МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

з дисципліни «Об’єктно-орієнтоване програмування (третя мова)»
для здобувачів освітнього ступеня «бакалавр»
зі спеціальності 126 Інформаційні системи та технології
(освітньої програми «Web-технології, Web-дизайн»)
усіх форм навчання

Черкаси
2019

УДК 004.434(076)
Л 12

Затверджено вченою радою ФІТІС,
протокол № 4 від 26.11.2019 р.,
згідно з рішенням кафедри інформаційних технологій проектування,
протокол № 5 від 04.11.2019 р.

Упорядники: Єгорова О.В., к.т.н., доцент,
Лавданський А.О., к.т.н., доцент

Рецензент: Оксамитна Л. П., к.т.н., доцент

Л 12 Лабораторний практикум з дисципліни «Об'єктно-орієнтоване програмування (третя мова)» для здобувачів освітнього ступеня «бакалавр» зі спеціальності 126 Інформаційні системи та технології (освітньої програми «Web-технології, Web-дизайн») усіх форм навчання [Електронний ресурс] / [упоряд. Єгорова О.В., Лавданський А.О.] ; М-во освіти і науки України, Черкас. держ. технол. ун-т. Черкаси: ЧДТУ, 2019. – 38 с. – Назва з титульного екрана.

Методичні рекомендації спрямовані на набуття здобувачами освітнього ступеня «бакалавр» зі спеціальності 126 «Інформаційні системи та технології» практичних навичок програмування мовою Ruby, створення веб-сторінок, валідації форм, оптимізації коду, роботи із базами даних у фреймворку Ruby on Rails.

УДК 004.434(076)

Виробничо-практичне
електронне видання
комбінованого використовування

ЛАБОРАТОРНИЙ ПРАКТИКУМ

з дисципліни «Об'єктно-орієнтоване програмування (третя мова)»
для здобувачів освітнього ступеня «бакалавр»
зі спеціальності 126 Інформаційні системи та технології
(освітньої програми «Web-технології, Web-дизайн»)
усіх форм навчання

Упорядники: **Єгорова** Ольга В'ячеславівна, **Лавданський** Артем Олександрович

В авторській редакції.

ЗМІСТ

ВСТУП

Ruby on Rails – це багаторівневий фреймворк, що призначений для розробки веб-додатків, які використовують бази даних. Фреймворк Ruby on Rails написаний на мові програмування Ruby і базується на архітектурі Model-View-Controller. Мова Ruby – це об'єктно-орієнтована мова програмування в якій всі данні є об'єктами, кожна функція – методом, змінні містять посилання на об'єкти, а надання значення полягає у копіюванні посилання на об'єкт.

Метою викладання навчальної дисципліни «Об'єктно-орієнтоване програмування (третя мова)» є теоретична та практична підготовка здобувачів освітнього ступеня бакалавра у напрямку самостійної розробки web-орієнтованих інформаційних систем мовою Ruby із використанням фреймворку Ruby on Rails.

Основне завдання дисципліни «Об'єктно-орієнтоване програмування (третя мова)» полягає у забезпеченні розуміння і засвоєння здобувачами освітнього ступеня бакалавра синтаксису, основних конструкцій, робочих класів та методів, принципів об'єктно-орієнтованого програмування в Ruby, основних концептів фреймворку Ruby on Rails, а також набуття практичних навичок програмування та використання прикладних і спеціалізованих комп'ютерних систем та середовищ з метою їх запровадження у професійній діяльності.

Метою лабораторних робіт є закріплення здобутих теоретичних знань та набуття здобувачами освітнього ступеня бакалавра практичних навичок програмування мовою Ruby, створення веб-сторінок, валідації форм, оптимізації коду, роботи із базами даних у фреймворку Ruby on Rails.

ЛАБОРАТОРНА РОБОТА № 1

Синтаксис та мовні конструкції мови програмування Ruby

Мета роботи: Закріпити набуті теоретичні знання із основ програмування мовою Ruby та набути практичні навички програмування мовою Ruby.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.
3. Виконати завдання до лабораторної роботи згідно свого варіанту.
4. Скласти та оформити звіт.

Теоретичні відомості

Мова програмування Ruby – це прагматична гнучка мова програмування з інтерпретацією команд, що орієнтована на постійне доопрацювання програмного коду (рефакторинг).

Мова Ruby підтримується всіма операційними системами і дозволяє розробляти сайти та працювати з базами даних.

Алфавіт мови Ruby містить символи, що розташовані на клавіатурі комп'ютера. У мові Ruby використовується кілька десятків *ключових (зарезервованих) слів*:

```
BEGIN END alias and begin
break case class def defined?
do else elsif end ensure
false for if in module
next nil not or redo
rescue retry return self super
then true undef unless until
when while yield
```

Назви змінних та інших *ідентифікаторів* починаються з букви або спеціального модифікатора. Основні правила такі:

- назви локальних змінних (і таких псевдозмінних, як `self` і `nil`) починаються із рядкової букви або знаку підкреслення;
- назви глобальних змінних починаються зі знаку `$`;
- назви змінних екземпляра (які належать об'єкту) починаються зі знака `@`;
- назви змінних класу (які належать класу) передаються двома знаками `@@`;
- назви констант починаються із прописної букви;
- в назвах ідентифікаторів знак підкреслення `_` можна використовувати так само як і рядкові букви.

Наприклад:

- локальні змінні – `dog, _sea, any_app`;
- псевдозмінні – `self, nil, __FILE__`;
- константи – `K6chip, Length, LENGTH`;

- змінні екземпляра – @toolcar, @qwd56ui, @not_var;
- змінні класу – @@rhaifn, @@at_var, @@not_var;
- глобальні змінні – $alhpa, $K87variable, $not_var.

Коментарі в Ruby починаються зі знаку #, який знаходиться поза рядком або символьною константою і продовжується до кінця рядка.

Наприклад,

```
z = k * 10 # Це коментар.
# Це теж коментар.
print "# А це не коментар."
```

Вся вбудована *документація* добувається із програми за допомогою зовнішніх інструментом. З точки зору інтерпретатора – це звичайний коментар, тобто текст, що розміщений між рядками, який починаючи від лексеми

=begin

і закінчуючи лексемою

=end

(включно) ігнорується інтерпретатором (цим лексемам не повинні передувати пробіли).

В мові Ruby змінні не мають типу, проте об’єкти, на які ці змінні посилаються, тип мають. Виділяють прості та складні **типи даних**. До простих відносять *числові*, *логічні* та *символьні* типи даних.

Ієрархія класів Ruby:

- Object:
  - NilClass;
  - TrueClass;
  - FalseClass;
  - Numeric;
    - Integer;
    - Fixnum — цілі числа до 2^{30} за модулем;
    - Bignum — ціле число понад 2^{30} за модулем;
    - Float — дійсне число з плаваючою комою;
    - Complex;
  - Time;
  - Date;
  - String — рядок (символів);
  - Range — діапазон;
  - Struct;
    - Struct::Tms;
  - Array — масив;
  - Hash — асоціативний масив;

Найпростішим видом числа є *ціле число* – це послідовність цифр, яка може починатись знаком плюс чи мінус. Наприклад: 1, 23, чи -10000.

В середині чисел не дозволяється ставити коми, проте дозволяється додавати символи підкреслювання. Для того, щоб зробити число більш читабельним, число можна записати так people = 45_000_000_000.

Рядок (string) – це послідовність символів, що обмежені з обох боків або подвійними лапками, або апострофами. Можна використовувати як подвійні ("), так і одинарні (') лапки, наприклад, "seashi", '2020', чи "The weather is fine!"

Символи – це спрощені рядки, які використовуються в тих випадках, коли їх необхідно вивести не на екран. Вони можуть містити букви, цифри чи символи підкреслювання і починаються з двокрапки, наприклад, :a, :b, or :count_of_time.

Діапазон – це екземпляр класу Range, який є неперервною послідовністю цілих чисел від початкового до кінцевого значення. Діапазон задається парою цілих чисел, що розділені двома або трьома крапками. Приклади діапазонів:

- "a".."z" # всі малі літери латиниці;
- "a"..."z" # те саме, що "a".."y";
- 1..100 # всі натуральні числа від 1 до 100 включно;
- 1...100 # всі натуральні числа від 1 до 99 включно.

Три крапки ... вказують на те, що останній елемент запису до діапазону не належить. Діапазони використовуються в Ruby для вибірки даних і організації циклів.

Масив – це розділений комами список елементів в квадратних дужках, наприклад, [1, 2, 3], ['heat', 'cats', 'football'], [1,"two", [3,4]].

Хеш – це словник у фігурних дужках. Словники співставляють слова з їх означеннями, наприклад, {'c' => 'cclark', 'v' => 'white'}.

Регулярний вираз – це послідовність символів, що оточені слешами, наприклад, /ruby/, /[0-9]+/ чи /^\d{3}-\d{3}-\d{4}/.

Арифметичні оператори призначені для виконання операцій над числами. Окрім звичайних арифметичних операцій додавання (+), віднімання (-), множення (*) та ділення(/), у мові Ruby також використовують оператор ділення по модулю (%), який обчислює залишок від ділення цілих чисел.

Особливість ділення цілих чисел: якщо одне число (але не обидва числа) є від'ємним, Ruby виконує операцію ділення із заокругленням результату з недостачею. Наприклад, при діленні –7 на 3 Ruby повертає –3.

Логічні оператори. Змінні логічного типу bool займають 1 байт пам'яті й можуть набувати лише два значення: 0 – false – хиба; 1 – true – істина.

Це найпростіший тип даних з найменшим набором операцій:

! – заперечення – виконується (значення true) тоді й лише тоді, коли хибний (значення false) операнд (аргумент);

?? – логічне «і» (кон'юнкція) – виконується тоді й лише тоді, коли істинні обидва операнди (аргументи);

|| – логічне «або» (диз'юнкція) – виконується тоді й лише тоді, коли істинний хоча б один операнд (аргумент);

^ – заперечення еквівалентності (виключна диз'юнкція) – виконується тоді й лише тоді, значення операндів (аргументів) відрізняються.

Цих операцій достатньо, щоб записати довільну залежність однієї булевої змінної від довільної скінченої кількості булевих змінних.

Традиційно назви логічних методів закінчують знаком запитання «?». За false може виступати символ порожнечі nil, за true – довільний об'єкт, відмінний від nill.

