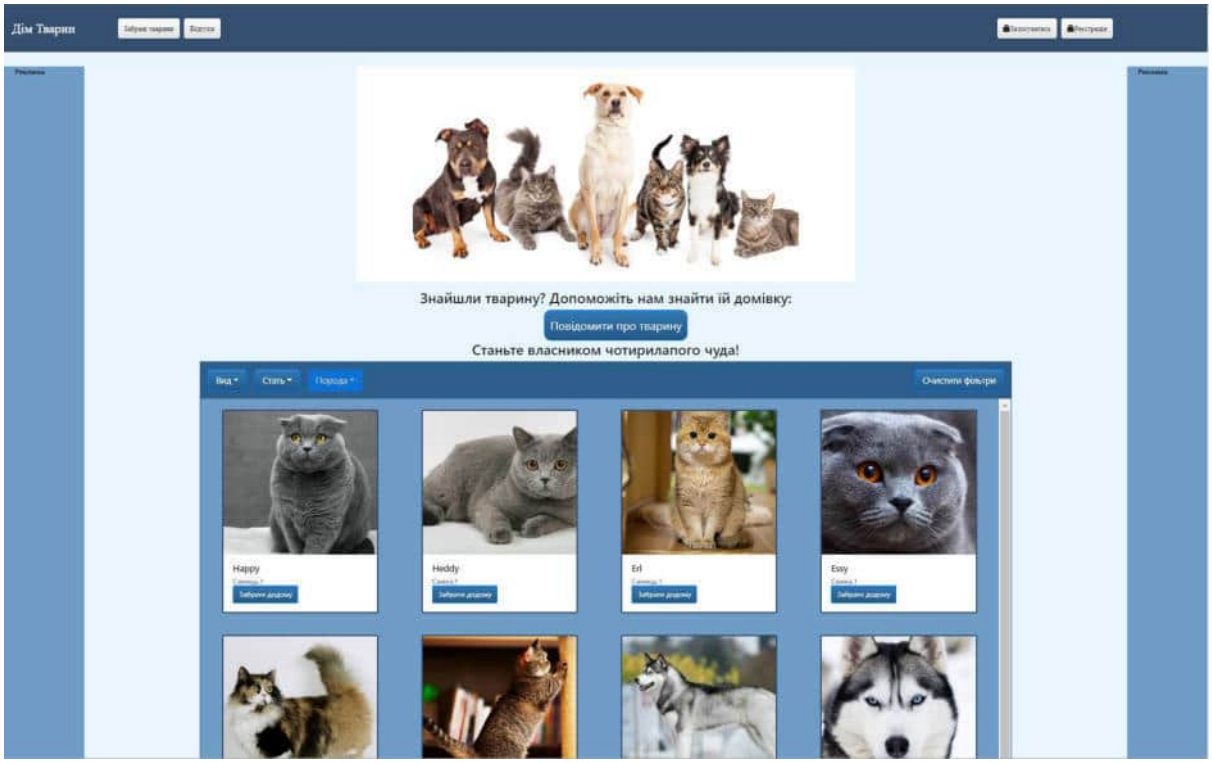

Рис. 4.1 Головна сторінка

В даний момент фільтрація не активна, нам відображають усіх доступних тварин. Якщо ж ми змінимо параметри фільтрації та виберемо вид кота, то ми побачимо наступні результати.

