**Приватний вищий навчальний заклад**

**"МІЖНАРОДНИЙ НАУКОВО-ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ АКАДЕМІКА Ю. БУГАЯ"**

**Факультет комп’ютерних наук та нафтогазової інженерії**

**Кафедра комп’ютерних наук і інформаційних систем**

**Пояснювальна записка**

до дипломної роботи

на здобуття освітньо-кваліфікаційного

рівня "бакалавр"

**на тему:**

**«Розробка програмного засобу для тестування веб-сайтів на доступність»**

Виконав: студент групи ЗПІ-21

напряму підготовки

6.050103 «Програмна інженерія»

Коритцев О.Ю.

Керівник: доцент Коротун Т.М. Рецензент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КИЇВ - 2017 рік**Приватний вищий навчальний заклад**

«МІЖНАРОДНИЙ НАУКОВО-ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ АКАДЕМІКА Ю. БУГАЯ»

**Напрям підготовки 6.050103 «Програмна інженерія»**

**ЗАТВЕРДЖУЮ**

Зав.кафедрою КНІС

\_\_\_\_\_\_\_\_\_\_\_ доцент Т.М. Коротун

“\_\_\_ “ \_\_\_\_\_\_\_\_\_\_\_2016 р.

**ЗАВДАННЯ**

**НА РОЗРОБКУ ВИПУСКНОЇ РОБОТИ**

**(освітньо-кваліфікаційний рівень — бакалавр)**

**Коритцев Олександр Юрійович**

1. Тема роботи: **“Розробка програмного засобу для тестування веб-сайтів на доступність”**

2. Керівник роботи: Коротун Тетяна Михайлівна, канд.фіз.-мат.наук, доцент кафедри комп’ютерних наук та інформаційних систем, затверджені наказом по Університету від "XX" xxxxxxxxxxxxxxxxxxxx.

3. Строк здачі студентом закінченої роботи: 27 березня 2017 р.

4. Перелік питань, що підлягають розробці:

* Загальні питання доступності веб ресурсів;
* Огляд існуючих засобів тестування доступності;
* Опис розробленого програмного засобу автоматизації тестування доступності.

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Назва етапів випускної роботи | Строк виконання етапів роботи | Примітка |
| 1. | Розробка розділу 1. Огляд існуючих засобів тестування доступності |  |  |
| 2. | Розробка розділу 2. Проектування і вибір рішень для створення автоматизованого застосунку тестування доступності. |  |  |
| 3. | Розробка розділу 3. Реалізація програмного засобу |  |  |
| 4. | Оформлення випускної роботи | 27 березня 2017 |  |

**Студент-випускник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.Ю. Коритцев**

**Науковий керівник\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т. М. Коротун**

Оглавление

[РЕФЕРАТ 6](#_Toc477978484)

[ВСТУП. 7](#_Toc477978485)

[РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ ЗАСОБІВ ДЛЯ ТЕСТУВАННЯ ДОСТУПНОСТІ. 11](#_Toc477978486)

[1.1.Загальні вимоги доступності. 11](#_Toc477978487)

[1.1.1.WCAG 11](#_Toc477978488)

[1.1.2.Сприйнятливість 11](#_Toc477978489)

[1.1.3.Операбельність 11](#_Toc477978490)

[1.1.4.Асистивність 12](#_Toc477978491)

[1.2.Приклади веб-доступності. 12](#_Toc477978492)

[1.3.Що таке тестування доступності. 14](#_Toc477978493)

[1.4.Як виконати тестування доступності 14](#_Toc477978494)

[1.5.Роль автоматизованих засобів в приймальних випробувань. 15](#_Toc477978495)

[1.6.Засоби тестування доступності. 15](#_Toc477978496)

[1.6.1. Achecker 15](#_Toc477978497)

[1.6.2 . Wave 16](#_Toc477978498)

[1.6.3. TAW 16](#_Toc477978499)

[1.6.4.Результати дослідження . 16](#_Toc477978500)

[Розділ 2. Проектування і вибір рішень для створення автоматизованого застосунку тестування доступності. 21](#_Toc477978501)

[2.1. Вибір мови програмування. 21](#_Toc477978502)

[2.1.1. Переваги інтерпретованості в поєднанні з високою продуктивністю. 21](#_Toc477978503)

[2.1.2. Розподіленість. 22](#_Toc477978504)

[2.1.3. Доступність інструментарію та ефективність розробок. 22](#_Toc477978505)

[2.1.4. Перспективи застосування. 23](#_Toc477978506)

[2.2. Вибір інструменту збірки проекту. 23](#_Toc477978507)

[2.2.1. Потреба в інструменті для збірки 23](#_Toc477978508)