Оператори порівняння порівнюють значення об'єкта, який розташований ліворуч від оператора, зі значенням об'єкта, який розташовано праворуч від цього оператора. Якщо умова виконується, повертається значення true, інакше – false. Складі позначення операторів порівняння наведено у табл. 1.1

Таблиця 1.1 – Оператори порівняння

Позначення	Назва
==	дорівнює
!=	не дорівнює
<	менше
>	більше
<=	не перевищує
>=	не більше

Оператори розгалуджень. В мові Ruby існують три типи ***операторів розгалуження***: одноальтернативні (неповне розгалуження), двоальтернативне (повне розгалуження) і багатоальтернативне (вибір, варіант).

Двоальтернативне розгалуження реалізується оператором такої структури:

```
if умова [then]
  код…
[elsif умова [then]
  код…]…
[else
  код…]
end
```

Умовою може бути довільний арифметичний чи логічний вираз.

Двоальтерантивний оператор розгалуження, який вказує на те, що код буде виконано за умови, що умова буде хибною (false) має таку структуру:

```
unless умова [then]
  код
[else
  код]
end
```

Розгалуження за багатьма варіантами вибору реалізується оператором такої структури:

```
case вираз0
when вираз1 [, вираз2 …] [then]
  код1 ]
when вираз3 [, вираз4 …] [then]
  код2 ] …
[else
```

```
 код3 ]
end
```

Він позначає порівняння значень виразу, записаного після case, і виразів, записаним після when, з допомогою оператора ===. При збігу хоча б із одним із значень код буде виконано. Якщо такого не відбудеться для жодного запису when, буде виконано код, записаний після else.

Цикли. Оператор циклу *while* виконує інструкції тіла циклу доки виконується певна умова і має таку структуру:

```
while умова [do]
 код (тіло циклу)
end
```

Оператор *until* припиняє виконувати інструкції тіла циклу як тільки умова набуде значення false і має таку структуру

```
until умова [do]
 код (тіло циклу)
end
```

Оператор *for* використовується для виконання тіла циклу для кожного елемента з діапазону значень

```
for змінна in діапазон [do]
 код (тіло циклу)
end
```

Оператори *керування циклом*:

- break – перериває виконання циклу;
- next – перериває виконання поточного кроку (ітерації) циклу з переходом до наступного кроку;
- redo – запускає цикл заново.

Завдання до роботи

1. Розробити комп'ютерну програму для розв'язання задачі. Варіанти завдань наведені у табл. 1.1

Таблиця 1.1 – Початкові дані

Варіант	Задача
1	Дано цілочисловий масив. Виконати циклічний зсув елементів масиву вправо на одну позицію.
2	Дано цілочисловий масив. Перетворити його, вставивши після кожного додатного елемента нульовий елемент.
3	Дано цілочисловий масив. Впорядкувати його за спаданням значенням елементів.
4	Дано цілочисловий масив і число К. Якщо всі елементи масиву менше К, то вивести true; в іншому випадку вивести false.
5	Дано цілочисловий масив. Вивести індекси елементів, які більше свого правого сусіда, і кількість таких чисел.
6	Написати програму для обчислення суми натуральних чисел, введених з клавіатури.

Продовження таблиці 1.1

7	Ввести натуральне число і надрукувати його мінімальну цифру. Наприклад, для 1234076 відповіддю буде 0, а для 77777 – 7.

2. Розробити комп'ютерну програму для розв'язання задачі. Варіанти завдань наведені у табл. 1.2.

Таблиця 1.2 – Початкові дані

Варіант	**Задача**
1	Ввести рядки із файлу, записати в список. Вивести рядки в файл у зворотному порядку.
2	Занести віршований твір одного автора в список та впорядкувати його за зростанням довжин рядків.
3	Дано текст. Необхідно знайти найбільшу кількість цифр, які розміщені поспіль.
4	Дано два текстові рядки. Необхідно визначити, скільки початкових символів першого рядка співпадає із початковими символами другого рядка. Необхідно розглянути два випадки: а) відомо, що рядки різні; б) рядки можуть співпадати.
5	В одному масиві записано кількість м'ячів, які забила футбольна команда в кожній із 20 ігор, в другому – кількість пропущених м'ячів в цій же грі. Для кожної гри визначити словесний результат гри (виграш, програш або нічия).
6	В одному масиві записаний зріст деяких студентів, а в іншому (з тією ж кількістю елементів) – їх прізвища в тому ж порядку, в якому вказаний зріст. Відомо, що всі студенти різного зросту. Вивести прізвище найвищого студента.
7	Ввести число, занести його цифри в стек. Вивести число, у якого цифри розміщені у зворотному порядку.

3. Розробити комп'ютерну програму для розв'язання задачі. Варіанти завдань наведені у табл. 1.3.

Таблиця 1.3 – Початкові дані

Варіант	Задача
1	В кожному рядку знайти та видати заданий підрядок.
2	В кожному рядку віршованого твору видалити всі слова, які містять від трьох до п’яти символів.
3	В кожному рядку віршового твору вибрати і зберегти 3 останні слова.
4	В кожному рядку віршового твору поміняти місця перше і останнє слово.
5	До кожного рядку віршованого твору додати підрядок заданої довжини.
6	В кожному рядку знайти і порахувати кількість слів, які починаються із приголосної букви.
7	Знайти та вивести слова тексту для яких перша буква одного слова співпадає із останньою буквою наступного слова.

Зміст звіту

1. Титульний аркуш.
2. Тема і мета роботи.
3. Короткі теоретичні відомості.
4. Протокол виконання завдання №1.
5. Протокол виконання завдання №2.
6. Протокол виконання завдання №3.
7. Висновки.

Контрольні питання

1. Сформулюйте правило запису ідентифікаторів змінних.
2. На скільки типів даних можуть посилатися змінні у програмі?
3. Поясність сутність арифметичних операторів.
4. Якими операторами мови Ruby реалізуються розгалуження?
5. Наведіть приклад алгоритму, що реалізується за допомогою вкладеного оператора умовного переходу.

ЛАБОРАТОРНА РОБОТА № 2

Об’єктно-орієнтоване програмування в Ruby

Мета роботи: Закріпити набуті теоретичні знання концепту об’єктно-орієнтованого програмування в мові Ruby та набути практичні навички програмування мовою Ruby.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.

3. Виконати завдання до лабораторної роботи згідно свого варіанту.
4. Скласти та оформити звіт.

Теоретичні відомості

Мова Ruby має вбудовані класи, але і користувач може додавати нові класи. Для визначення нового класу застосовується така конструкція:

```
class ClassName
# ...
end
```

Назва класу – це глобальна константа, тому вона повинна починатися із прописної букви. Визначення класу може містити константи, змінні класу, методи класу, змінні екземпляра і методи екземпляра. Дані рівня класу доступні всім об'єктам цього класу, тоді як дані рівня екземпляра доступні тільки одному об'єкту.

В Ruby змінні екземплярів починаються з @, наприклад, @name. Змінні екземпляра не оголошуються. Вони просто «оживають» після першого виклику. Змінні екземплярів доступні для всіх методів екземпляра класу.

Створення об'єкта. Для створення об'єкта існуючого класу використовується метод new. Фактично метод new викликає метод initialize. Метод initialize в Ruby – це спеціальний метод, який діє як конструктор. В середині даного методу повинен бути ініціалізований стан об'єкта, наприклад,

```
class City
  def initialize (name, population) # "Конструктор"
    @name = name
    @population = population
  end
end
```

*Доступ до даних***.** Змінні екземплярів є приватними, вони не доступні поза класом. Доступ до методів є публічним за замовчуванням. Для доступу до змінних екземплярів треба визначити методи «getter» / «setter». Логіка getter/setter проста: одержати існуюче значення і встановити нове значення. Замість фактичного визначення методів getter/setter можна використовувати синтаксис attr_*:

- attr_accessor – getter і setter;
- attr_reader – тільки getter;
- attr_writer – тільки setter.

Разом з тим, в мові Ruby існує self – це посилання на поточний об'єкт. Для звернення до елементів інтанс-об'єкта не обов'язково вказувати self, оскільки він вбудований за замовчуванням, але іноді він потрібен для створення методів класу.

*Наслідування класу***.** Методи і змінні класів викликаються на клас, а не на екземпляр класу. Self поза визначенням методу відноситься до об'єкта Class. Зміні класу починаються з @@. Кожний клас неявно є нащадком Object. Сам Object є нащадком BasicObject. Множинного спадкування не існує, проте використовуються домішки (mixins). Приклад наслідування класів:

```
class Figure # неявно є нащадком Object
 def to_s
   "Геометрична фігура"
 end
 def dimensions
   "площа"
 end
end
class Dot < Figure # означає наслідування, є нащадком класу Figure
 def dimensions # перезаписуємо
   "координата"
 end
end
```

Модулі є контейнерами для класів, методів і констант або інших модулів. Вони схожі на класи, але не можуть бути інстанційовані. Class спадкує Module і додає new. Модулі використовуються для множини назв і домішок. Домішки дозволяють розділити готовий код між декількома класами.

Завдання до роботи

1. Розробити комп'ютерну програму для розв'язання задачі. Варіанти завдань наведені у табл. 2.1.

Таблиця 2.1 – Початкові дані

Варіант	Задача
1	Створити наступну ієрархію класів: транспорт, автомобіль, автобус, трамвай, тролейбус.
2	Створити наступну ієрархію класів: птах, курка, лебідь, пінгвін, страус.
3	Створити наступну ієрархію класів: тварина, корова, кит, дельфін, собака.
4	Створити наступну ієрархію класів: сонячна система, космічне тіло, планета, зірка, супутник.
5	Створити наступну ієрархію класів: рослина, дерево, квітка, мох, кущ.
6	Створити наступну ієрархію класів: океан, протока, море, річка, озеро.
7	Створити наступну ієрархію класів: країна, область, місто, селище, село.

2. Розробити комп'ютерну програму для розв'язання задачі. Варіанти завдань наведені у табл. 2.2.