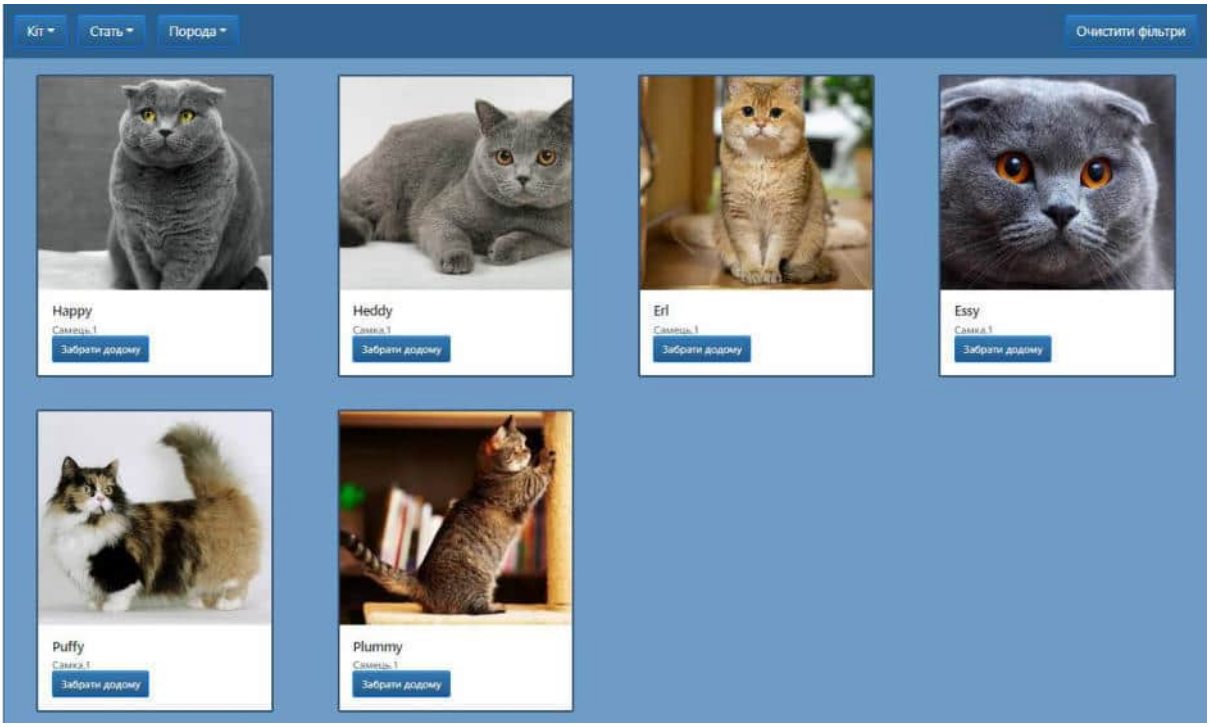

Рис. 4.2 Галерея тварин із фільтрацією

Спробуємо забрати першу кішку додому, натискаємо на кнопку на картці, та бачимо напис:

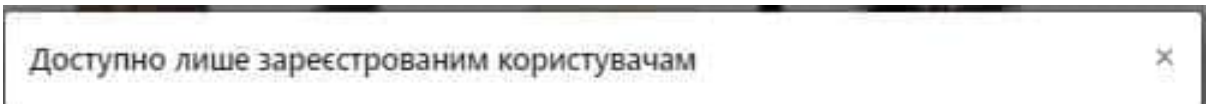

Рис. 4.3 Повідомлення про доступність функціоналу лише зареєстрованим користувачам

Тож зареєструємо нового користувача, для цього в правому верхньому кутку натиснемо на кнопку «Зареєструватись», після чого заповнимо наступну форму:

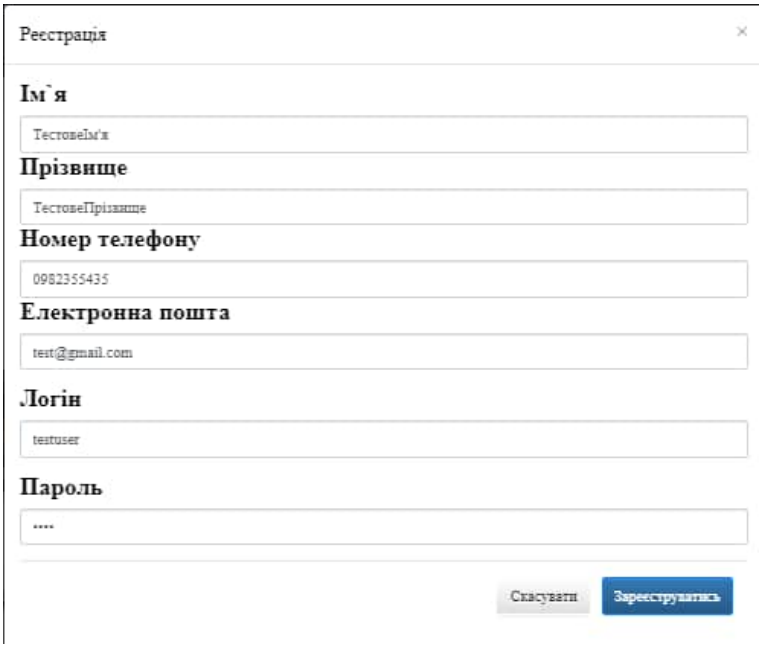

Рис. 4.4 Форма реєстрації

Спробуємо надіслати заявку на отримання тварини ще раз, і побачимо що відкрилось модальне вікно з детальною інформацією про тварину, натиснувши кнопку «Забрати» ми підтверджуємо свою заявку

Також спробуємо повідомити про знайдену тварину, для цього натискаємо на кнопку «Повідомити про тварину» та заповнюємо всі необхідні поля.