[2.2.2. ANT. 23](#_Toc477978509)

[2.2.3. MAVEN. 24](#_Toc477978510)

[2.2.4.Історія Maven 25](#_Toc477978511)

[2.2.4. Життєвий цикл Maven проекту 25](#_Toc477978512)

[2.3. Вибір системи контролю версій. 27](#_Toc477978513)

[2.3.1. Централізовані системи контролю версій. 28](#_Toc477978514)

[2.3.2. Розподілені системи контролю версії. 28](#_Toc477978515)

[2.3.3.Збереження файлів 29](#_Toc477978516)

[2.3.4.Локальні операції 29](#_Toc477978517)

[2.3.5.Цілісність даних 30](#_Toc477978518)

[2.3.6.Гілки 30](#_Toc477978519)

[2.4. BDD. 31](#_Toc477978520)

[2.4.1. Керована поведінкою розробка 31](#_Toc477978521)

[2.4.2. jBehave. 31](#_Toc477978522)

РЕФЕРАТ

“Створення програмного засобу для тестування веб-сайтів на доступність”. Коритцев О.Ю. – Київ, МНТУ, Кафедра комп’ютерних наук та інформаційних систем, 2017 – xx с., xx рис., xx літературних джерел.

**Метою роботи** є розробка програмного засобу для автоматизованого тестування доступності веб ресурсів та інтеграцією з системою безперервної інтеграції Jenkins.

**Об’єктом дослідження** є засоби, що можуть використовуватися для тестування доступності.

**Задачами дослідження є:**

визначення існуючих засобів для тестування доступності;

визначення можливостей існуючих засобів для тестування доступності;

визначення способів використання існуючих засобів для тестування доступності;

розробка вимог до автоматизованого засобу тестування доступності веб ресурсів;

Розробка програмного застосунку, який проводить автоматизоване тестування веб ресурсів на доступність.

**Основними результатами роботи** є розроблений програмний засіб для автоматизованого тестування веб сайтів на доступність.

Ключові слова: ДОСТУПНІСТЬ, ACCESSABILITY, CONTINUOUS INTEGRATION Jenkins, СИСТЕМА КОНТРОЛЮ ВЕРСІЙ GIT, WCAG - Web Content Accessibility Guidelines, BDD .

ВСТУП.

На сьогоднішній день все частіши починають використовувати програмні засоби для автоматизованого тестування програмних систем.

Це зокрема тестування Фронт-Енду та тестування Бек-Енду.

Тестування Фронт-Енд частини е важливим тому ,що цей рівень програмної системи розроблений для спілкування с користувачами. І від того наскільки зручно спланований інтерфейс користувача залежить й успіх програмної системи в цілому.

Тестування Бек-Енд рівня проводиться для виявлення різного типу проблем і здебільшого пов’язано з нефункціональними вимогами. Такими як надійність, відновлювальність системи , вимоги до часу зберігання даних та інше.

Для тестування інтерфейсу користувача використовують різні види тестування .

Приклад:

тестування зручності використання (usability testing);

тестування інтерфейсу користувача (ui testing);

тестування локалізації (localization testing);

тестування доступності(accessibility testing)

**Мета Роботи** це дослідження методів тестування доступності веб ресурсів та проектування програмного засобу для тестування доступності.

Глобальна доступність на сьогоднішній день це поширення інформації якою зможуть користуватися люди з різними обмеженнями. Доступність систем це адаптація інтерфейсів для людей з обмеженими можливостями.

**Критерії доступності** , це сукупність правил з гайдлайнів (методики та вказівки для створення доступного веб ресурсу) доступності[1].

До них входять:

Правильне читання контенту веб – ресурсу Скрінрідером (програма яка зчитує текстову інформацію та передає її через аудіо канал) ;

Розмір шрифту та фон;

Розміщення alt описів зображень та гіперпосилань;

**Актуальність роботи***.* Тема доступності веб-ресурсів відносно нова для України. Тому розробка програмного застосунку для тестування доступності веб ресурсів з використанням мови JAVA та сучасного BDD[2] підходу для створення тестових сценаріїв є актуальною та має практичне значення. Також на сьогоднішній день немає повноцінного автоматизованого засобу для тестування доступності веб ресурсів.

**Об’єктом дослідження** є засоби, що можуть використовуватися для тестування доступності.

**Задачами дослідження є:**

визначення існуючих засобів для тестування доступності;

визначення можливостей існуючих засобів для тестування доступності;

визначення способів використання існуючих засобів для тестування доступності;

розробка вимог до автоматизованого засобу тестування доступності веб ресурсів;

Розробка програмного застосунку, який проводить автоматизоване тестування веб ресурсів на доступність.