Таблиця 2.2 – Початкові дані

Варіант	Задача
1	Визначити ієрархію квіті. Створити декілька об’єктів-квітів. Зібрати букет (використовуючи аксесуари) з визначенням його вартості. Виконати сортування квітів у букеті на основі міри свіжості. Знайти квітку у букеті, яка відповідає заданому діапазону довжини стебла.
2	Визначити ієрархію цукерок і інших солодощів. Створити декілька об’єктів-цукерок. Зібрати дитячий подарунок із визначенням його ваги. Виконати сортування цукерок у подарунку на онові одного з його параметрів. Знайти цукерку у подарунці, яка відповідає заданому діапазону вмісту цукру.
3	Визначити ієрархію електроприладів. Ввімкнути деякі в розетку. Підрахувати кількість спожитої електричної енергії. Виконати сортування приладів у квартирі за потужністю. Знайти прилад у квартирі, який відповідає заданому діапазону параметрів.
4	Визначити ієрархію овочів. Зробити салат. Підрахувати калорійність. Виконати сортування овочів для салату на основі одного із параметрів. Знайти овочі в салаті, які відповідають заданому діапазону калорійності.
5	Визначити ієрархію музичних композицій. Записати на диск зібрання музичних творів. Підрахувати тривалість звучання записаної збірки. Виконати перестановку композицій диску за стилістичною приналежністю. Знайти композицію, яка відповідає заданому діапазону довжини треків.
6	Визначити ієрархію рухомого складу залізничного транспорту. Створити пасажирський потяг. Підрахувати загальну чисельність пасажирів та багажу. Знайти в потязі вагони, які відповідають заданому діапазону параметрів кількості пасажирів.
7	Визначити ієрархію тарифів мобільного оператора. Створити список тарифів компанії. Підрахувати загальну чисельність клієнтів. Виконати сортування тарифів на основі розміру абонентської плати. Знайти тариф в компанії, який відповідає заданому діапазону параметрів.

Зміст звіту

1. Титульний аркуш.
2. Тема і мета роботи.
3. Короткі теоретичні відомості.
4. Протокол виконання завдання №1.
5. Протокол виконання завдання №2.
6. Висновки.

Контрольні питання

1. Опишіть конструкцію, яка використовується для створення нового класу в мові Ruby.
2. Опишіть процес створення нового об’єкта в мові Ruby.
3. Які методи забезпечують доступ до даних в мові Ruby?
4. Поясність сутність методу self.
5. Поясність призначення модулів в мові Ruby.

ЛАБОРАТОРНА РОБОТА № 3

Створення Router та ActiveController у фреймворку Ruby on Rails

Мета роботи: Закріпити набуті теоретичні знання та набути практичні навички створення сторінок за допомогою фреймворку Ruby on Rails.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.
3. Виконати завдання до лабораторної роботи згідно свого варіанту.
4. Скласти та оформити звіт.

Теоретичні відомості

Розробка на Ruby (та Ruby on Rails зокрема) з-під Windows завжди була проблематичною. Сприяють цьому декілька особливостей:

– Ruby та Ruby on Rails набагато повільніше працюють на Windows, ніж на Unix-подібних системах;

– багато гемів на Windows не працюють взагалі;

– стиль програмування на Ruby загалом являє собою «юніксовий»;

– Ruby спільнота здебільшого працює на Linux та Mac системах.

Для вирішення цієї проблеми існують такі рішення:

- RailsInstaller;
- Ubuntu як підсистема Windows;
- Vagrant.

RailsInstaller. Популярне рішення серед новачків, яке дозволяє встановити Ruby та Rails, bundler, GIT та sqlite3. Все це успішно запуститься командою rails server. Проте нативно інстальований Ruby погано проявляє себе у Windows.

Ubuntu як підсистема Windows. З останніми оновленнями Windows 10 дозволяє інсталювати всередині себе підсистему Ubuntu. Таким чином, термінал Ubuntu і сама система стають доступними для використання. Проте редагувати файли цієї Ubuntu, використовуючи редактори встановлені у Windows, ззовні не можливо.

Vagrant. Рішення, яке дозволяє розвернути всередині Windows оточення для розробки, використовуючи віртуальну машину, зокрема VirtualBox. Сам по

собі VirtualBox дозволяє не тільки встановити майже будь-що всередині себе, а і одержати доступ до графічного інтерфейсу. За допомогою обгортки Vagrant можна легко запустити віртуальні машини з уже готовою начинкою всередині, використовуючи написані Vagrant бокси.

1. Встановлення Vagrant і VirtualBox.

1.1 Завантажуємо Vagrant і віртуальну машину VirtualBox, яку буде використовувати Vagrant. Завантажити Vagrant. Завантажити VirtualBox. Запускаємо інсталятор і послідовно виконуємо кроки.

1.2 Відкриваємо консоль PowerShell (Shift + right mouse click). Перевіряємо чи встановлений Vagrant командою

```
vagrant –version
```

1.3 Встановлюємо необхідний для синхронізації тек плагін командою

```
vagrant plugin install vagrant-vbguest
```

1.4 Згортаємо консоль.

2. Налаштування конфігурації Vagrant і віртуальної Ubuntu.

2.1 В ОС Windows створюємо директорію в якій будуть зберігатися файли, пов'язані із Ruby on Rails на Windows. Дана директорія буде тотожна за своїм вмістом у ОС Windows і в ОС Linux.

2.2 В дану директорію додаємо файл *Vagrantfile* та скрипт з налаштування оточення *bootstrap.sh*. *Vagrantfile* – файл конфігурації. За допомогою shell-скрипта відбувається налаштування Ruby-оточення.

3. Запуск віртуальної машини.

3.1 Повертаємося в консоль PowerShell і вводимо команду `vagrant up`. Vagrant використає *Vagrantfile* і почне завантажувати Vagrant бокс з дистрибутивом Ubuntu. Після встановлення Ubuntu Vagrant під'єднається до неї, перейде до термінала і виконає скрипт (список команд) з *bootstrap.sh.*

3.2 У цей час у командному рядку буде багато чого відбуватися, але вам нічого не потрібно робити, лише ввести свій логін і пароль для користувача Windows (потрібні для *smb* синхронізації).

4. Встановлення ssh зв'язку із віртуальною машиною.

4.1 В консолі PowerShell після завершення інсталяції вводимо команду `vagrant ssh` для приєднання до віртуальної машини.

4.2 Для перегляжу доступних у Linux директорій вводимо команду `ls`.

4.3 Командою `cd` переходимо у директорію в якій потім будемо зберігати директорію проекта на Ruby on Rails, наприклад, `$ cd workspace`, де `workspace` – це назва папки.

Розглянемо створення сайту за допомогою фреймворку Ruby on Rails на прикладі веб-блогу.

5. Створення нового Ruby on Rails проекту.

5.1 У відкритій раніше директорії (наприклад, `workspace`) створюємо новий rails проект командою

`$ rails new` ***назва проекта*** `--webpack`

Наприклад,

```
$ rails new simplebl --webpack
```

5.2 Запускаємо локальний сервер командою

```
$ rails server -b 0.0.0.0
```

5.3 В браузері переходимо на сторінку localhost:3000 і перевіряємо, чи перекинув Vagrant порт і чи проект дійсно працює (рис. 3.1).

Рисунок 3.1 – Тестова сторінка Ruby on Rails проекту

6. Створення контролера.

6.1 Відкриваємо ще одну консоль PowerShell (друга консоль).

6.2 Для встановлення ssh зв'язку із віртуальною машиною вводимо команду vagrant ssh.

6.3 Переходимо у директорію, в якій зберігається проект на Ruby on Rails, наприклад, $ cd workspace/simplebl.

6.4 Створюємо контролер, який називається greetings, командою

```
$ rails generate controller greetings
```

6.5 У редакторі коду Sublime Text або Atom відкриваємо файл app/controllers/greetings_controller.rb і додаємо код

```
class GreetingsController < ApplicationController
  def hello
  end
end
```

6.6 У директорії app/views/greetings створюємо файл hello.html.erb і додаємо код

```
<h1>Головна сторінка</h1>
```

6.7 Переходимо у файл config/webpack/routes.rb і вказуємо URL адресу, яку будемо відслідковувати

```
Rails.application.routes.draw do
root 'greetings#hello'
end
```

Таким чином, буде призначений контролер, який відповідає за відображення головної сторінки, і метод в контролері, який відповідає за html-шаблон.

6.8 Переходимо у файл environments/development.rb і програмний код

```
config.file_watcher = ActiveSupport::EventedFileUpdateChecker
```

замінюємо на

```
config.file_watcher = ActiveSupport::FileUpdateChecker
```

Після внесення правки для перегляду змін на сайті досить буде лише оновити браузер, а не перевантажувати локальний сервер.

6.9 В першій консолі зупиняємо роботу сервера командою Ctrl+C. Запускаємо сервер командою `$ rails server -b 0.0.0.0`. Запускаєм браузер, переходимо за адресою http://localhost:3000/greetings/hello.

7. Створення статичних сторінок.

7.1 Відкриваємо ще одну консоль PowerShell (друга консоль).

7.2 Створюємо контролер, який називається posts, командою

```
$ rails generate controller posts
```

7.3 Переходимо у файл app/controllers/posts_controller.rb і додаємо код

```
class PostsController < ApplicationController
def index
end
```

7.4 Переходимо у директорію app/views/posts, створюємо файл index.html.erb і додаємо код

```
<h4>Сторінка про нас</h4>
```

7.5 Переходимо у файл config/webpack/routes.rb і вказуємо URL адресу, яку будемо відслідковувати

```
get 'index'=>'posts#index'
```

7.6 Передаємо дані у шаблон. Переходимо у директорію app/controllers/posts_controller.rb і додаємо код

```
class PostsController < ApplicationController
def index
@heading='Сторінка про нас!'
@text='Трохи тексту'
end
end
```

7.7 Переходимо у файл app/views/posts/index.html.erb і додаємо код

```
<h4><%= @heading %></h4>
<p4><%= @text %></p4>
```

7.8 Переходимо в браузер, оновлюємо сторінки http://localhost:3000/ і http://localhost:3000/index.

7.9 Повертаємося до першої консолі PowerShell. Для зупинки сервера натискаємо Ctrl+C.

7.10 В консолі для виходу із робочої папки вводимо команду exit.

7.11 В консолі для зупинки віртуальної машини вводимо команду vagrant halt

7.12 Переходимо до другої консолі, виходимо із робочої папки і зупиняємо віртуальну машину.