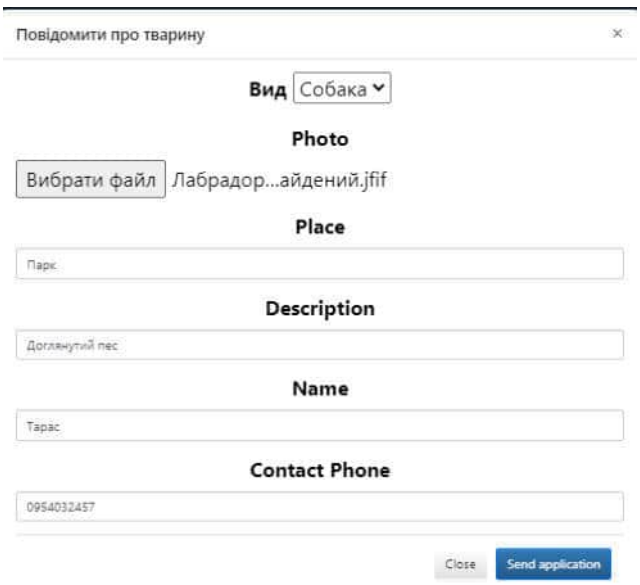

Рис. 4.5 Форма повідомлення про знайдену тварину

Тепер залогуємось як адміністратор, та перевіримо заявки.

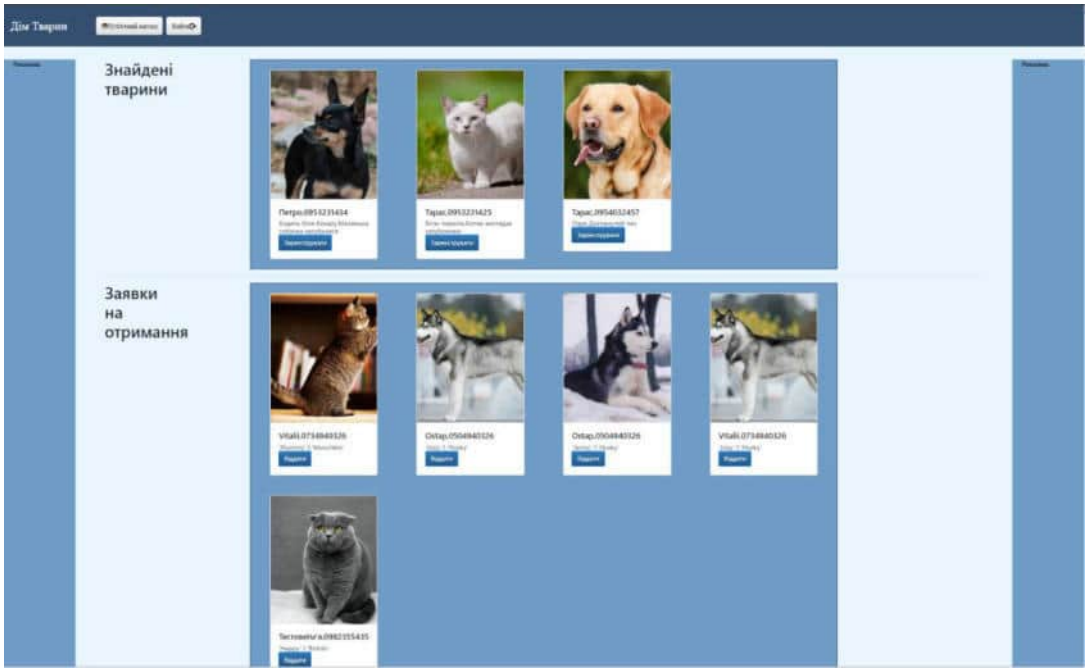

Рис. 4.6 Сторінка адміністратора

Бачимо дві галереї заявок, спробуємо відреагувати на них, спочатку додамо собаку до списку тварин притулку. Натискаємо кнопку зареєструвати, та заповнюємо поля у формі.