**Основними результатами роботи** є розроблений програмний засіб для автоматизованого тестування веб сайтів на доступність.

Дипломна робота складається з 3 розділів і висновків.

У першому розділі будуть розглянуті поняття і концепції доступності веб ресурсів. Також буде провединий анализ існуючих засобів тестування доступності веб ресурсів, та буде обраній один для інтеграції з застосунком автоматичного тестування доступності.

У другому розділі проведений аналіз та вибір мови програмування,

системи контролю версій, системи збірки проекту та системи безперервної інтеграції. Також буде опис і аналіз BDD плагінів які увійдуть у білд проекту.

У третьому розділі описується і пояснюється технічна реалізація реалізація застосунку для автоматичного тестування доступності. Опис конфігурації а також розглянуті щляхи інтеграції з системами CONTINUOUS INTEGRATION.

# РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ ЗАСОБІВ ДЛЯ ТЕСТУВАННЯ ДОСТУПНОСТІ.

1.1.Загальні вимоги доступності.

1.1.1.WCAG - Web Content Accessibility Guidelines[3] або Рекомендації по доступності веб-контенту.

1.1.2.Сприйнятливість - Інформація і призначені для користувача компоненти інтерфейсу повинні бути презентабельними для користувачів в тому, як вони можуть сприйматися;

* Text Alternatives - Надання текстових альтернатив для будь-якого *не текстового вмісту*, так щоб буз зрозумілий людям. Люди потребують, наприклад, великий шрифт, шрифт Брайля, слова, символи або простою мовою;
* Time-based Media - Забезпечити альтернативи для почасових ЗМІ (аудіозапис, субтитри, сурдопереклад);
* Adaptable - Створення контенту, який може бути представлений різними способами (наприклад, простий макет) без втрати інформації або структури;
* Distinguishable - Зробити це простіше, щоб побачити і почути зміст, включаючи відділення переднього плану від фон;

1.1.3.Операбельність **-** Компоненти для користувача інтерфейсу і навігації повинні бути операбельні;

* Keyboard Accessible - Зробіть усі функціональні можливості, доступними з клавіатури.(навігація, табуляція);
* Enough Time - Надавання користувачам достатньо часу для читання і використовування контенту;
* Seizures - Не створюйте контент який може спонукати судоми;
* Navigable - Надати можливості користувачам орієнтуватися в контенті, зміст та можливисть виявити місце знаходження у ньому;
* Readable - контент має бути зрозумілим і доступним для читання;

1.1.4.Асистивність **-** Контент повинен бути досить асистивним, бо він може бути інтерпретований широким спектром користувальницьких агентів, включаючи допоміжні технології.

* Compatible- Забезпечення максимальної сумісності з існуючими і майбутніми призначеними для користувача агентами, включаючи допоміжні технології;

Веб принципово призначений для роботи всіх людей, незалежно від їх апаратних засобів, програмного забезпечення, мови, культури, місце або фізичних або розумових здібностей. Коли веб відповідає цій меті, він доступний для людей з різноманітним діапазоном слуху, руху, зору і когнітивними здібностями.

## 1.2.Приклади веб-доступності.

Правильно спроектовані веб-сайтів і веб-ресурси можуть бути використані людьми з обмеженими можливостями. Тим не менш, у даний час багато сайтів і інструменти розробляються з обмеженим доступом, які роблять його важким або неможливим для деяких людей, щоб використовувати їх. Нижче наведені лише кілька прикладів.



Рис. 1. Альтернативний текст для зображень.

Alt текст є класичним прикладом Рис.1. Зображення повинні включати в себе еквівалентний альтернативний текст в розмітці / код.

Якщо альтернативний текст не передбачений для зображень, інформація про зображення недоступна, наприклад, для людей, які не можуть бачити і використовують скрінрідери для читання з екрану, який читає вголос інформацію на сторінці, включаючи альтернативний текст для візуального образу.

Коли еквівалентний альтернативний текст надається, інформація доступна всім людям, які є сліпими, а також для людей, які виключають зображення на своєму мобільному телефоні для зниження витрат енергії.

Подібно до того, як зображення не доступні для людей, які не можуть бачити, звукові файли не доступні для людей, які не можуть чути. Надання текстової інформації “транскрипт” робить звукову інформацію доступною для людей, які є глухими , а також для пошукових систем і інших технологій, які не можуть розрізняти аудіо ряд.

Це легко і відносно недорого для розробників веб-сайтів, щоб забезпечити стенограми для подкастів і аудіофайлів. Є також послуги транскрипції, які створюють текстові стенограми в форматі HTML.

Як: Зробити свій веб-сайт і веб-інструмент Доступним.