Завдання до роботи

1. Встановити Vagrant і VirtualBox.
2. Налаштувати конфігурацію Vagrant і віртуальної Ubuntu.
3. Запустити віртуальну машину.
4. Встановити ssh зв’язок із віртуальною машиною.

5. Створити новий Ruby on Rails проект для створення сайту згідно свого варіанту.

6. В Ruby on Rails проекті створити статичні сторінки сайту згідно свого варіанту.

Варіанти завдань

1. Web-орієнтована інформаційна система садового центру.
2. Web-орієнтована інформаційна система туристичної агенції.
3. Web-орієнтована інформаційна система поліграфічного підприємства.
4. Web-орієнтована інформаційна система рекламного агентства.
5. Web-орієнтована інформаційна система салону краси.
6. Web-орієнтована інформаційна система меблевого салону.
7. Web-орієнтована інформаційна система кадрового агентства.

Зміст звіту

1. Титульний аркуш.
2. Тема і мета роботи.
3. Короткі теоретичні відомості.
4. Протокол виконання завдання №1.
5. Протокол виконання завдання №2.
6. Протокол виконання завдання №3.
7. Протокол виконання завдання №4.
8. Протокол виконання завдання №5.
9. Протокол виконання завдання №6.
10. Висновки.

Контрольні питання

1. Опишіть призначення MVC моделі у фреймворку Ruby on Rails.
2. Поясніть призначення контролера у фреймворку Ruby on Rails.
3. Опишіть процес створення нового контролера у фреймворку Ruby on Rails.
4. Поясність призначення роутера у фреймворку Ruby on Rails.
5. Опишіть процес налаштування ресурсних маршрутів.

ЛАБОРАТОРНА РОБОТА № 4

Робота з базами даних з використанням фреймворку Ruby on Rails

Мета роботи: Закріпити набуті теоретичні знання та набути практичні навички роботи із базою даних SQLite у фреймворку Ruby on Rails.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.
3. Виконати завдання до лабораторної роботи згідно свого варіанту.

4. Скласти та оформити звіт.

Теоретичні відомості

1. Запуск віртуальної машини.

1.1 Відкриваємо консоль PowerShell (перша консоль).

1.2 В консолі вводимо команду `vagrant up` для запуску віртуальної машини.

1.3 Для встановлення ssh зв'язку із віртуальною машиною вводимо команду `vagrant ssh`.

1.4 Переходимо у папку, в якій знаходиться проект на Ruby on Rails, наприклад, `$ cd workspace/simplebl`.

1.5 Запускаємо сервер командою `$ rails server -b 0.0.0.0` і згортаємо консоль.

1.6 Відкриваємо ще одну консоль PowerShell (друга консоль).

1.7 Встановлюємо ssh зв'язок із віртуальною машиною командою `vagrant ssh`.

1.8 Переходимо у папку, в якій знаходиться проект на Ruby on Rails, наприклад, `$ cd workspace/simplebl`, і згортаємо консоль.

Подовжуємо розглядати приклад створення веб-блогу за допомогою фреймворку Ruby on Rails.

2. Створення сторінки на сайті для додавання нових статей.

2.1 У редакторі коду Sublime Text або Atom відкриваємо папку проекту.

2.2 Переходимо у директорію `config/webpack/routes.rb` і додаємо код `resources :posts` для відслідковування всіх необхідних URL адрес, тобто вкажемо контролери з якими працюємо.

2.3 Переходимо у консоль PowerShell (друга консоль), вводимо команду `$rake routes` і перевіряємо URL адреси, які відслідковуються (рис. 4.1).

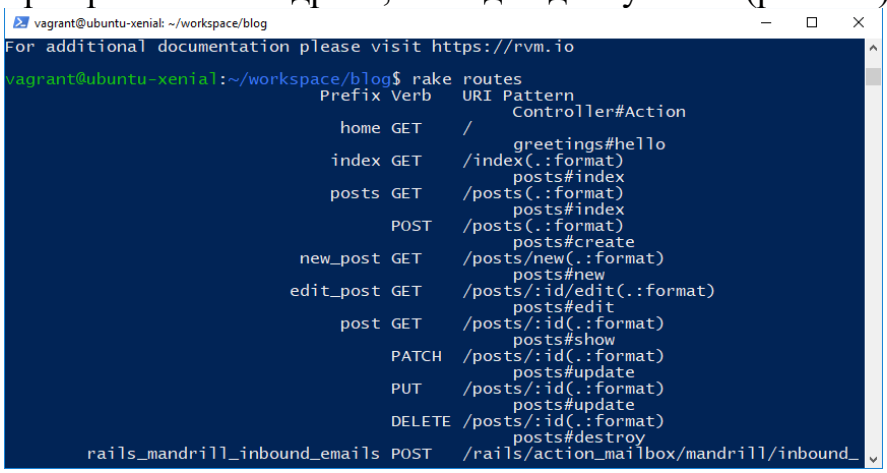

Рисунок 4.1 – Приклад нової маршрутизації

`home GET /` відслідковує головну сторінку, `posts GET /` відслідковує сторінки, які називаються `post`. Таким чином, ми створили URL адреси сторінок. Створенні URL адреси використаємо для відображення, зберігання, створення видалення статей тощо.

2.4 Переходимо у директорію app/controllers/posts_controller.rb і створюємо новий метод

```
def new #створюємо новий пост
end
```

2.5 Переходимо у директорію app/views/posts, створюємо файл new.html.erb і записуємо в нього код

```
<h2><Додати новий пост></h2>
```

2.6 Запускаєм браузер, переходимо за адресою localhost:3000/posts/new (рис. 4.2).

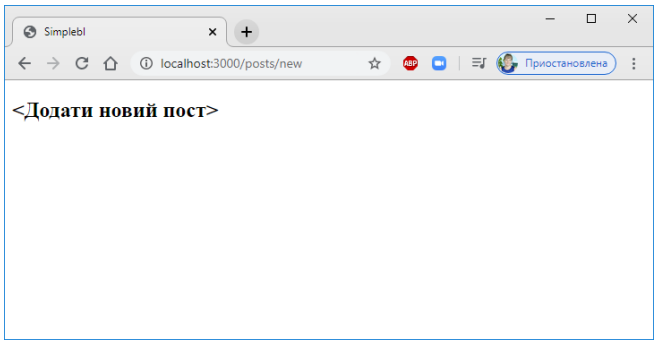

Рисунок 4.2 – Тестова сторінка

2.7 Створюємо форму з якої будем використовувати дані, зокрема, поля назва і текст статті. Звертаємося до форми за назвою post, а до полів за допомогою змінної f. В файл app/view/posts/new.html.erb додаємо код

```
<%= form_for :post, url: posts_path do |f|%>
<p>
        Назва <br>
        <%= f.text_field :title %>
</p>
<p>
        Текст статті <br>
        <%= f.text_area :body %>
</p>
<p>
        <%= f.submit %>
</p>
<% end %>
```

2.8 Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/new (рис. 4.3).

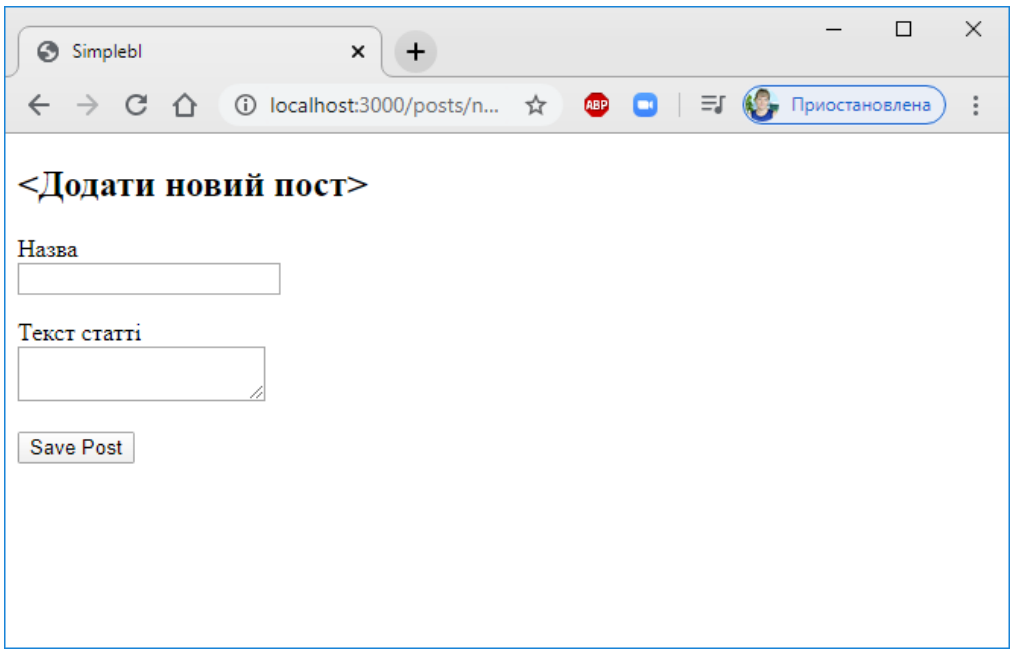

Рисунок 4.3 – Тестова сторінка

2.9 Переходимо у файл app/controllers/posts_controller.rb і створюємо новий метод, який буде спрацьовувати при створенні нового поста. В цьому ж методі записуємо тестову функцію, для перевірки даних, які додаються в пост.

```
def create
        render plain: params[:post].inspect
end
```

2.10 Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/new (рис. 4.4). Все працює вірно.

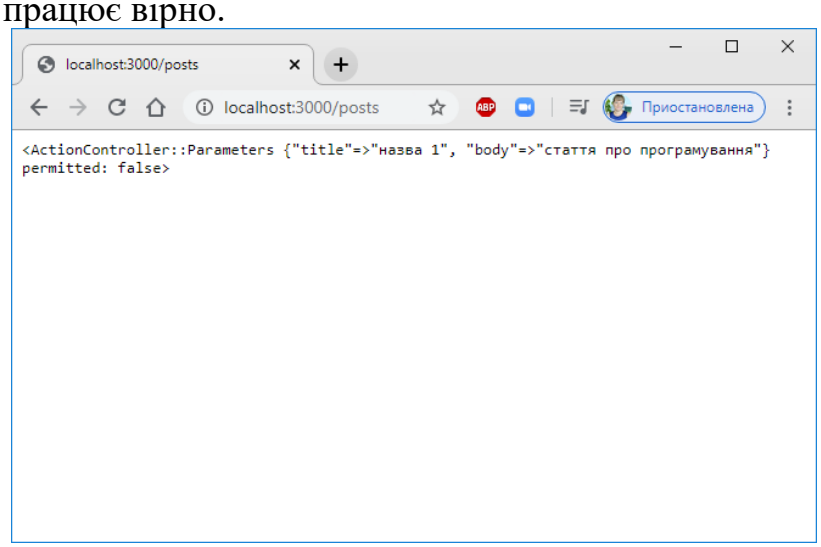

Рисунок 4.4 – Тестова сторінка

3. Створення бази даних.