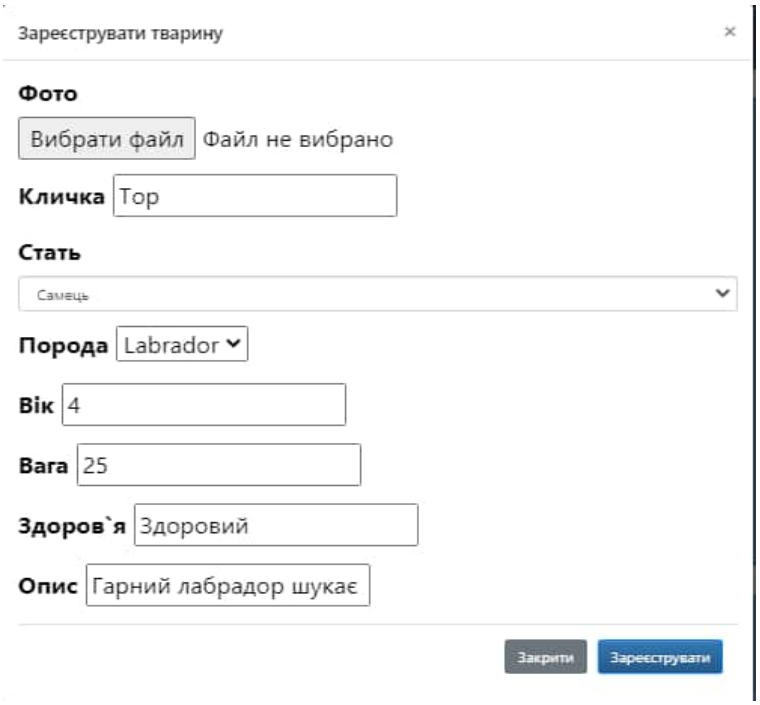

Рис. 4.7 Форма для додавання знайденої тварини до притулку

Після цього натискаємо кнопку “Зареєструвати”, можемо помітити, що заявка зникла зі списку. Натиснемо на кнопку *Public View* та знайдемо цю собаку в списку на головній сторінці.

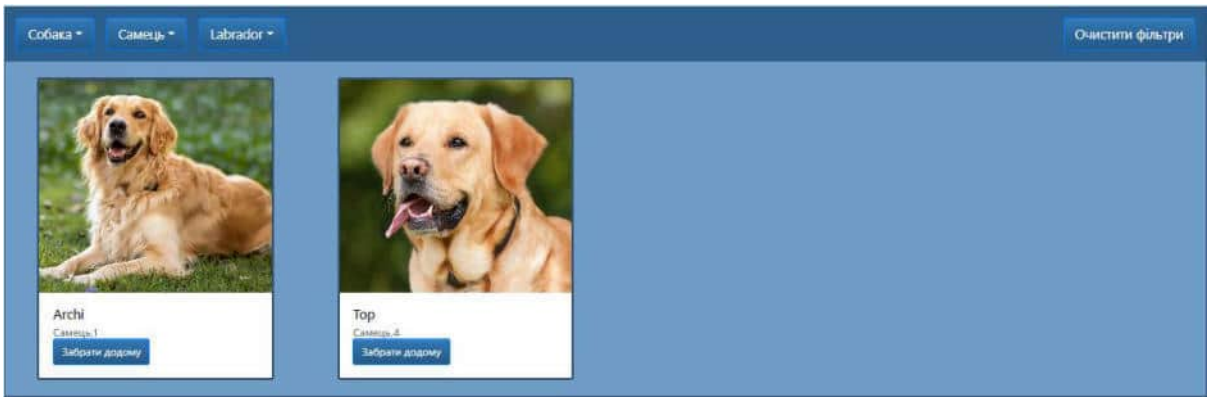

Рис. 4.8 Відфільтрована галерея тварин у притулку

Повернемося на сторінку адміністратора, використовуючи кнопку AdminView, цього разу нам не потрібно логуватись оскільки ми використовуємо сесію для збереження логіну та паролю.

Відреагуємо на Заявку на отримання кота, яку ми надіслали від нашого нового користувача. Для цього натискаємо на кнопку «Віддати» та в модальному вікні можемо побачити детальну інформацію про користувача та натискаємо «Так».

Рис. 4.9 Модальне вікно для передачі тварини новому власнику

Після цього, можна переконатись що картка зникла зі списку заявок, а кіт зник з головної сторінки, та з’явився на сторінці «забраних тварин».