Більшість основ доступності ще простіше і дешевше, ніж надання стенограми. Проте, відповідні методи слабо інтегровані в деякі веб-інструменти, освіту і процеси розвитку. Якщо ви новачок в доступності, це займає деякий час і зусилля, щоб дізнатися загальні проблеми і рішення. Початкове місце є Introduction to Web Accessibility.[]

Деякі бар'єри доступності є більш складними потрібно більше часу розробки і зусиль. W3C WAI надає великі ресурси, щоб допомогти, такі як розуміння **WCAG** 2.0: Керівництво по розумінню і здійсненню Web Content Accessibility Guidelines 2.0.

Використання коштів розробки, які підтримують доступність робить його більш легким для розробників веб-сайтів. Браузери також відіграють певну роль в доступності. Основні компоненти веб-доступності пояснюють взаємозв'язок між різними компонентами веб-розробки і взаємодії.

## 1.3.Що таке тестування доступності.

Тестування доступності є підмножиною юзабіліті-тестування, де в користувачів розглянутих людей з усіма можливостями та інвалідів. Значення цього тестування є перевірка як зручність використання і доступність.

Доступність прагне задовольнити людей різних здібностей, таких як:

* візуальними погіршенням
* фізичне погіршення
* порушення слуху
* Когнітивні порушення
* вивчення знецінення

Хороший веб-додаток повинен задовольнити всіх множин людей і не обмежується тільки людям з обмеженими фізичними можливостями. До них відносяться:

* Користувачі з поганою інфраструктурою зв'язку
* Люди похилого віку і нові користувачі, які часто неписьменним комп'ютер
* Користувачі, що використовують стару систему (НЕ може працювати останню версію програмного забезпечення)
* Користувачі, які використовують Нестандартне обладнання
* Користувачі, у яких є доступ обмежений

## 1.4.Як виконати тестування доступності

Ініціатива Web Accessibility (WAI) описує стратегію попередніх і атестаційних оглядів веб-сайтів. Ініціатива Web Accessibility (WAI) включає в себе перелік програмних засобів для надання допомоги в оцінці відповідності. Ці інструменти варіюються від конкретних питань, таких як колірної сліпоти до інструментів, які будуть виконувати автоматизовані засоби індексацію.

Приклади WAVE, TAW, Achecker.

## 1.5.Роль автоматизованих засобів в приймальних випробувань.

Усі вказані вище засоби тестування доступності дуже хороші при визначенні сторінок і рядків коду, які необхідно вручну перевірити на доступність.

Перевірити синтаксис коду сайту;

Пошук відомих шаблонів, які можуть;

ідентифікувати сторінки, що містять елементи, які можуть викликати проблеми;

Визначити деякі актуальні проблеми доступності;

Виявити деякі потенційні проблеми;

Інтерпретація результатів від автоматизованих засобів тестування доступності вимагає досвіду в методах доступності з розумінням технічних і юзабіліті питань.

## 1.6.Засоби тестування доступності.

Автоматичні перевірки доступності можуть з упевненістю ідентифікувати обмежену кількість бар'єрів в веб-контенті. Людина повинна взаємодіяти з оглядом доступності, щоб приймати рішення про потенційні проблеми, які не можуть визначити автоматизовані інструменти. Наприклад, будь-яка перевірка, яка вимагає оцінки сенсу, наприклад, точно чи текст посилання описує мету посилання, або в достатній мірі описує те що міститься в зображенні, людина повинна прийняти рішення.

### 1.6.1. Achecker

Засіб перевірки доступності - це інструмент оцінки доступності Інтернету. Ввівши URL-адресу веб-сторінки або завантаживши його веб-доступ до HTML-файлу, ви можете ознайомитися. Це безкоштовний інструмент, який дозволяє вибирати формат звіту. AChecker дозволяє користувачеві створювати рекомендації щодо забезпечення доступності.

### 1.6.2 . **Wave**

Wave - це безкоштовний інструмент для веб-доступу, створений WEBAIM. Він використовується для перевірки веб-сторінки вручну для різних аспектів доступності. Цей інструмент може бути використаний для перевірки внутрішньої, захищеної паролем, динамічно згенерованої або чутливої ​​веб-сторінки. Основні функції панелі Web Accessibility Toolbar включають в себе ідентифікацію компонентів веб-сторінки, надання доступу до альтернативного перегляду вмісту сторінки і полегшення використання сторонніх онлайн-додатків. Він забезпечує 100% конфіденційну і безпечну звітність про доступність.

