

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ В. Н. КАРАЗІНА
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК

КОНТРОЛЬНА РОБОТА
ДИСЦИПЛІНА: «МОВИ ПРИКЛАДНОГО ПРОГРАМУВАННЯ»
ВАРІАНТ 2

Виконав:

Студент групи КС31

Касьяненко М.М.

Перевірив:

Паршенцев Б.В.

ЗМІСТ

Завдання.....	3
Теоретичні.....	3
Прктичні.....	3
Хід Роботи.....	4
Теоретичне завдання 1.....	4
Відповідь.....	4
Теоретичне завдання 2.....	5
Відповідь.....	5
Теоретичне завдання 3.....	7
Відповідь.....	7
Пракичне завдання 1.....	8
Відповідь.....	9
Пракичне завдання 2.....	10
Відповідь.....	10

ЗАВДАННЯ

Теоретичні

1. Як у Ruby відбувається обробка помилок та винятків (exceptions)
2. Які переваги має використання блоків (blocks) замість циклів (loops)?
3. Що таке модифікатори доступу (access modifiers) у Ruby і які значення вони можуть набувати?

Практичні

1. Напишіть функцію, яка набирає масив цілих чисел і повертає середнє арифметичне значення елементів масиву.
2. Створіть простий додаток "Нотатки", який дозволяє користувачам створювати, переглядати, редагувати та видаляти нотатки. Кожна нотатка повинна мати заголовок та текст.

ХІД РОБОТИ

Теоретичне завдання 1

Як у Ruby відбувається обробка помилок та винятків (exceptions)

Відповідь

У Ruby ви можете обробляти винятки за допомогою ключових слів `begin`, `rescue` та `end`.

```
1 begin
2   # code that might raise an exception
3 rescue
4   # code to handle the exception
5 end
```

Працює це наступним чином:

1. Ключове слово `begin` запускає блок коду, який може згенерувати виключення.
2. Якщо у блоці `begin` виникне виключення, Ruby перейде до блоку `rescue`.
3. Блок `rescue` містить код для обробки виключення.
4. Ключове слово `end` позначає кінець початкового блоку і кінець коду обробки винятків.
5. Ви також можете вказати тип виключення, яке ви хочете обробити за допомогою ключового слова `rescue`, наприклад, так:

```
1 begin
2   # code that might raise an exception
3 rescue TypeError => e
4   # code to handle a TypeError
5 rescue ZeroDivisionError => e
```

```
6   # code to handle a ZeroDivisionError
7   end
```

Тут ми маємо два блоки `rescue`, кожен з яких обробляє різні типи винятків. Синтаксис `=> e` створює змінну `e`, яка міститиме об'єкт виключення, за допомогою якого ви можете отримати більше інформації про виключення.

Якщо ви хочете, щоб код виконувався незалежно від того, було згенеровано виключення чи ні, ви можете використати ключове слово `ensure`, наприклад, так:

```
1   begin
2     # code that might raise an exception
3   rescue
4     # code to handle the exception
5   ensure
6     # code that will always run, whether an exception was raised or
not
7   end
```

Це корисно для звільнення ресурсів, закриття файлів або зняття блокування, незалежно від того, стався виняток чи ні.

Теоретичне завдання 2

Які переваги має використання блоків (blocks) замість циклів (loops)?

Відповідь

Блоки і цикли в Ruby використовуються для різних цілей і мають різний синтаксис і функціональність.

Блок - це фрагмент коду, який можна передати методу як аргумент, і він може бути виконаний один або багато разів у цьому методі. Блоки беруться у фігурні дужки `do...end` або `{...}` і можуть приймати аргументи за допомогою синтаксису `pipe |...|`.

Блоки часто використовуються для визначення спеціальної поведінки методів або ітераторів.

```
1 array = [1, 2, 3]
2 new_array = array.map do |num|
3   num * 2
4 end
5 puts new_array #=> [2, 4, 6]
```

У цьому прикладі блок отримує аргумент `num` і повертає результат його множення на 2. Потім метод `map` ітераційно перебирає кожен елемент масиву і виконує блок для кожного елемента, створюючи новий масив з результатами.