Рис. 4.10 Галерея заявок на отримання тварини

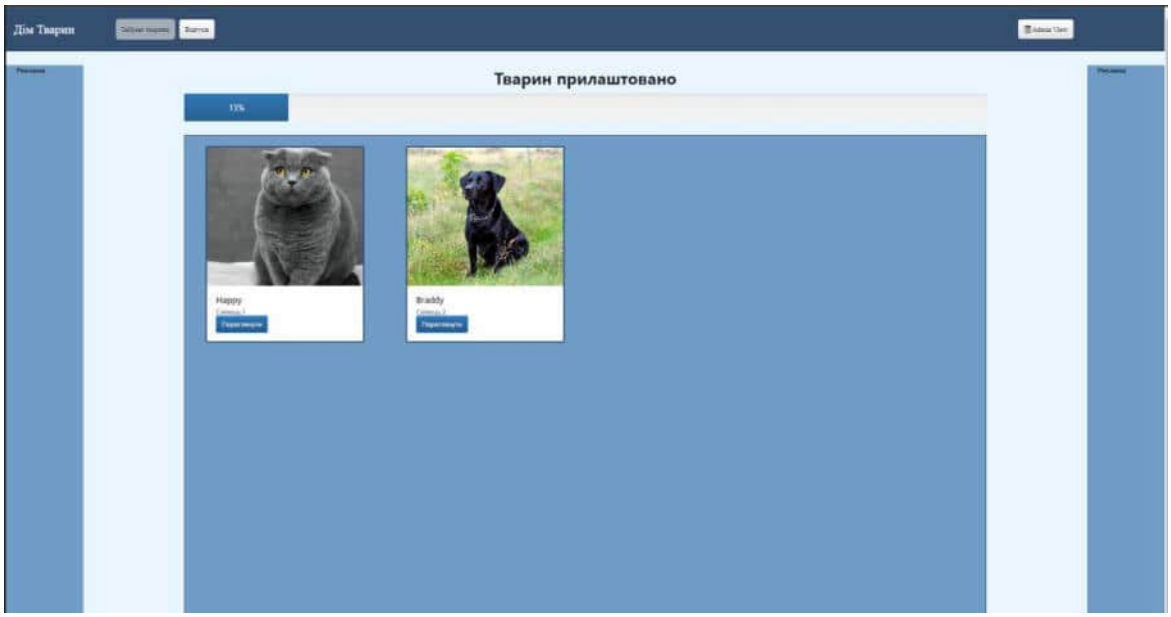

Рис. 4.11 Сторінка забраних тварин

Повернемось на сторінку адміністратора і помітимо що у нас є дві заявки на отримання однієї тварини, в такому випадку адміністратор має вирішити, кому віддати тварину, або ж не віддати нікому. В цьому йому можуть допомогти відгуки про цього користувача.

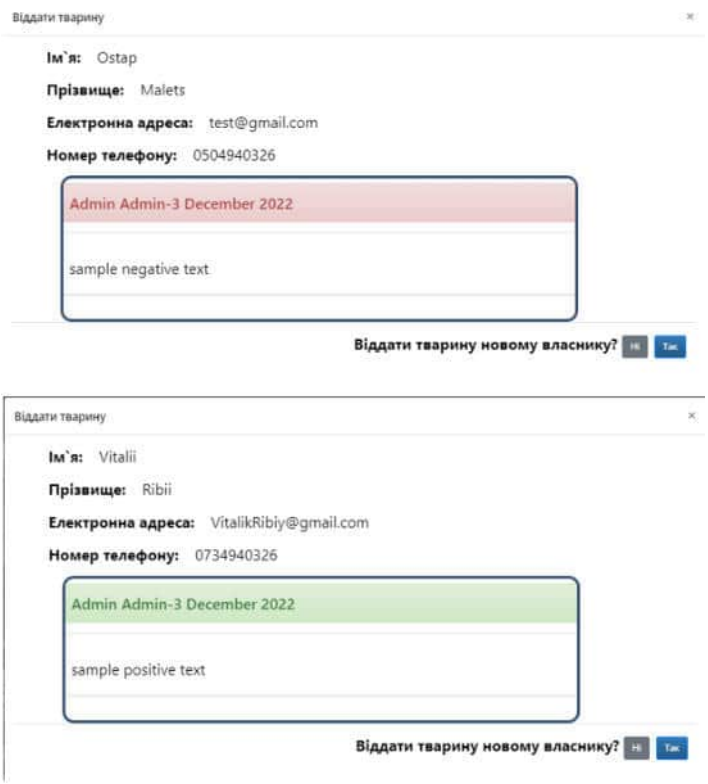

Рис. 4.12 Інформація про користувачів що хочуть забрати тварину

Так ми бачимо, що один користувач має позитивні відгуки, а інший негативні, тож віддамо тварину другому користувачеві. В результаті зі сторінки адміністратора зникнуть обидві заявки, новим власником тварини стане другий користувач, та картка цієї тварини з’явиться на сторінці забраних тварин.