### 1.6.3. **TAW**

TAW - це онлайн-інструмент для визначення доступності вашої мережі. Цей інструмент аналізує веб-сайт відповідно до основних принципів доступності W3C в Інтернеті і показує проблеми доступності. Проблеми з тестуванням веб-доступу діляться на пріоритети 1, пріоритет 2 і пріоритет 3. Цікавою особливістю TAW є здатність генерувати підмножини WCAG 1.0 для тестування. В інструменті TAW ви можете або протестувати одну сторінку або декілька сторінок «павуком» на сайті. TAW також дозволяє нам визначати додаткові перевірки за допомогою діалогового вікна «Перевірка користувача».

### 1.6.4.Результати дослідження .

Для тестування беремо сайт верховної ради України.

<http://rada.gov.ua/>

Державна влада також почали зважати на людей з вадами зору, слуху.

«Державний комітет телебачення і радіомовлення України та Міністерство регіонального розвитку, будівництва та житлово-комунального господарства України (Мінрегіон) прийняли спільний наказ «Про затвердження Змін до Порядку функціонування веб-сайтів органів виконавчої влади», зареєстрований в Міністерстві юстиції України за № 240/26685.»

«Наказом запроваджено низку вимог щодо забезпечення доступності інформації на офіційних веб-сайтах органів виконавчої влади для користувачів з вадами зору та слуху. Зокрема, відтепер система керування інформаційним вмістом веб-сайту органу виконавчої влади повинна мати універсальну модульну архітектуру та включати, крім інших,  модуль адаптації інформації для користувачів з вадами зору та слуху.

Ще одна із вимог - інформація на веб-сайті повинна залишатися доступною при відключеному дизайні офіційного веб-сайту, збільшеному розмірі шрифтів та на монохромному екрані. Ці та інші зміни слід запровадити на всіх веб-сайтах органів виконавчої влади.»

Кожен програмний засіб проводить тестування зсилаючись на основні правила та поради викладені у WCAG. Результати тестування доступності

Репорти можна побачити на Рис. 2 –Рис.5.

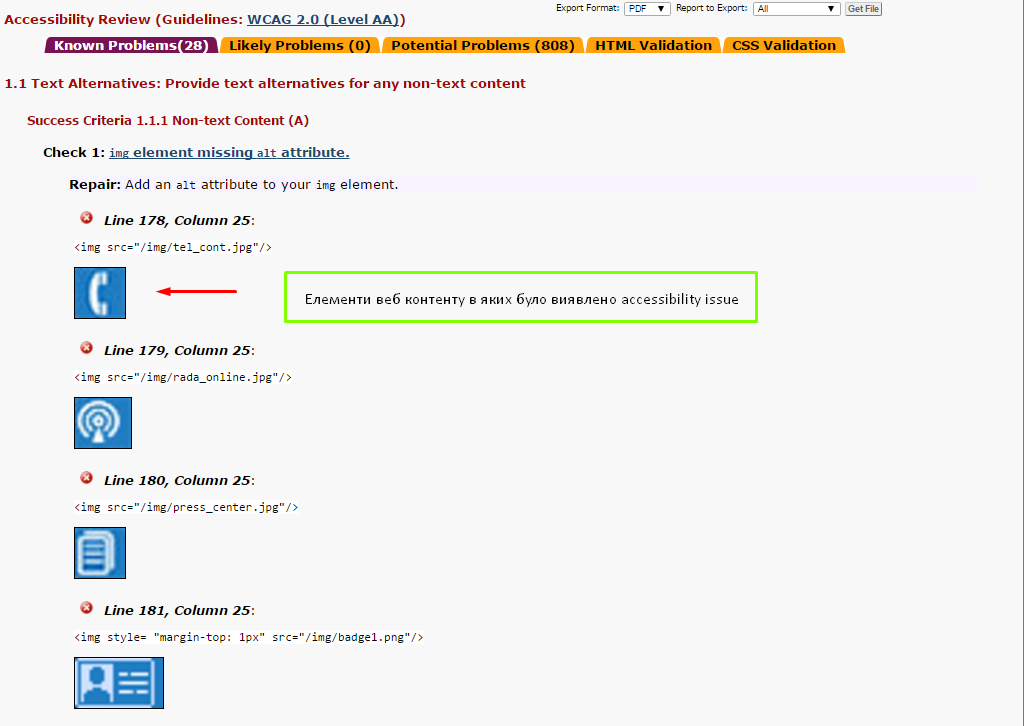


Рис. 2. Achecker репорт.