З іншого боку, цикл - це конструкція, яка повторює блок коду певну кількість разів або до тих пір, поки не буде виконано певну умову. Ruby надає кілька типів циклів, таких як `while`, `until`, `for` та `each`. Ось приклад циклу `while`, який виводить числа від 1 до 5:

```
1 i = 1
2 while i <= 5 do
3   puts i
4   i += 1
5 end
```

У цьому прикладі блок коду всередині циклу `while` буде виконуватися до тих пір, поки виконується умова `i <= 5`. На кожній ітерації значення `i` збільшується на 1, поки не досягне 6 і цикл не завершиться.

Таким чином, блоки використовуються для визначення поведінки, яка буде виконуватися методами або ітераторами, в той час як цикли використовуються для повторення блоку коду певну кількість разів або до тих пір, поки не буде виконана

певна умова. Хоча блоки можна передавати як аргументи до циклів, вони не є взаємозамінними з циклами і служать для різних цілей.

Теоретичне завдання 3

Що таке модифікатори доступу (access modifiers) у Ruby і які значення вони можуть набувати?

Відповідь

Модифікатори доступу в Ruby - це ключові слова, які керують видимістю методів або змінних у класах. Вони використовуються, щоб обмежити або дозволити доступ до певних частин класу з інших частин програми.

У Ruby є три модифікатори доступу: `public`, `private` та `protected`. Ось короткий огляд кожного з них:

- `public`: Методи та змінні, позначені як `public`, можуть бути викликані з будь-якого місця, як всередині класу, так і за його межами. За замовчуванням усі методи та змінні в класі є загальнодоступними, якщо не вказано інше.
- `private`: Методи та змінні, позначені як `private`, можуть бути викликані тільки зсередини класу. До них не можна отримати доступ ззовні класу, навіть з екземплярів того самого класу. Закриті методи часто використовуються для внутрішніх деталей реалізації, які не призначені для зовнішнього світу.
- `protected`: Методи та змінні, позначені як `protected`, можна викликати з класу або його підкласів. До них не можна отримати доступ ззовні класу, навіть з екземплярів того самого класу. Захищені методи часто використовуються для забезпечення спільної реалізації класу та його підкласів.

Щоб вказати модифікатор доступу в Ruby, ви можете використати одне з наступних ключових слів, за яким слідує визначення методу або змінної:

```
1  class MyClass
2    def public_method
3      # This method can be called from anywhere
4    end
5
```

```
6     private
7
8     def private_method
9         # This method can only be called from within the class
10    end
11
12    protected
13
14    def protected_method
15        # This method can only be called from within the class or
its subclasses
16    end
17 end
```

У цьому прикладі `public_method` за замовчуванням позначений як `public`, а `private_method` і `protected_method` позначені як `private` і `protected` відповідно за допомогою ключових слів `private` і `protected`.

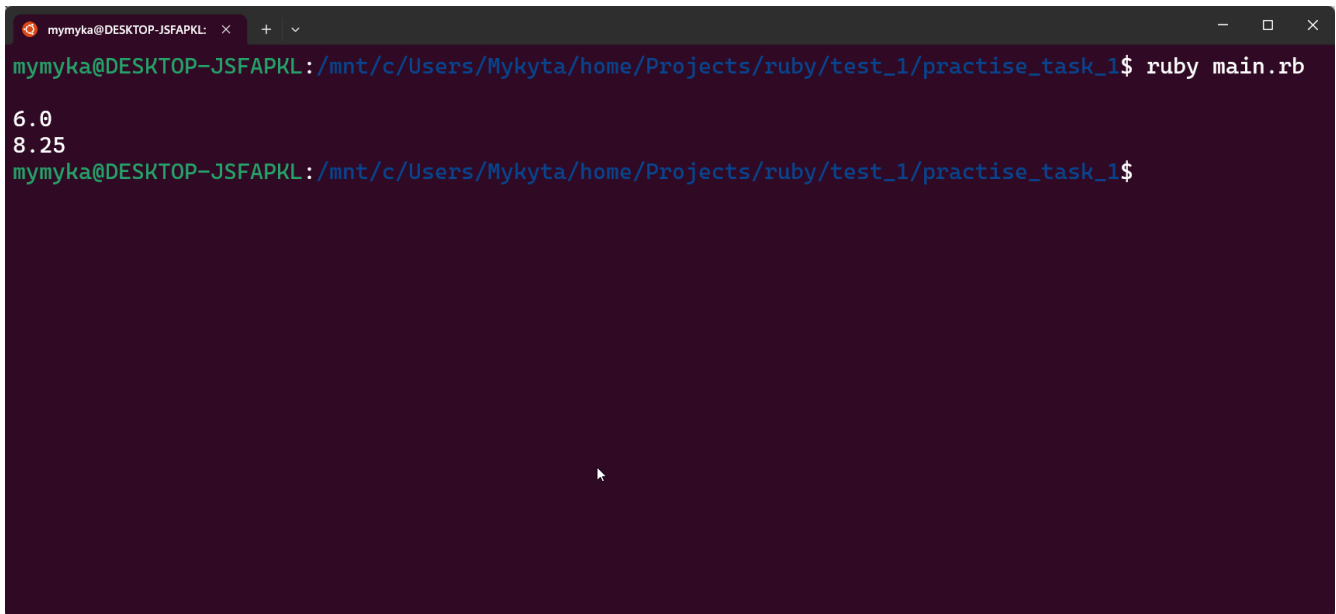
Варто зазначити, що модифікатори доступу можна перевизначати у підкласах, і що вони застосовуються лише до методів та змінних екземплярів, а не до методів класу чи змінних класу. Крім того, сімейство методів `attr_*`, таких як `attr_reader`, `attr_writer` та `attr_accessor`, можна використовувати для визначення загальнодоступних аксесорів для змінних екземплярів без явного визначення самих методів.