В Ruby on Rails є база даних, яка називається SQLite. Можна працювати з іншими базами даних, але для уникнення проблем використовувати їх не рекомендують, бо вони будуть відрізнятися стилем створення та наповненням таблиць, виведенням інформації тощо.

3.1 Переходимо у консоль PowerShell (друга консоль), вводимо команду $rails g model Post title:string body:text і створюємо модель в якій будемо зберігати базу даних. Post – це назва моделі, title:string, body:text – це типізовані параметри моделі (поля форми), які будуть зберігатися у базі даних.

3.2 В редакторі коду перевіряємо, чи створився файл міграції db/migrate/20200409183749_create_posts.rb. Даний файл відповідає за таблицю бази даних, але сама таблиця бази даних ще не створена, тобто працювати із базою даних поки що не можна.

3.3 Переходимо у консоль PowerShell (друга консоль) і, для того, щоб створити таблицю, виконуємо міграцію командою $rake db:migrate

Тепер при створенні поста його можна розміщувати в базі даних, зберігати в базу даних і перезавантажувати сторінку для відображення нового поста.

3.4 Переходимо у файл app/controllers/posts_controller.rb і додаємо код

```
def show
    @post = Post.find(params[:id]) #шукаємо необхідний пост в базі даних і беремо його по ID
end

def create
    #render plain: params[:post].inspect

    @post=Post.new(post_params) # передаємо в метод дані із дозволених полів

    @post.save # зберігаємо пост в базі даних
    redirect_to @post #викликаємо метод для переадресації користувача на нову сторінку
end

    private def post_params
    params.require(:post).permit(:title, :body)
  end
```

3.5 Переходимо у файл app/view/posts/show.html.erb і додаємо код

```
<h1><%= @post.title %></h1>
<h1><%= @post.body %></h1>
```

3.6 Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/new (рис. 4.5-4.6). Все працює вірно.

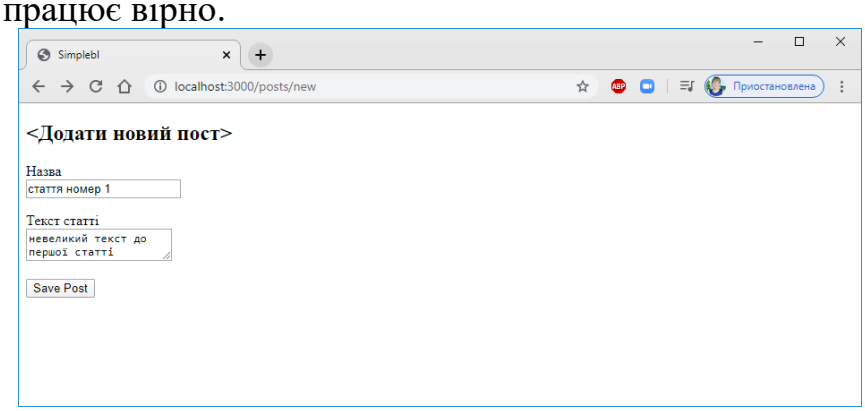

Рисунок 4.5 – Тестова сторінка

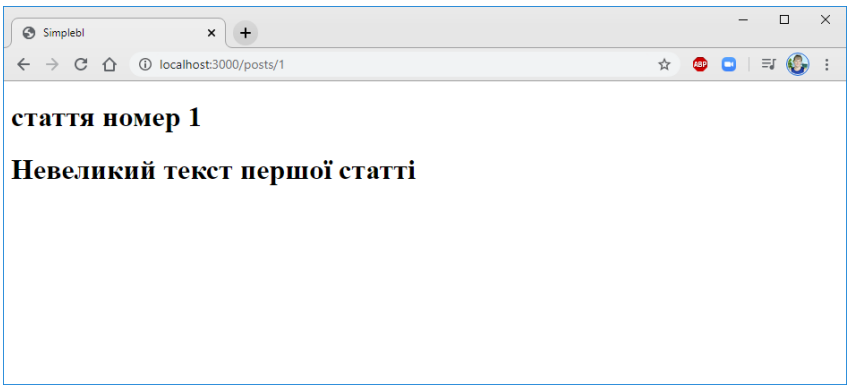

Рисунок 4.6 – Тестова сторінка

3.7 Переходимо у файл app/view/posts/index.html.erb і код

```
<h4><%= @heading %></h4>
<p4><%= @text %></p4>
```

замінюємо на

```
<h1><Все наши пости></h1>
<% @post.each do |post| %>
<h3><%= post.title %></h3>
<p><%= post.body %></p>
<% end %>
```

3.8 Переходимо в браузер, оновлюємо сторінку localhost:3000/index (рис. 4.7). Все працює вірно.

Рисунок 4.7 – Тестова сторінка

4. Додавання Bootstrap стилів до сайту.

4.1 В браузері переходимо на сайт https://www.bootstrapcdn.com/, копіюємо посилання на css стилі https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css і додаємо його у файл app/view/layouts/application.html.erb (рис. 4.8).

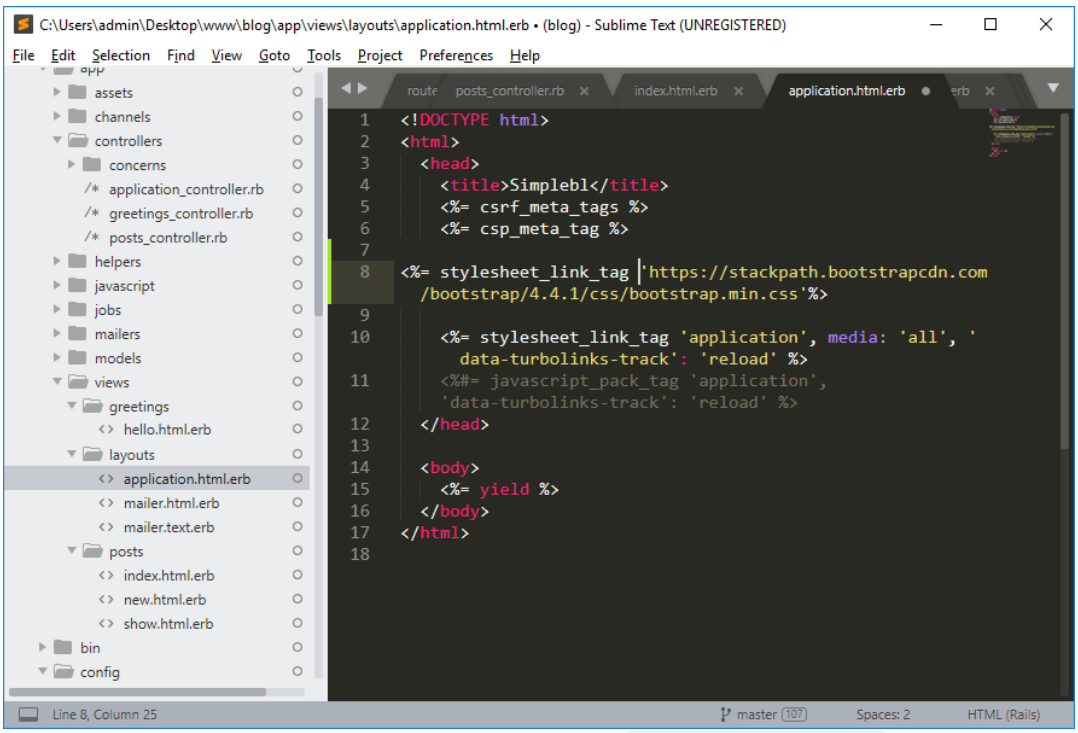

Рисунок 4.8 – Вміст файла application.html.erb

4.2 Переходимо в браузер, оновлюємо сторінку localhost:3000/index (рис. 4.9). Стиль сайту змінився.

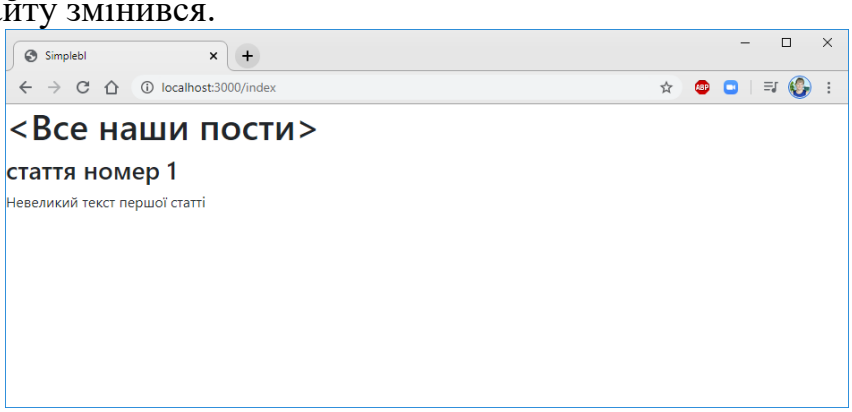

Рисунок 4.9 – Тестова сторінка

4.3 Повертаємо до файлу app/view/layouts/application.html.erb і додаємо код для модифікації верхньої частини сайту

```
<div class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-white border-bottom shadow-sm">
  <h5 class="my-0 mr-md-auto font-weight-bold">Об'єктно-орієнтоване програмування</h5>
  <nav class="my-2 my-md-0 mr-md-3">
    <span class="p-2 text-dark"><%= link_to "Головна сторінка", home_path%></span>
     <span class="p-2 text-dark"><%= link_to "Статті", index_path%></span>
  </nav>
  <span class="p-2 text-dark"><%= link_to "Додати статтю", new_post_path %></span>
</div>

<div class="container">
  <%= yield %>
```

```
</div>
```

4.4 Переходимо до файлу config/webpack/routes.rb і додаємо код

```
root 'greetings#hello', as: 'home'
```

4.5 Переходимо в браузер, оновлюємо сторінки localhost:3000 і localhost:3000/posts/new (рис. 4.10-4.11). Верхня частина сайту змінилася.