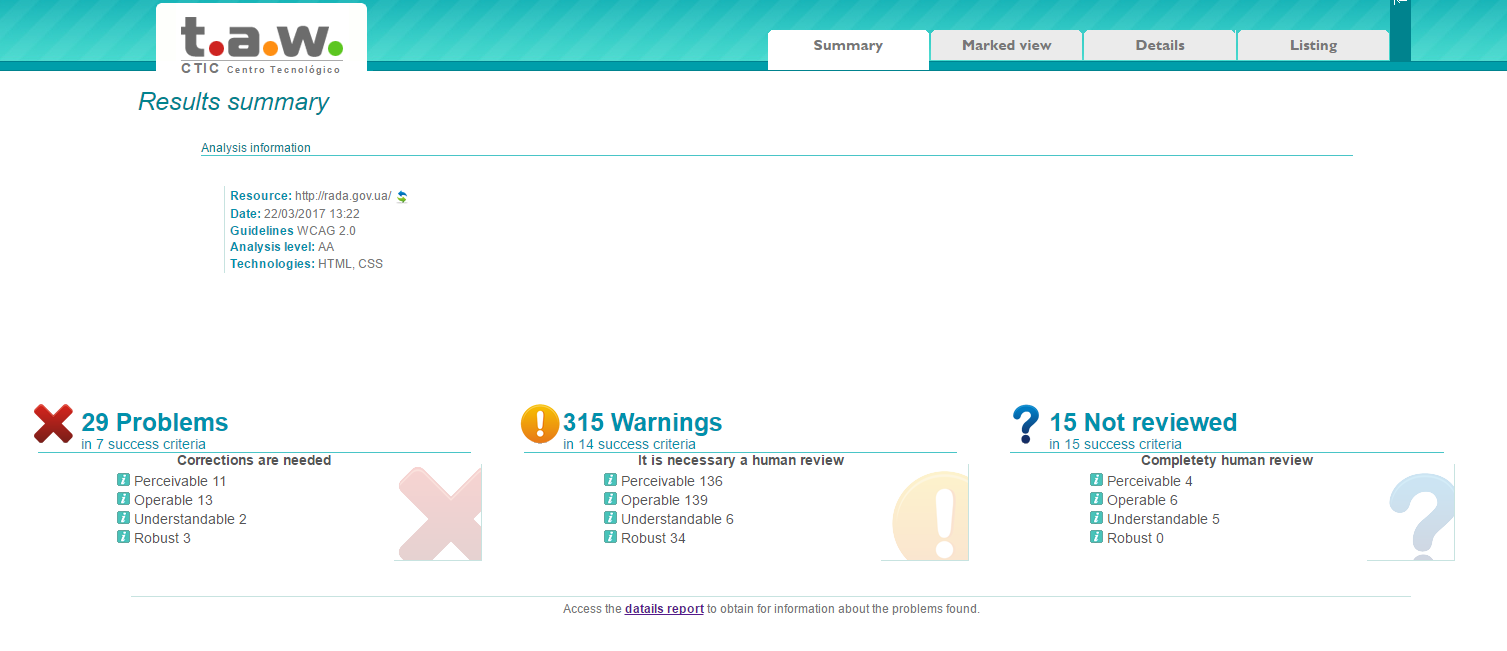


Рис.3. TAW репорт.