Практичне завдання 1

Напишіть функцію, яка набирає масив цілих чисел і повертає середнє арифметичне значення елементів масиву.

Відповідь

```
1  buffer = Array.new
2
3  def average(val, buffer)
4    if val
5      buffer.append(val)
6    end
7
8    sum = 0.0
9
10   buffer.each do |num|
11     sum += num
12   end
13
14   return sum / buffer.length
15 end
16
17
18 average(1, buffer)
19 average(2, buffer)
20 average(15, buffer)
21
22 puts average(nil , buffer) # 6.0
```



```
mymyka@DESKTOP-JSFAPKL: /mnt/c/Users/Mykyta/home/Projects/ruby/test_1/practise_task_1$ ruby main.rb
6.0
8.25
mymyka@DESKTOP-JSFAPKL: /mnt/c/Users/Mykyta/home/Projects/ruby/test_1/practise_task_1$
```

Рисунок 1: Результат роботи програми

Практичне завдання 2

Створіть простий додаток "Нотатки", який дозволяє користувачам створювати, переглядати, редагувати та видаляти нотатки. Кожна нотатка повинна мати заголовок та текст.

Відповідь

1. Створюємо новий проект Ruby on Rails, виконавши наступну команду в терміналі:

```
rails new notes_app
```

2. Змінюємо директорію на нову дирекорію проекту:

```
cd notes_app
```

3. Створюємо каркас для моделі нотатки з атрибутами заголовка та тіла:

```
rails generate scaffold Note title:string body:text
```