Рис. 4.13 Галерея після передачі тварини обраному користувачеві

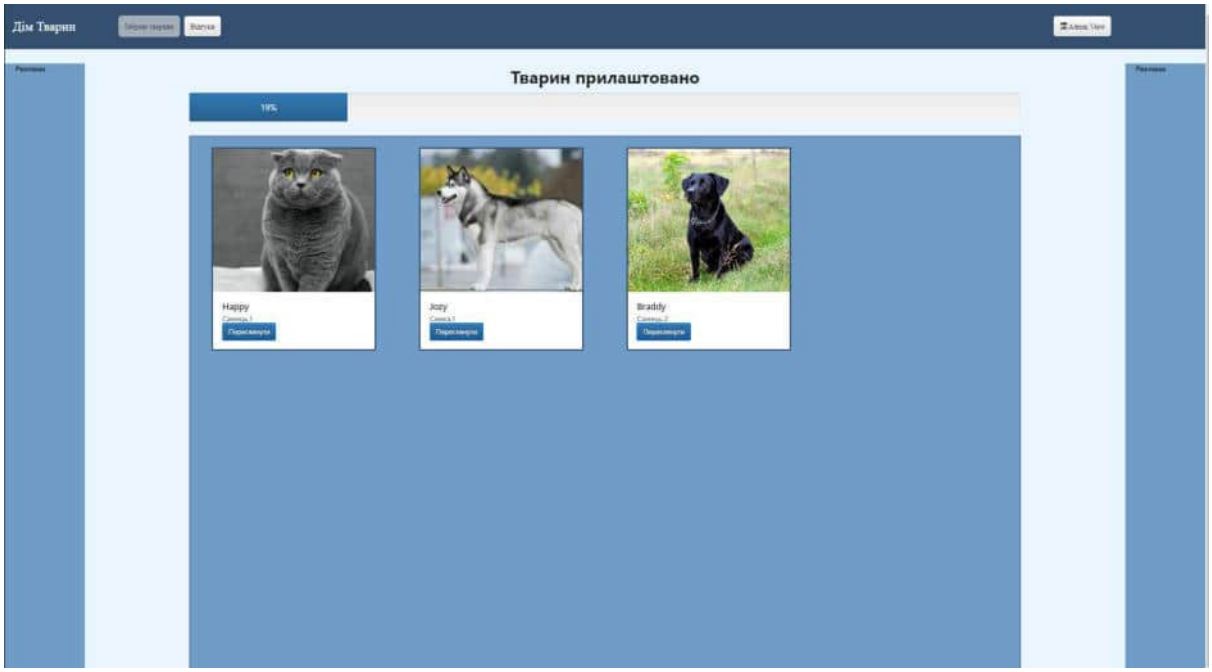

Рис. 4.14 Сторінка забраних тварин після передачі тварини новому користувачу

На сторінці забраних тварин, усі користувачі можуть переглянути подальшу долю тварини, та який відсоток тварин знайшли нових господарів. Так виглядають деталі про тварину, яку забрав наш тестовий користувач. Давайте додамо звіт про цю тварину. Для цього необхідно бути залогованим як користувач, який забрав цю тварину, тоді внизу модального вікна буде розміщена кнопка «Додати звіт». Після натискання на неї, відкриється модальне вікно де ми можемо завантажити фото та залишити свій коментар. Так виглядає інформація про тварину до публікації звіту, та після.