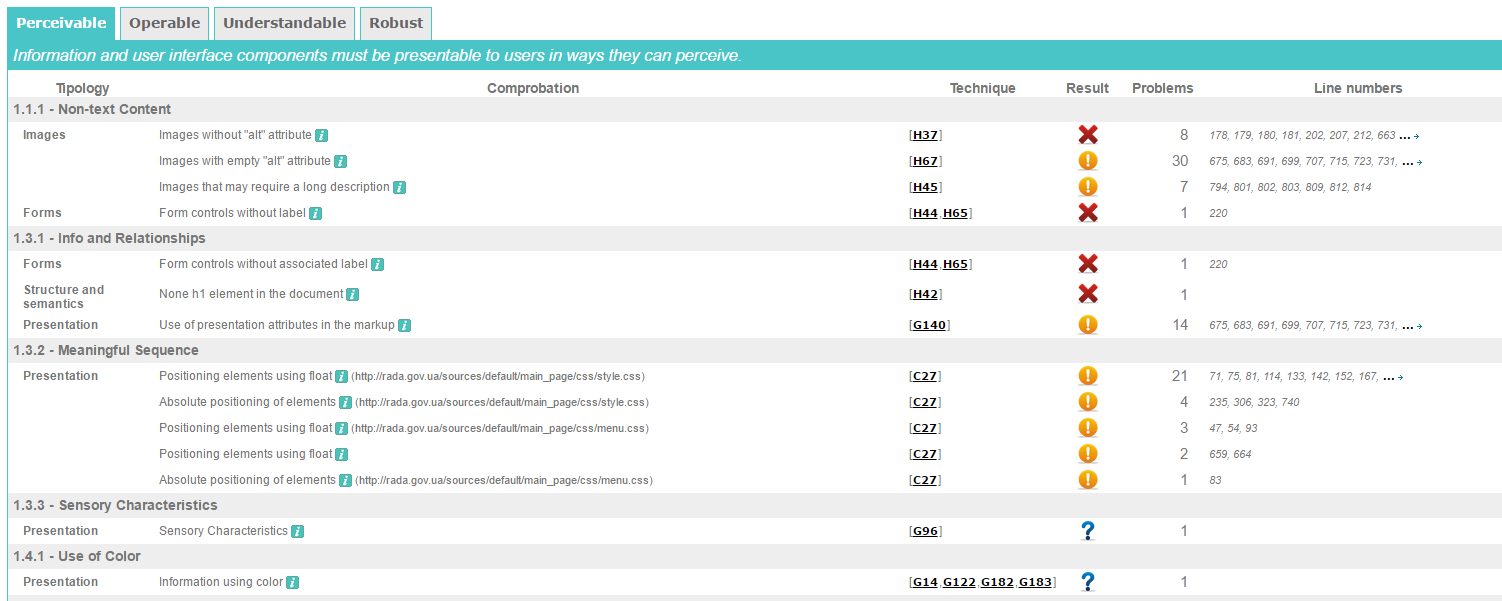


Рис.4. TAW розгорнутий детальній репорт.

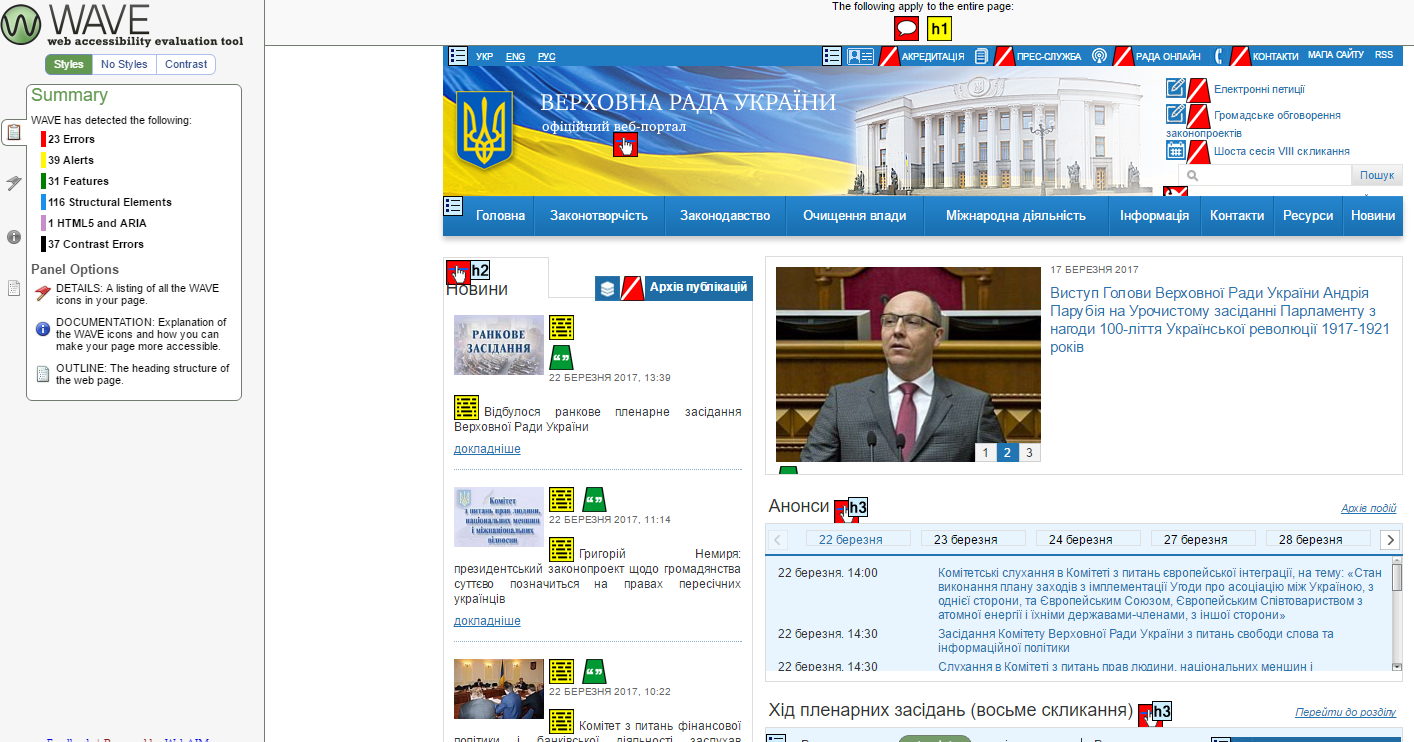


Рис.5. WAVE репорт.