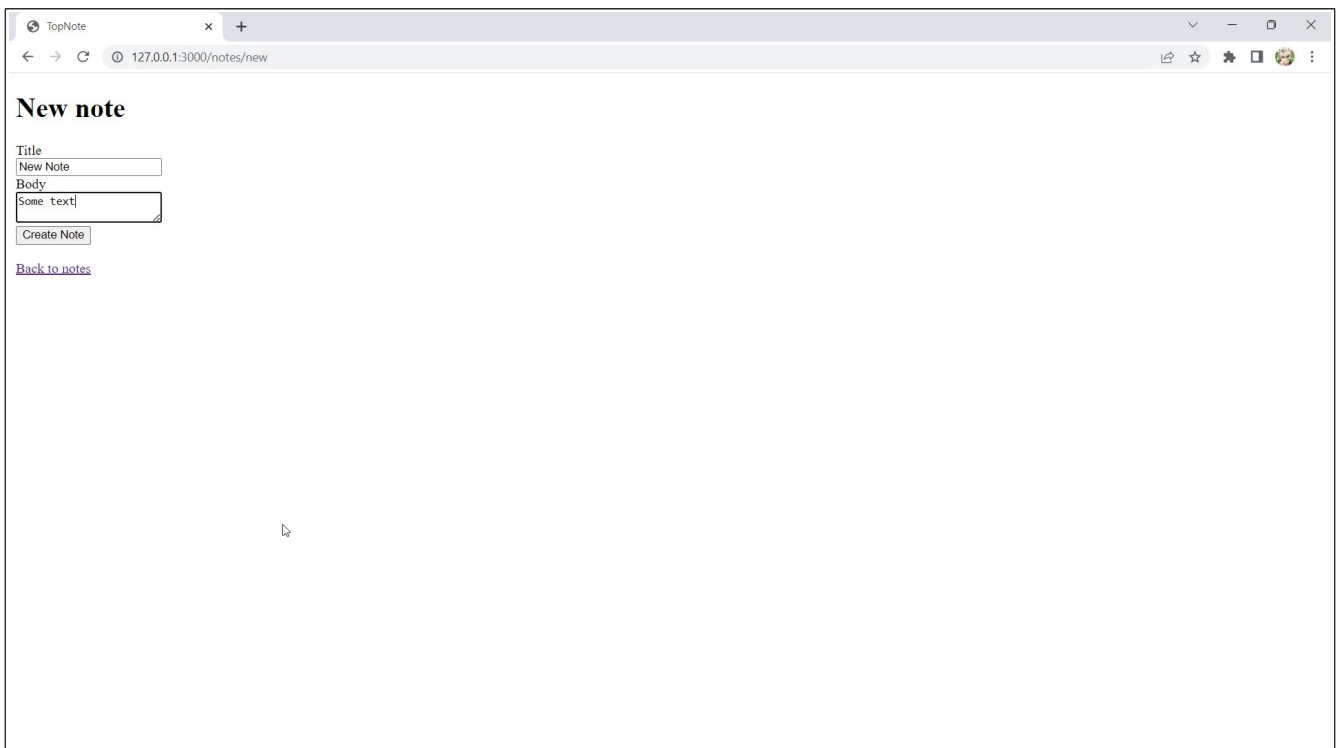
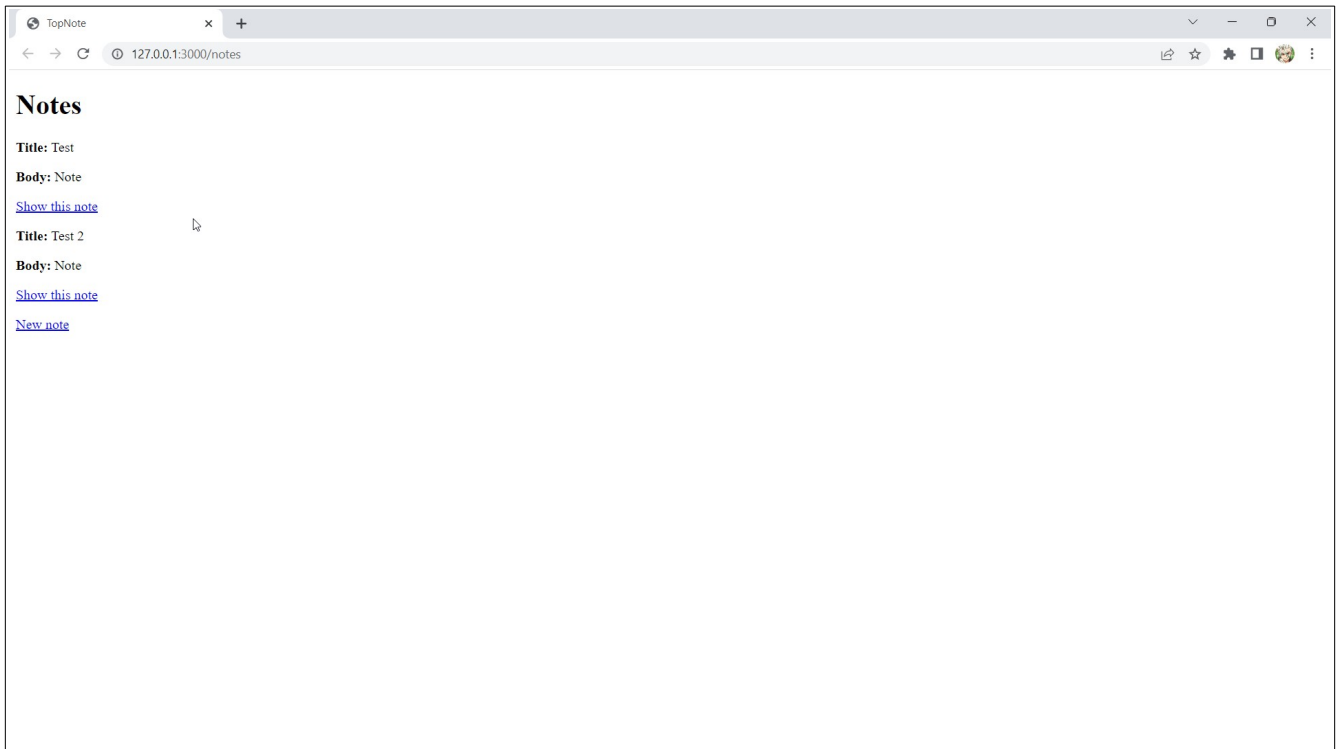
4. Запускаємо міграцію бази даних, щоб створити таблицю нотаток:

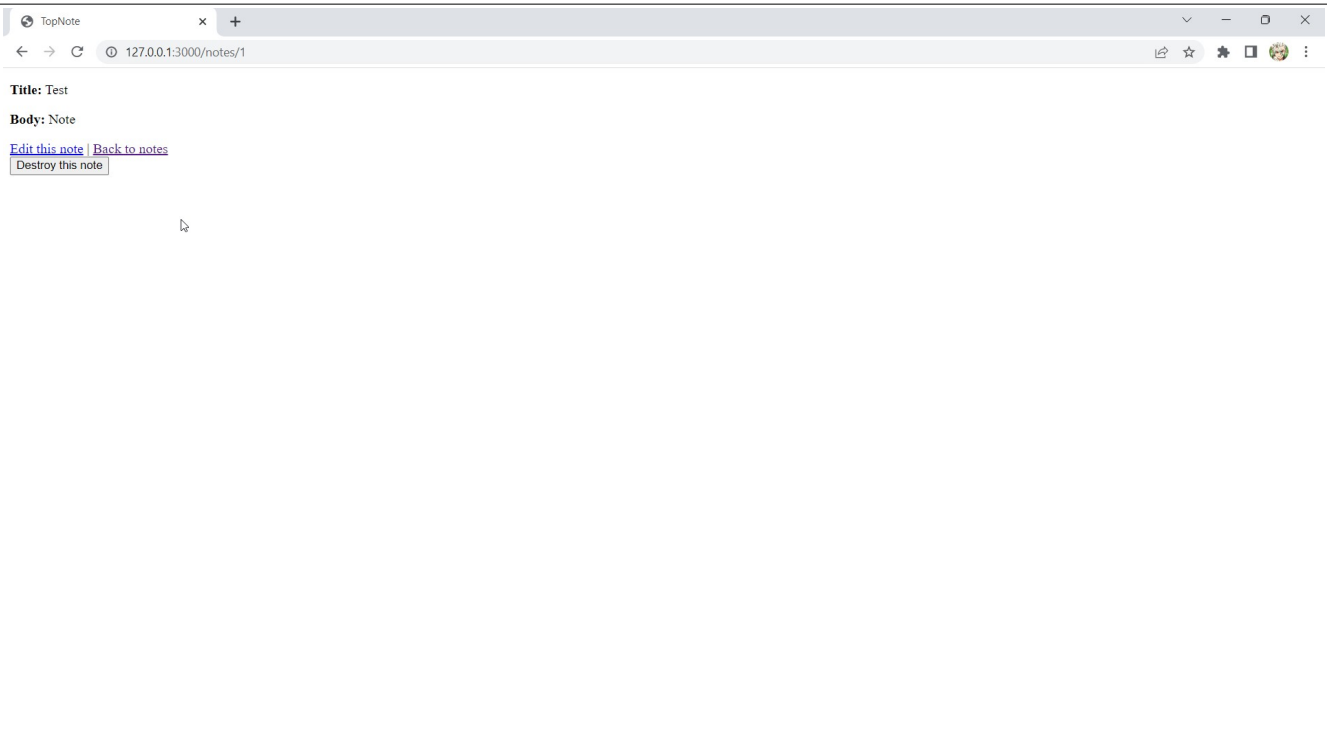
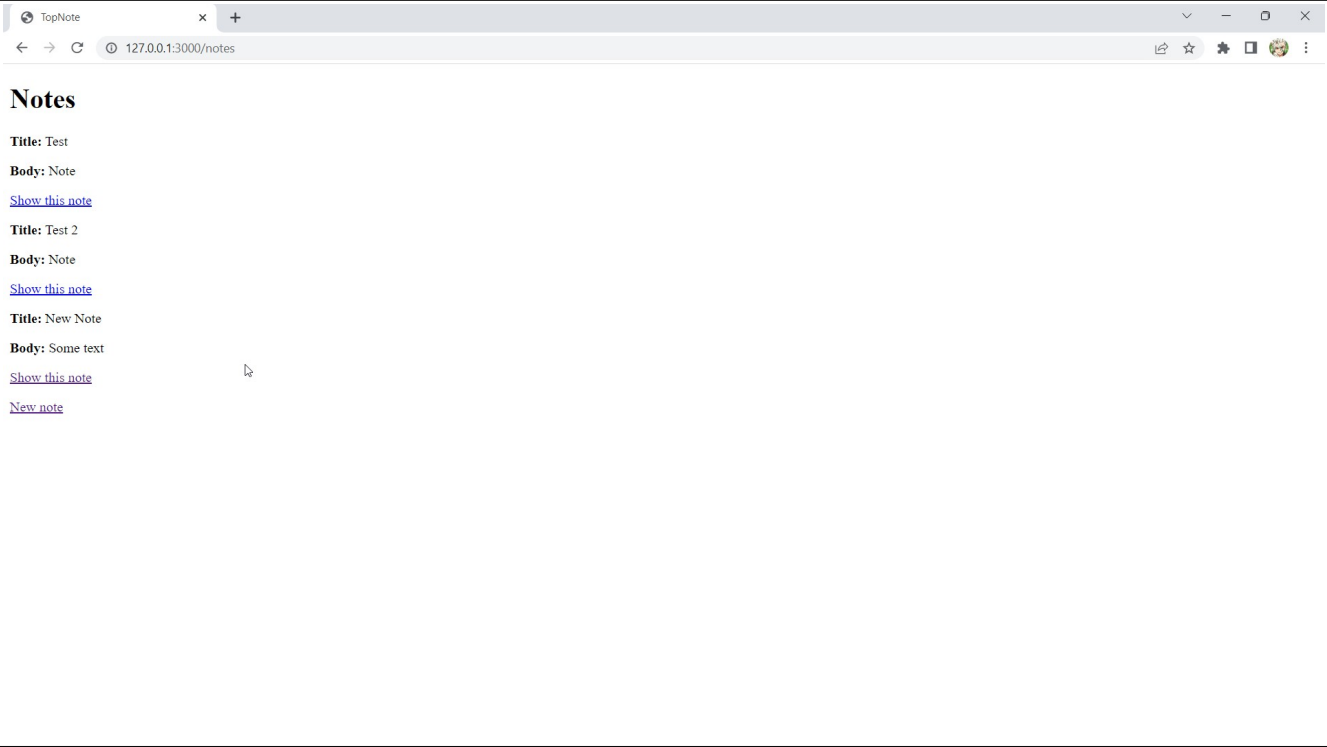
```
rails db:migrate
```