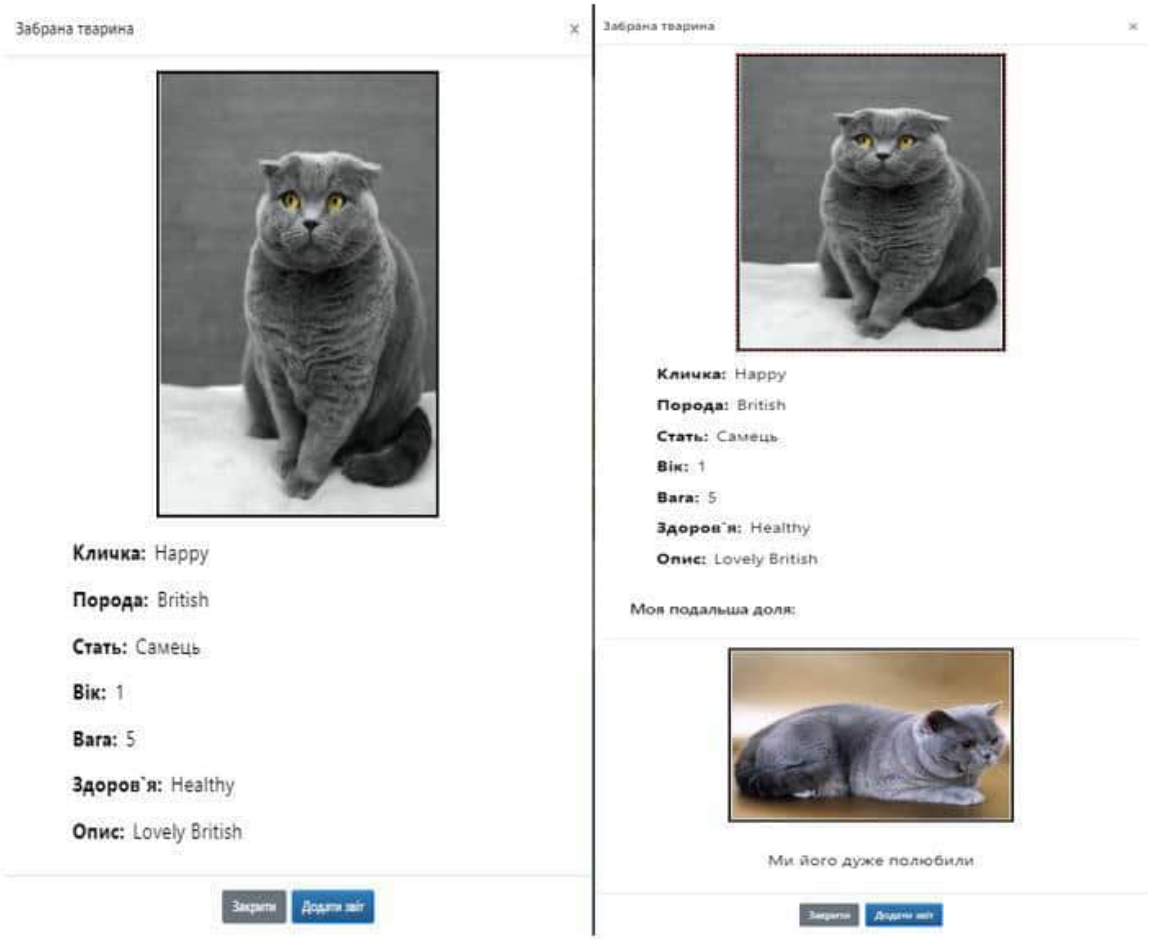

Рис. 4.15 Деталі про забрану тварину без звітів та зі звітами

Якщо адміністратор хоче лишити відгук про користувача який забрав тварину, він може зайти на цю ж картку, та побачити кнопку «Додати відгук». Після чого цей відгук буде відображатись адміністратору, якщо цей користувач знову захоче забрати тварину.

Рис. 4.16 Кнопки в модальному вікні для адміністратора

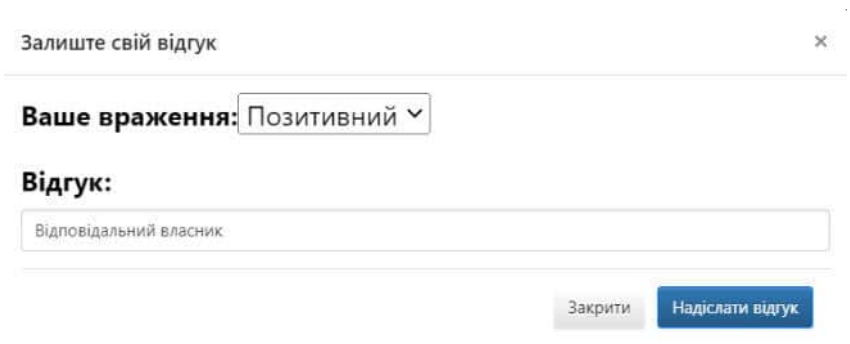

Рис. 4.17 Модальне вікно створення відгуку про користувача

Останньою сторінкою є сторінка відгуків про притулок, де ми можемо побачити список відгуків, від кого вони, коли вони залишені та текст. Цю сторінку переглядати може кожен відвідувач, однак щоб залишити відгук необхідно бути залогованим.