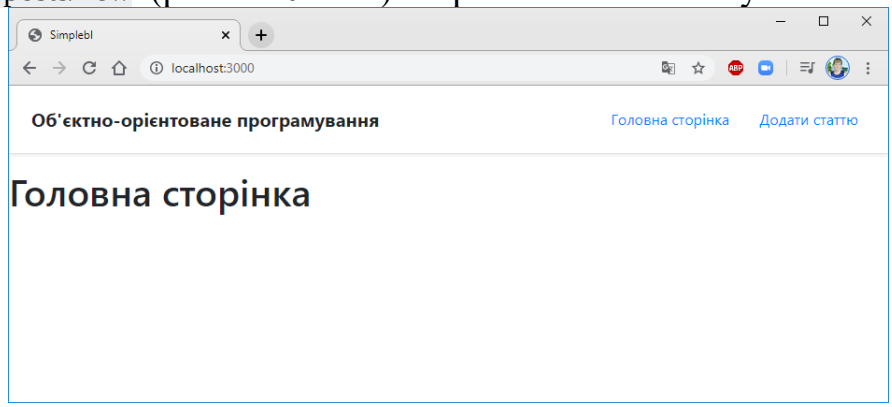

Рисунок 4.10 – Тестова сторінка

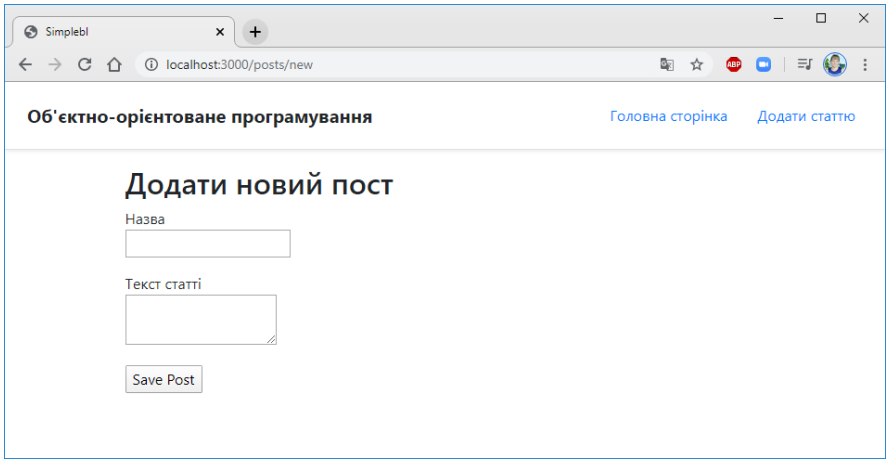

Рисунок 4.11 – Тестова сторінка

4.6 Повертаємося до файлу config/webpack/routes.rb і додаємо код

```
get 'index'=>'posts#index', as:'index'
```

4.7 Модифікуємо стиль форми для додавання статті. Переходимо до файлу app/view/posts/new.html.erb і додаємо код

```
<h2>Додати новий пост</h2>
<%= form_for :post, url: posts_path do |f|%>
<p>
        Назва <br>
        <%= f.text_field(:title, {:class =>'form-control'}) %>
</p>
<p>
        Текст статті <br>
        <%= f.text_area(:body, {:class =>'form-control'}) %>
</p>
<p>
        <%= f.submit({:class =>'btn btn-success'})%>
</p>
<% end %>
```

4.8 Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/new (рис. 4.12).

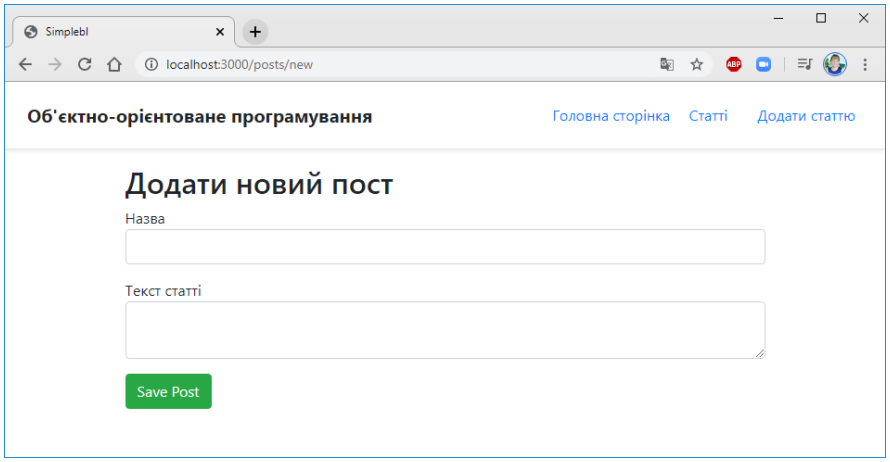

Рисунок 4.12 – Тестова сторінка

4.9 Повертаємося до першої консолі PowerShell. Для зупинки сервера натискаємо Ctrl+C.

4.10 В консолі для виходу із робочої папки вводимо команду exit.

4.11 В консолі для зупинки віртуальної машини вводимо команду vagrant halt.

4.12 Переходимо до другої консолі, виходимо із робочої папки і зупиняємо віртуальну машину.

Завдання до роботи

1. В Ruby on Rails проекті створити динамічні сторінки сайту згідно свого варіанту.
2. В Ruby on Rails проекті створити бази даних для сайту згідно свого варіанту.
3. В Ruby on Rails проекті додати bootstrap стилі до сайту згідно свого варіанту.

Варіанти завдань

1. Web-орієнтована інформаційна система садового центру.
2. Web-орієнтована інформаційна система туристичної агенції.
3. Web-орієнтована інформаційна система поліграфічного підприємства.
4. Web-орієнтована інформаційна система рекламного агентства.
5. Web-орієнтована інформаційна система салону краси.
6. Web-орієнтована інформаційна система меблевого салону.
7. Web-орієнтована інформаційна система кадрового агентства.

Зміст звіту

1. Титульний аркуш.
2. Тема і мета роботи.
3. Короткі теоретичні відомості.

4. Протокол виконання завдання №1.
5. Протокол виконання завдання №2.
6. Протокол виконання завдання №3.
7. Висновки.

Контрольні питання

1. Опишіть процес створення динамічних сторінок сайту у фреймворку Ruby on Rails.
2. Поясніть призначення моделі у фреймворку Ruby on Rails.
3. Опишіть процес створення бази даних у фреймворку Ruby on Rails.
4. Опишіть процес створення таблиці у фреймворку Ruby on Rails.
5. Опишіть процес додавання bootstrap стилів до сайту.

ЛАБОРАТОРНА РОБОТА № 5

Валідація форм у фреймворку Ruby on Rails

Мета роботи: Закріпити набуті теоретичні знання та набути практичні навички валідації форм і розширення функціоналу сайту у фреймворку Ruby on Rails.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.
3. Виконати завдання до лабораторної роботи згідно свого варіанту.
4. Скласти та оформити звіт.

Теоретичні відомості

Подовжуємо розглядати приклад створення веб-блогу за допомогою фреймворку Ruby on Rails.

1. Запуск віртуальної машини.

1.1 Відкриваємо консоль PowerShell (перша консоль).

1.2 В консолі вводимо команду `vagrant up` для запуску віртуальної машини.

1.3 Для встановлення ssh зв’язку із віртуальною машиною вводимо команду `vagrant ssh`.

1.4 Переходимо у папку, в якій знаходиться проект на Ruby on Rails, наприклад, `$ cd workspace/simplebl`.

1.5 Запускаємо сервер командою `$ rails server -b 0.0.0.0` і згортаємо консоль.

1.6 Відкриваємо ще одну консоль PowerShell (друга консоль).

1.7 Встановлюємо ssh зв’язок із віртуальною машиною командою `vagrant ssh`.

1.8 Переходимо у папку, в якій знаходиться проект на Ruby on Rails, наприклад, `$ cd workspace/simplebl`, і згортаємо консоль.

2. Валідація сторінки для додавання нових статей.

2.1 У редакторі коду Sublime Text або Atom відкриваємо папку проекту.

2.2 Перевіряємо коректність введення назви статті. Поле для введення назви статті не повинно бути порожнім, а назва статті має містити щонайменше 5 символів.

Переходимо у файл app/models/post.rb і додаємо код

```
class Post < ApplicationRecord
	validates :title, presence:true, length: {minimum: 5}
	# presence:true перевірка буде викликатися автоматично
end
```

2.3 Переходимо у файл app/controllers/posts_controller.rb і додаємо код

```
def create
	#render plain: params[:post].inspect
	@post=Post.new(post_params)
	if(@post.save)
	redirect_to @post
else
	render 'new' # сторінка перезавантажиться
end
```

if(@post.save) якщо ми можемо зберегти пост, то ми переадресовуємо користувача на метод show, який показує сторінку із статтею, яка тільки-но була створена. Зберегти пост можна, якщо в полі назва статті вказані символи і цих символів більше 5.

2.4 Створюємо повідомлення про помилки. Переходимо у файл app/views/posts/new.html.erb і додаємо код

```
<% if @post.errors.any? %>
<% @post.errors.full_messages.each do |msg|%>
<div class="alert alert-danger"><%= msg %></div>
<% end %>
<% end %>
```

2.5 Переходимо у файл app/controllers/posts_controller.rb і додаємо код

```
def new
@post = Post.new
end
```

2.6 Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/new (рис. 5.1).