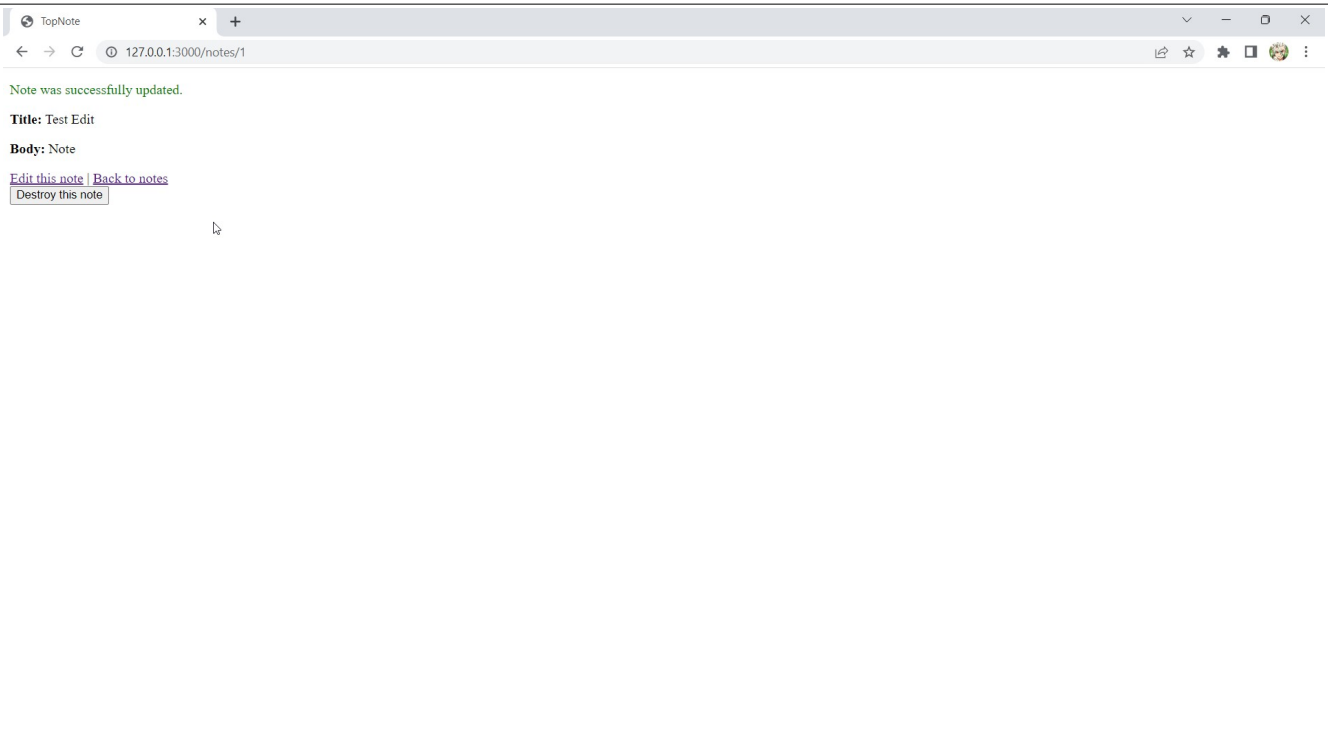
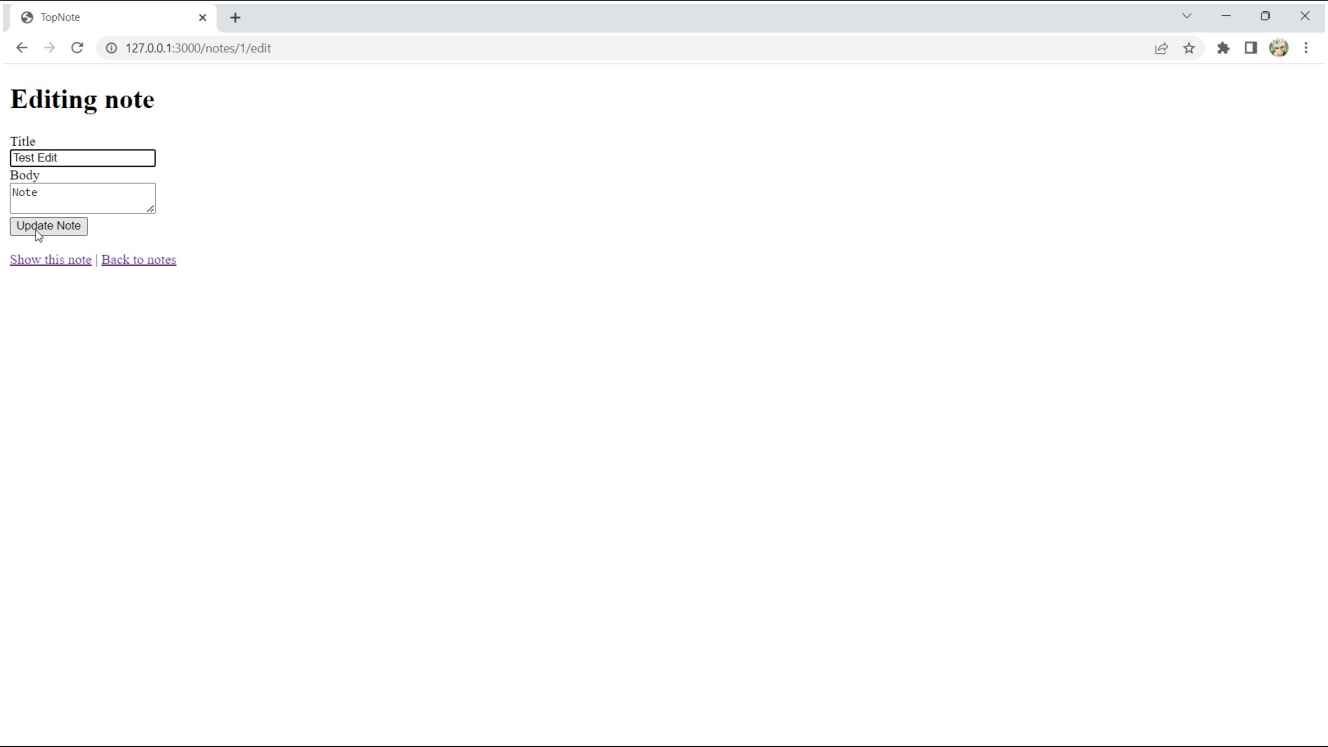
5. Запускаємо сервер:

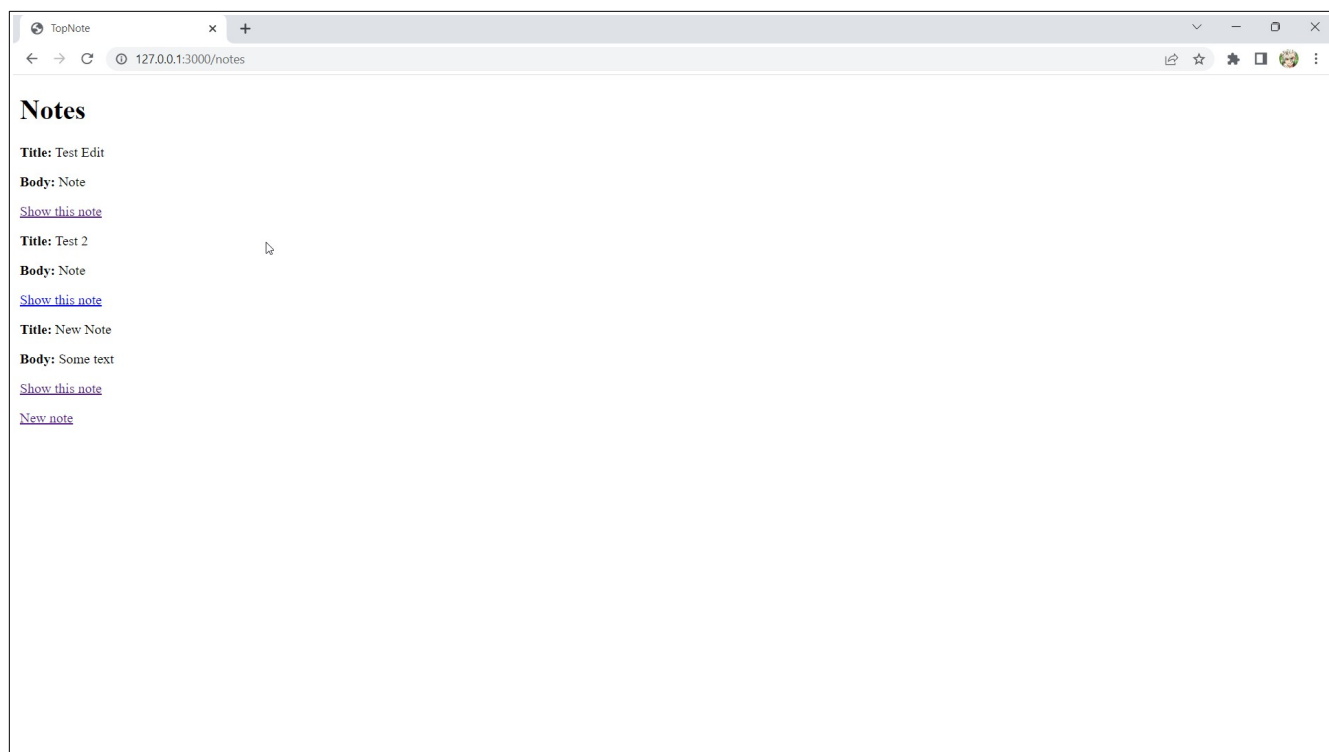
rails server

6. Відкриваємо веб-браузер і переходимо за адресою `http://localhost:3000/notes`, щоб побачити сторінку з нотатками.









Посилання на GitHub репозиторій з програмним кодом:
https://github.com/mymyka/ruby/tree/main/test_1/practise_task_2/top_note