# ЛАБОРАТОРНА РОБОТА 1.
# ТЕСТУВАННЯ КОДУ RAILS ДОДАТКУ

**Мета роботи:** ознайомитися з тестування інфраструктури коду. Одержати практичні навики з використання гемів для .

**Обладнання:**

– встановлене програмне середовище ruby + Rails;

– будь-який редактор програмного коду: JetBarins WebStorm, Visual Studio Code, Sublime Text, Atom;

– підключення до мережі Інтернет;

– підготовлений проект за допомогою фреймворку Rails.

## 1.1 Теоретичні відомості

Lintering – це процес автоматичної перевірки коду на відповідність певним стандартам кодування та виявлення потенційних проблем в коді. Linter – це інструмент аналізу коду, який автоматично перевіряє код на наявність помилок, стиль кодування та дотримання стандартів. Він аналізує код без запуску програми та надає повідомлення про помилки та порушення стандартів в коді. Найпоширеніші лінтери – це інструменти для перевірки коду на мовах програмування, таких як JavaScript, Python, Ruby, Java та інші. Їх можна використовувати окремо або вбудовувати в інтегровані середовища розробки (IDE), що дозволяє отримувати інформацію про порушення стандартів кодування під час розробки.

Лінтеринг допомагає забезпечити якість коду та зменшує кількість помилок, що можуть з'явитися в програмі. Також він полегшує розуміння та читабельність коду і поліпшує співпрацю між членами команди розробників.

Статичні аналізатори коду можна розділити на:

− аналізатори стилів коду: перевірка коду відповідно до найкращих практик, щоб реальні програмісти Ruby могли написати код, який може підтримуватися іншими реальними Ruby програмістами (Rubocop, reek);

− засоби сканування вразливостей: сканують веб-додатки з метою пошуку вразливих місць безпеки, таких як міжсайтовий скриптинг (cross-site scripting), впровадження SQL і впровадження команд (Brakeman, Bundle-audit);

– оптимізатори продуктивності коду: перевіряють швидкісні ідіоми, написані у коді, і дають поради щодо їх покращення (Bundle-leak, Fasterer).

Rubocop – це найпоширеніший інструмент для лінтерингу коду Ruby. Він дозволяє виконувати перевірки коду на дотримання стилю і кращих практик програмування, таких як змінні, методи, імена класів, розмір файлів, використання коментарів та інше.

**Використання Rubocop:**

1 встановіть Rubocop за допомогою команди:

```
gem install rubocop
```

2 додайте rubocop до файлу Gemfile проекту:

```
gem 'rubocop', require: false
```

3 для генерації файлу конфігурування Rubocop в директорії проекту виконайте наступну команду:

```
rubocop --auto-gen-config
```

Ця команда згенерує файл `.rubocop.yml` у кореневій директорії проекту з налаштуваннями Rubocop;

4 запустіть Rubocop командою:

```
rubocop
```

Rubocop застосує налаштування з файлу `.rubocop.yml` та надасть список порушень;

5 для внесення змін до налаштувань Rubocop відкрийте файл `.rubocop.yml` та внесіть зміни до параметрів, які потрібно змінити;

6 запустіть Rubocop ще раз, щоб перевірити, що зміни відповідно відображаються у налаштуваннях;

7 очікуйте, щоб Rubocop перевірив код проекту на відповідність стандартам і надав список порушень;

8 виправте порушення, що знайшов Rubocop. Для цього перегляньте описи порушень та рекомендації, які надає Rubocop, та внесіть зміни до коду;

9 запустіть Rubocop ще раз, щоб перевірити, чи були виправлені всі порушення;

10 повторюйте кроки 8 і 9 до тих пір, поки Rubocop не поверне жодного порушення.

Зверніть увагу, що Rubocop може знайти багато порушень, які не обов'язково повинні бути виправлені, але рекомендується дотримуватись стандартів, які він надає, щоб забезпечити кращу

читабельність і сумісність коду з іншими бібліотеками та проектами.

Для врахування особливостей розробки проектів можна використовувати розширення Rubocop, такі як rubocop-performance та rubocop-rails.

**Rubocop-performance** – це плагін для Rubocop, який надає додаткові правила для виявлення потенційних проблем з продуктивністю в Ruby-коді.

Даний плагін включає в себе правила, що допомагають виявляти надмірне використання пам'яті та процесорного часу в програмах на Ruby. Він перевіряє деякі типові шаблони, що можуть приводити до низької продуктивності програм, такі як використання циклів зайвого рівня вкладеності, зайвих операцій зі строками та багато іншого.

Наприклад, Rubocop-performance може виявити код, що виконує повільну операцію та запропонувати більш оптимальний спосіб її виконання. Він також може підказати, які методи класу Ruby найбільш продуктивні та їх слід використовувати.