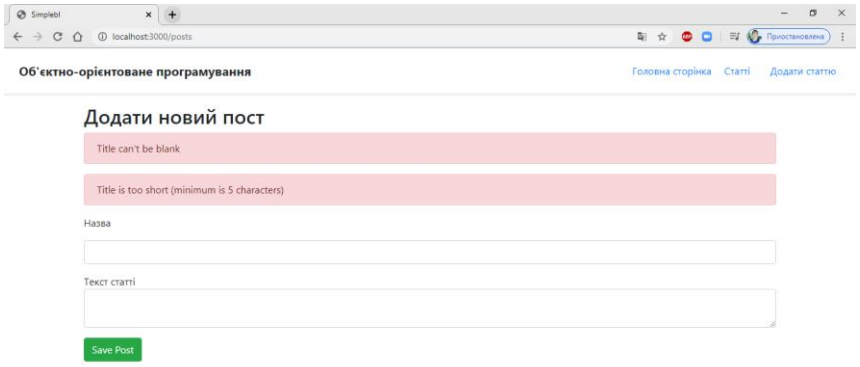

Рисунок 5.1 – Тестова сторінка

2.7 Виконаємо валідацію головної сторінки. Переходимо у файл app/views/posts/index.html.erb і додаємо код

```
<h1>Статті</h1>
<% @post.each do |post| %>
<div class="alert alert-light">
<h3><%= post.title %></h3>
<p><%= post.body %></p>
<%= link_to "Читати далі", post_path(post), :class => 'btn btn-outline-primary'%>
<% end %>
```

2.8 Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/new (рис. 5.2).

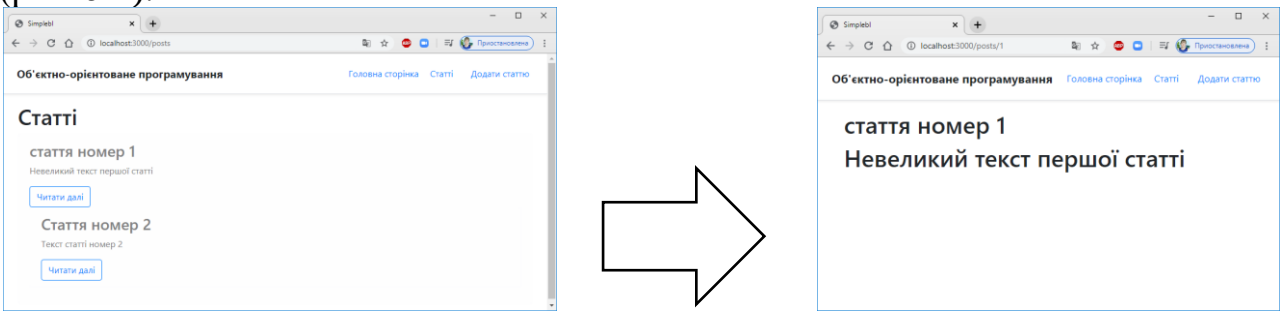

Рисунок 5.2 – Тестові сторінки

3. Редагування та видалення нових статей.

3.1 Переходимо у файл app/views/posts/show.html.erb і створюємо кнопку «Редагувати»

```
<hr>
<%= link_to "Редагувати", edit_post_path(@post), :class =>'btn btn-warning'%>
```

Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/1 (рис. 5.3).

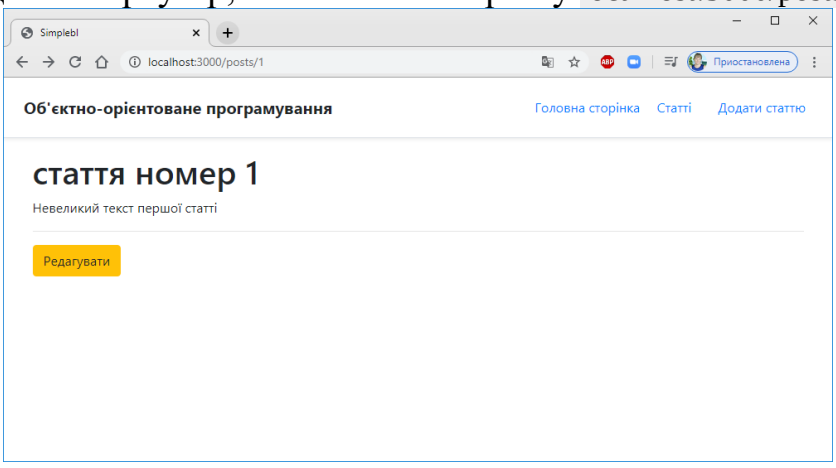

Рисунок 5.3 – Тестова сторінка

3.2 Переходимо у файл app/controllers/posts_controller.rb і додаємо метод редагування статті

```
def edit
    @post = Post.find(params[:id])
end
```

3.3 Переходимо у директорію app/views/posts, створюємо файл edit.html.erb і записуємо в нього код

```
<h2> Редагування статті </h2>
<%= form_for :post, url: post_path(@post), method: :patch do |f|%>
<% if @post.errors.any? %>
<% @post.errors.full_messages.each do |msg|%>
<div class="alert alert-danger"><%= msg %></div>
<% end %>
<% end %>

<p>
        Назва <br>
        <%= f.text_field(:title, {:class =>'form-control'}) %>
</p>
<p>
        Текст статті <br>
        <%= f.text_area(:body, {:class =>'form-control'}) %>
</p>
<p>
        <%= f.submit({:class =>'btn btn-success'})%>
</p>
<% end %>
```

Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/1/edit (рис. 5.4).

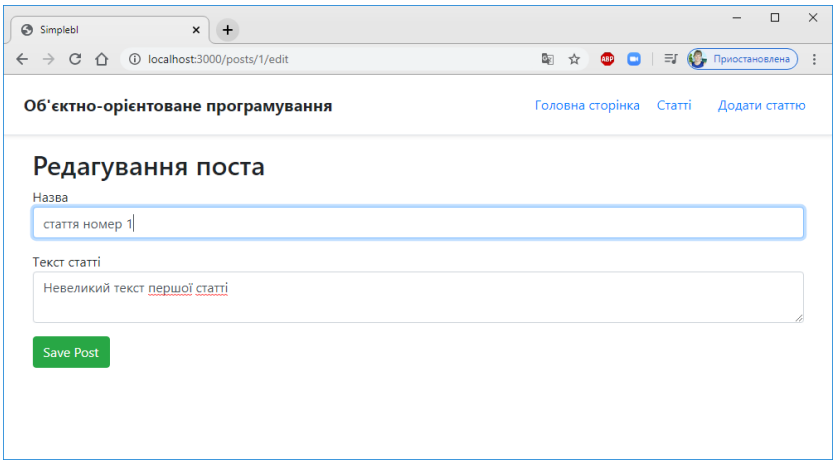

Рисунок 5.4 – Тестова сторінка

3.4 Створюємо метод оновлення статті. Переходимо у файл app/controllers/posts_controller.rb і додаємо код

```
def update
        @post = Post.find(params[:id])
if(@post.update(post_params))
        redirect_to @post
else
        render 'edit'
end
end
```

Якщо стаття оновлюється правильно, тобто не видає помилки, то користувач скеровується на метод show – відображення статті. Якщо стаття додається із помилками, користувач переадресовується на редагування статті (рис. 5.5).

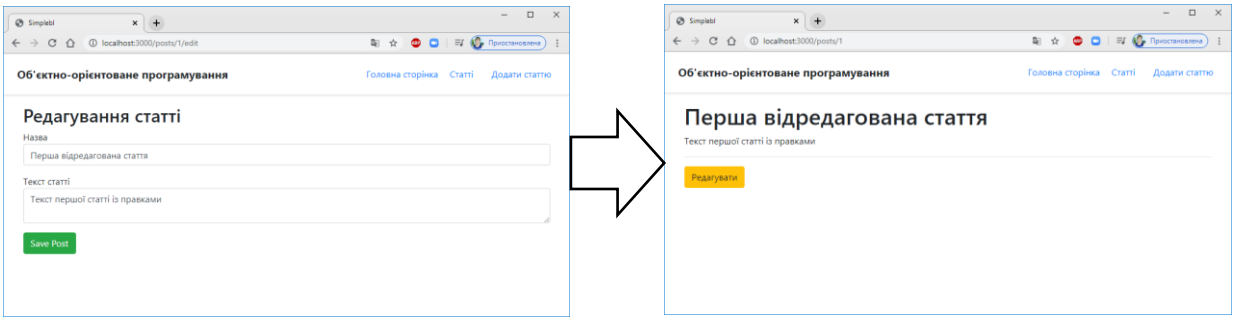

Рисунок 5.5 – Тестові сторінки

3.5 Створюємо метод для видалення статті. Переходимо у файл app/controllers/posts_controller.rb і додаємо код

```
def destroy
    @post = Post.find(params[:id])
    @post.destroy
    redirect_to posts_path
end
```

3.6 Переходимо у файл app/views/posts/show.html.erb і створюємо кнопку «Видалити»

```
<%= link_to "Видалити", post_path(@post), method: :delete, data: {confirm: "Видалити статтю?"}, :class =>'btn btn-danger'%>
```

Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/1 (рис. 5.6).

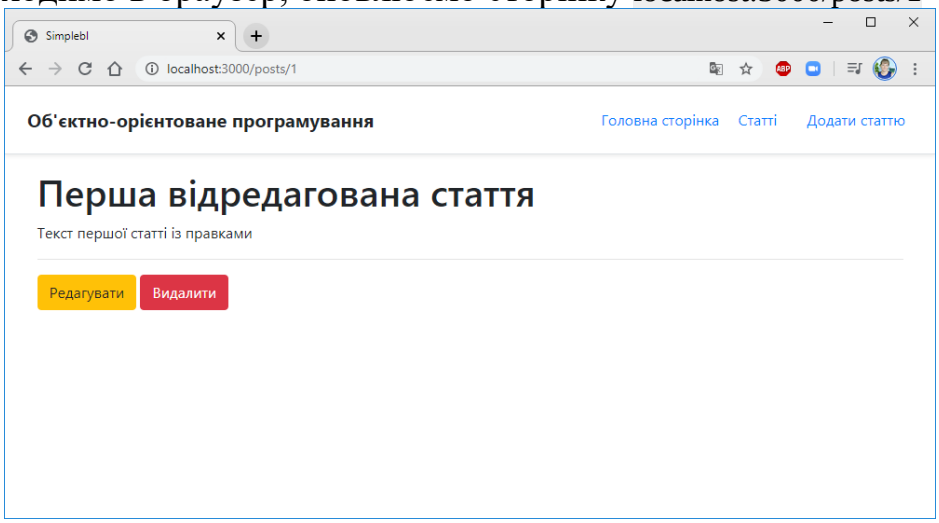

Рисунок 5.6 – Тестова сторінка

3.7 Повертаємося до першої консолі PowerShell. Для зупинки сервера натискаємо Ctrl+C.