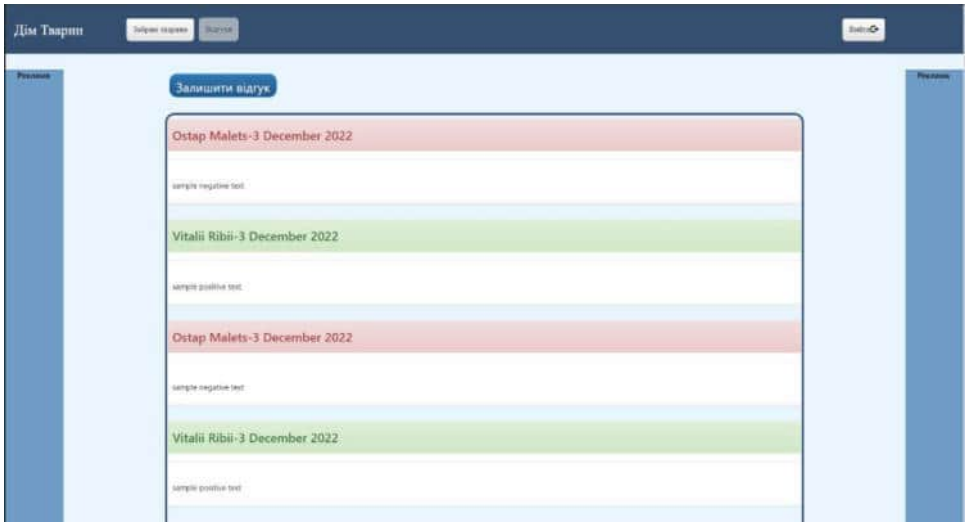

Рис. 4.18 Сторінка відгуків

Для цього необхідно натиснути на кнопку «Залишити відгук» та заповнити відкриту форму.

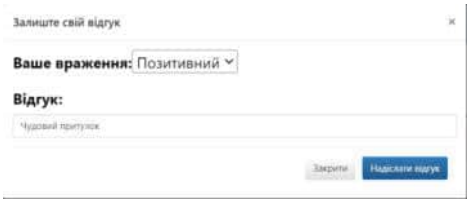

Рис. 4.19 Модальне вікно створення відгуку

Після цього можемо побачити цей відгук на сторінці, зверху оскільки цей відгук є найновішим.

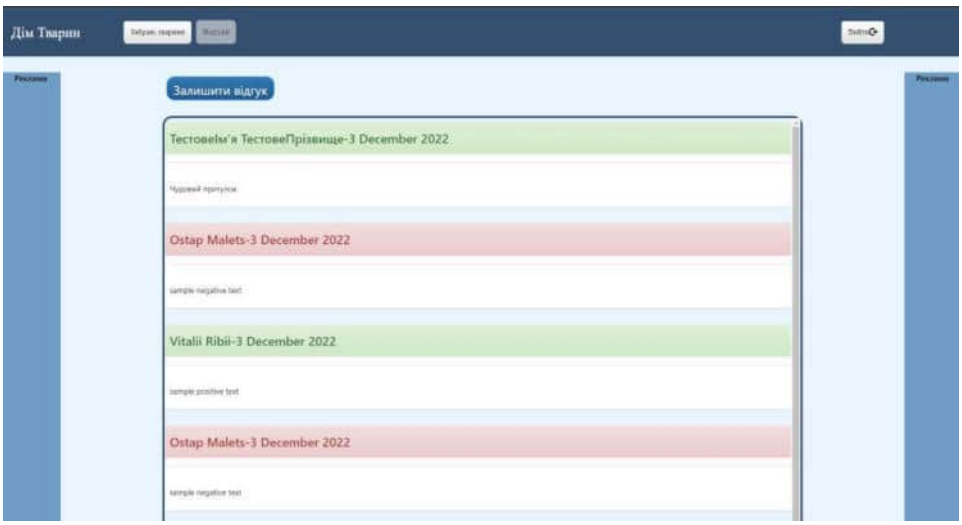

Рис. 4.20 Сторінка відгуків із щойно доданим відгуком

ВИСНОВОК

В результатів виконання магістерської роботи, я освоїв веб розробку в середовищі Pharo. Розглянув різноманітні підходи до написання сайтів, використовуючи цей інструмент. Навчився додавати сторонні пакети, досліджувати функціонал невідомих мені пакетів, використовуючи внутрішні інструменти, та використовувати систему контролю версій у середовищі Pharo.

Розробив сайт притулку для тварин з різноманітними компонентами, що доступні у фреймворку Seaside та Bootstrap. Навчився будувати складну архітектуру веб компонент, через механізми наслідування та динамічної генерації розмітки.

На основі розробленого мною сайту сформував докладну та покрокову методичку для початківців у веб розробці на Pharo, де описав способи вирішення потенційних труднощів, та використання різноманітних інструментів.

Також я прийняв участь у міжнародній науково-практичній конференції "молодіжна наука заради миру та розвитку" де розповів про свої напрацювання та інструмент який досліджую.

СПИСОК ЛІТЕРАТУРИ

1. TinyBlog: Building a web interface with Seaside – http://rmod-pharo-mooc.lille.inria.fr/MOOC/PharoMOOC/TinyBlog/TinyBlog-EN.pdf
2. Dynamic Web Development with Seaside Stéphane Ducasse, Lukas Renggli, C. David Shaffer, Rick Zaccone with Michael Davies – https://book.seaside.st/book
3. http://www.swa.hpi.uni-potsdam.de/seaside/tutorial
4. https://www.pharo.org/documentation