**Rubocop-rails** – це плагін для Rubocop, який надає правила для виявлення потенційних проблем в коді додатків Ruby on Rails.

Даний плагін містить правила, які допомагають виявляти неправильне використання фреймворку Rails та порушення його конвенцій. Він допомагає підтримувати стиль кодування відповідно до Rails Best Practices та зменшує кількість можливих помилок, що можуть виникнути в додатках Rails.

Наприклад, Rubocop-rails може виявити використання застарілих методів у контролерах та запропонувати їх більш сучасні замінники. Він також може виявити неправильне використання ActiveRecord та запропонувати більш оптимальний спосіб взаємодії з базою даних.

Rails Best Practices – це інструмент, який дозволяє виконувати перевірки коду на дотримання кращих практик програмування в додатку Rails, таких як правильне використання моделей, контролерів, представлень, маршрутів і т. д.

Тестування захищеності веб-ресурсу – це один з видів тестування, що має на меті перевірку безпеки ресурсу, аналіз його вразливостей та ризиків. Загальні принципи безпеки базуються на принципах конфіденційності, цілісності та доступності. Перевіряють такі аспекти: контроль доступу, аутентифікацію,

валідацію вхідних значень, криптографічні алгоритми, механізми обробки помилок, конфігурацію сервера, інтеграцію із сторонніми сервісами, стійкість до Dos / DDos атак. Ступінь безпеки веб-додатку в роботі слід оцінити відповідно до міжнародного стандарту OWASP (Open Web Application Security Project). Інформацію щодо стандарту можна отримати за посиланням https://owasp.org/www-pdf-archive/ASVS_3_0_Ukrainian_Beta.pdf.

Тестування безпеки також передбачає пошук вразливостей веб-ресурсу, які бувають наступних видів: XSS (Cross-Site Scripting – виконання шкідливих скриптів); XSRF / CSRF (Request Forgery – використання недоліків HTTP протоколу); Code injections (SQL, PHP, ASP і т.д. – запуск зловмисником запуск виконуваного коду); Server-Side Includes (SSI) Injection (вставка серверних команд в HTML код або запуск їх безпосередньо з сервера); Authorization Bypass (можливість отримання несанкціонованого доступу до облікового запису користувача).

Пошук згаданих вразливостей можливо проводити також за допомогою автоматизованих сканерів захищеності. Вони допомагають виявити вразливості: етапу кодування; етапу впровадження та конфігурації; етапу експлуатації веб-сайту.

**Fasterer** – це гем для Ruby, який допомагає виявляти і рекомендувати виправлення швидкісних проблем в коді Ruby. Fasterer використовує статичний аналіз коду, щоб знайти часті та менш ефективні конструкції, які можна було б замінити на більш оптимальні рішення. Fasterer також може перевіряти, як код впливає на швидкість виконання всього проекту, що допомагає зосередитися на найбільш важливих оптимізаціях та забезпечити більш ефективну роботу проекту в цілому.

Встановлення гему Fasterer:

```
gem install fasterer
```

Запуск аналізу:

```
fasterer
```

### 1.2 Порядок виконання роботи

1 В проекті Rails провести статичне тестування коду з використанням Rubocop та його розширень rubocop-performance та rubocop-rails.

2 Навести скріни виявлених порушень правил оформлення коду та виправити їх.

3 Додати до розробленого додатку нову функціональність – локалізацію (можливість використання української та англійської мов інтерфейсу):

3.1 додати можливість переключення мов;

3.2 провести тестування коду з використанням rubocop та усунути всі виявлені порушення.

4 Встановити гем Fasterer та з його допомогою провести перевірку та виправлення швидкісних проблем в коді додатку.

5 Оформити звіт.

**1.3 Зміст звіту**

Звіт має містити:

- титульний аркуш;
- мету роботи і завдання;
- покроковий опис роботи, програмний код сценаріїв, копії екранів з результатами виконаної роботи.
- висновки.

**Запитання для самоконтролю:**

1 Що таке лінтеринг?

2 На які групи за функціональністю можна розділити статичні аналізатори коду?

3 Для чого використовувати Rubocop?

4 Які розширення Rubocop Ви знаєте. Яке їхнє призначення?

5 Для чого призначений гем Fasterer?

**Рекомендована література**

1 Linting and Auto-formatting Ruby Code With RuboCop - [Електронний ресурс]. – Режим доступу: https://www.honeybadger.io/blog/linting-formatting-ruby/

2 Офіційна документація по платформі Rails [Електронний ресурс]. – Режим доступу: https://guides.rubyonrails.org/v7.0/getting_started.html

**Тривалість заняття: 2** год.