3.8 В консолі для виходу із робочої папки вводимо команду exit.

3.9 В консолі для зупинки віртуальної машини вводимо команду vagrant halt.

Завдання до роботи

1. В Ruby on Rails проекті виконати валідацію сторінок сайту згідно свого варіанту.

2. В Ruby on Rails проекті розширити функціонал сайту згідно свого варіанту.

Варіанти завдань

1. Web-орієнтована інформаційна система садового центру.
2. Web-орієнтована інформаційна система туристичної агенції.
3. Web-орієнтована інформаційна система поліграфічного підприємства.
4. Web-орієнтована інформаційна система рекламного агентства.
5. Web-орієнтована інформаційна система салону краси.
6. Web-орієнтована інформаційна система меблевого салону.
7. Web-орієнтована інформаційна система кадрового агентства.

Зміст звіту

1. Титульний аркуш.
2. Тема і мета роботи.
3. Короткі теоретичні відомості.
4. Протокол виконання завдання №1.
5. Протокол виконання завдання №2.
6. Висновки.

Контрольні питання

1. Поясність суть процесу валідації форм.
2. Де зберігаються дані, які приходять з форм у вигляді параметрів?
3. Опишіть процес передачі параметрів для зв’язаних моделей у фреймворку Ruby on Rails.
4. Опишіть процес одержання даних контролером у фреймворку Ruby on Rails.
5. Опишіть процес ручного створення методів у контролерах без використання генератора коду.

ЛАБОРАТОРНА РОБОТА № 6

Debugging і оптимізація коду у фреймворку Ruby on Rails

Мета роботи: Закріпити набуті теоретичні знання та набути практичні навички оптимізації коду у фреймворку Ruby on Rails.

Порядок виконання роботи

1. Вивчити теоретичні відомості.
2. Разом з викладачем вибрати варіант завдання.
3. Виконати завдання до лабораторної роботи згідно свого варіанту.
4. Скласти та оформити звіт.

Теоретичні відомості

Подовжуємо розглядати приклад створення веб-блогу за допомогою фреймворку Ruby on Rails.

1. Запуск віртуальної машини.

1.1 Відкриваємо консоль PowerShell (перша консоль).

1.2 В консолі вводимо команду vagrant up для запуску віртуальної машини.

1.3 Для встановлення ssh зв'язку із віртуальною машиною вводимо команду vagrant ssh.

1.4 Переходимо у папку, в якій знаходиться проект на Ruby on Rails, наприклад, $ cd workspace/simplebl.

1.5 Запускаємо сервер командою $ rails server -b 0.0.0.0 і згортаємо консоль.

1.6 Відкриваємо ще одну консоль PowerShell (друга консоль).

1.7 Встановлюємо ssh зв'язок із віртуальною машиною командою vagrant ssh.

1.8 Переходимо у папку, в якій знаходиться проект на Ruby on Rails, наприклад, $ cd workspace/simplebl.

2. Додавання коментарів до статей.

2.1 У консолі PowerShell (друга консоль), вводимо команду $ rails g model Comment username:string body:text post:references і створюємо модель в якій будемо зберігати базу даних коментарів. Comment – це назва моделі, username:string body:text post:references – це типізовані параметри моделі (поля форми), які будуть зберігатися у базі даних. username:string – імя користувача, body:text – зміст коментар, post:references – прив'язка до статті.

2.2 У редакторі коду Sublime Text або Atom відкриваємо папку проекту і перевіряємо, чи створився файл міграції db/migrate/20200410182718_create_comments.rb. Даний файл відповідає за таблицю бази даних, але сама таблиця бази даних коментарів ще не створена, тобто працювати із базою даних поки що не можна.

2.3 Переходимо у консоль PowerShell (друга консоль) і, для того, щоб створити таблицю, виконуємо міграцію командою $rake db:migrate

Тепер при створенні коментаря його можна розміщувати в базі даних, зберігати в базу даних і перезавантажувати сторінку для відображення нового коментаря.

2.4 Встановлюємо взаємозв'язок між статтями і коментарями. У редакторі коду переходимо у файл app/models/comment.rb і додаємо код

```
class Comment < ApplicationRecord
  belongs_to :post #коментарі належать певним статтям
end
```

2.5 Переходимо у файл app/models/post.rb і додаємо код, який вказує, що статті можуть мати багато коментарів

```
class Post < ApplicationRecord
        has_many :comments
        validates :title, presence:true, length: {minimum: 5}
        # presence:true перевірка буде викликатися автоматичного
end
```

2.6 Переходимо у файл config/webpack/routes.rb і додаємо код для відслідковування URL адрес (посилань)

```
resources :posts do
        resources :comments
end
```

2.7 У консолі PowerShell (друга консоль), створюємо контролер, який називається Comments, командою

```
$ rails generate controller Comments
```

2.8 Переходимо у файл app/views/posts/show.html.erb і створюємо кнопку «Коментувати»

```
<hr>
<h2> Коментувати </h2>
<%= form_for([@post, @post.comments.build]) do |f|%>
<p>
        Користувач <br>
        <%= f.text_field(:username, {:class =>'form-control'}) %>
</p>
<p>
        Текст коментаря <br>
        <%= f.text_area(:body, {:class =>'form-control'}) %>
</p>
<p>
        <%= f.submit("Додати коментар", {:class =>'btn btn-success'})%>
</p>
<% end %>
```

Переходимо в браузер, оновлюємо сторінку localhost:3000/posts/1 (рис. 6.1).

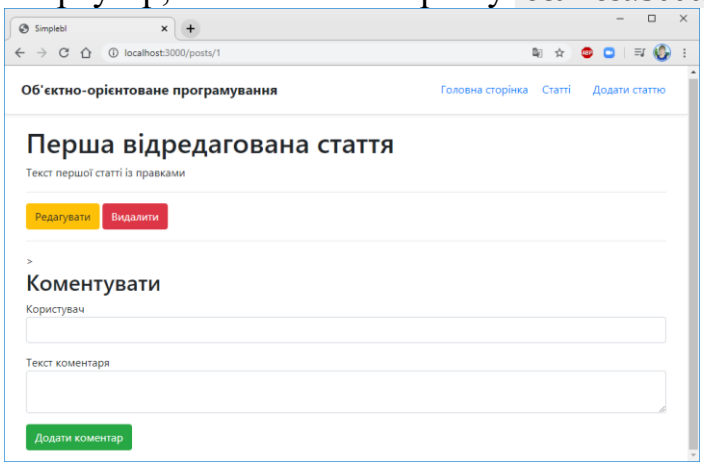

Рисунок 6.1 – Тестова сторінка

2.8 У редакторі коду відкриваємо файл app/controllers/comments_controller.rb і створюємо метод для створення коментарів

```
class CommentsController < ApplicationController
        def create
        @post = Post.find(params[:post_id])
        @comment = @post.comments.create (comment_params)
redirect_to post_path(@post)
        end
```

```
        private def comment_params
                params.require(:comment).permit(:usename, :body)
        end
end
```

2.9 Переходимо у файл app/views/posts/show.html.erb і створюємо функціонал для відображення коментарів

```
<h2>Всі коментарі</h2>
<% @post.comments.each do |comment| %>
<div class="alert alert-light">
<p><strong><%= comment.username %></strong>: <%= comment.body %></p>
</div>
<% end %>
```

2.10 Переходимо у браузер і оновлюємо сторінку.

3. Авторизація і реєстрація на сайті.

Додаємо перевірку, чи зареєстрований користувач. Якщо він авторизується, то він зможе додавати, редагувати і видаляти статті. В протилежному випадку будь-які дії заборонені.

У редакторі коду відкриваємо файл app/controllers/posts_controller.rb і створюємо метод для виконання авторизації користувачів.

Завдання до роботи

1. В Ruby on Rails проекті реалізувати RSS-фід для сайту згідно свого варіанту.
2. В Ruby on Rails проекті реалізувати інструкцію для пошукових машин про те, що не варто індексувати.
3. В Ruby on Rails проекті додати функціонал для авторизації користувачів сайту згідно свого варіанту.

Варіанти завдань

1. Web-орієнтована інформаційна система садового центру.
2. Web-орієнтована інформаційна система туристичної агенції.
3. Web-орієнтована інформаційна система поліграфічного підприємства.
4. Web-орієнтована інформаційна система рекламного агентства.
5. Web-орієнтована інформаційна система салону краси.
6. Web-орієнтована інформаційна система меблевого салону.
7. Web-орієнтована інформаційна система кадрового агентства.

Зміст звіту

1. Титульний аркуш.
2. Тема і мета роботи.
3. Короткі теоретичні відомості.
4. Протокол виконання завдання №1.
5. Протокол виконання завдання №2.
6. Протокол виконання завдання №3.
7. Висновки.

Контрольні питання

1. Опишіть процес реалізації RSS-фіду у фреймворку Ruby on Rails.
2. Опишіть процес створення карти сайту за допомогою гема DynamicSitemaps у фреймворку Ruby on Rails.
3. Опишіть процес створення у фреймворку Ruby on Rails інструкції для пошукових машин про те, що не варто індексувати.
4. Опишіть процес реалізації авторизації користувачів сайту у фреймворку Ruby on Rails.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Хартл М. Ruby on Rails для начинающих. СпБ.: ДМК Пресс, 2016. 572 с.

2. Фултон Х. Программирование на языке Ruby. М.: ДМК-Пресс, 2007. 688 с.

3. Фернандес О. Пусть Rails. Подробное руководство по созданию приложений в среде Ruby on Rails. Пер. с англ. СПб.: Символ-Плюс, 2009. 768 с.

4. Симдянов И. Самоучитель Ruby. СПб.: БХВ-Петербург, 2016. 656 с.

5. Мова програмування Ruby і середовище програмування. URL: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/43_Ruby/index.html (дата звернення: 27.07.2019).

6. Ruby on Rails на русском. URL: http://blog.topolyan.com/tag/ruby/ (дата обращения: 18.08.2019).

7. Ruby on Rails на Windows: огляд проблеми та налаштування за допомогою Vagrant і Virtual. URL: Box https://codeguida.com/post/1265 (дата звернення: 27.07.2019).

8. Уроки Ruby on Rails. URL: https://itproger.com/course/ruby-on-rails (дата обращения: 01.07.2019).