* TAW, Achecker та WAVE знаходять однакову кількість помилок;
* TAW, Achecker працюють як незалежні веб застосунки.

Для тестування веб ресурсу потрібно переходити на їх сайт. Вони надають демо доступ та можливість протестувати сайт ввевши адресу цільового сайту у спеціальне поле це можна побачити на Рис.6 та рис.7



Рис.6. TAW.

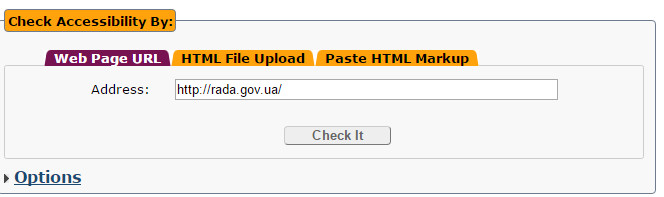


Рис.7. Achecker.

WAVE на окрему від попередніх працює як розширення браузера. Також він надае більш повну и наглядну інформацію про помилки.

Зважаючи на це використання WAVE буде більш доцільне в автоматизованому засобі для тестування доступності. Завдяки тому ,що WAVE розширює розмітку цільового сайту, це надає можловість робити скріншоти місць які мають помилки доступності. Це можно буде використати при формуванні репорту після проходження тесту. Репор якій містить зображення з помилками душе швидку спрямує девелоперів на вирішення питання.

# Розділ 2. Проектування і вибір рішень для створення автоматизованого застосунку тестування доступності.

## 2.1. Вибір мови програмування.

Одна з основних вимог мова має бути кроплотформенною. Для того щоб засіб для тестування доступності моно було використовувати на багатьох платформах таких як WINDOWS, UNIX, LINUX…

Крос-платформність – властивість програмного забезпечення працювати більш ніж на одній програмній (в тому числі – операційній системі) або апаратній платформи, та технології, що дозволяють досягти такої властивості. Кросплатформеність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення.

Java і C# — крос-платформні мови на рівні виконання, тобто їх виконавчі файли можна запускати на різних платформах без попередньої перекомпіляції. PHP, ActionScript, Perl, Python, Tcl і Ruby — крос-платформні інтерпретовані мови, їх інтерпретатори існують для багатьох платформ.

Та з усіх мов виділяеться JAVA. JAVA одна з найперших мов яка була розроблена для кросплатформеності. І один із слоганів JAVA це «Написане раз – працюе усюди». В основному це завдяки тому що java-code компілюеться у байт код і після цього його можно запустити на будь якій машині на якій встановлено JVM – Java Virtual Machine.

### 2.1.1. Переваги інтерпретованості в поєднанні з високою продуктивністю.

Як вже згадувалось, Java дозвляє створювати незалежні від платформи програми шляхом компіляції в проміжне представлення, яке називається байткодом. Багато попередніх спроб зпроб знати розв’язок проблеми незалежності від платформи були зроблені за рахунок продуктивності. Інтерпретуючі системи, подібні до BASIC, Perl, страждають на майже неподоланний дефіцит продуктивності. Це було враховано при створенні Java Незважаючи на те, що Java є інтерпретованою мовою, генерація байткодів була ретельно оптимізована в такий спосіб, щоб одержуваний байткод можна було легко перекладати в машинний код, який працює з дуже високою продуктивністю. Виконуючі системи такого роду не втрачають жодних переваг переносимого коду.

### 2.1.2. Розподіленість.

Мова Java призначена для створення програм, які працюють в розподіленому середовищі Internet на базі протоколів TCP/IP. Насправді доступ до ресурсів за допомогою URL відрізняється від доступу к файлу. Крім того в Java наявний засіб передачі повідомлень в межах внутрішнього адресного простору. Це дозволяє забеспечити віддалене виконання процедур. Ці інтерфкйси включені у пакет RMI (remote metod invocation). Цей засіб привносить високий рівень абстракції в програмування дл я середовища клієнт/сервер.

Java-програми несуть у собі значний обсяг інформації про типи часу виконання (run-time type information), яка використовується для дозволу доступу до об’єктів під час роботи програми. Це дозволяє забезпечити безпечну та оптимальну динамічну компоновку. В такий спосіб досягається захищеність середовища виконання аплетів.

### 2.1.3. Доступність інструментарію та ефективність розробок.

Зазначена вище простота програмування на Java є причиною того, що розробки на Java коштуватимуть дешевше аналогічних на більш потужніх мовах програмування. Цьому ж сприяє і переносимість програм на Java, оскільки ліквідуються витрати пов’язані з адаптацією програми на конкретній платформі. До того ж інтегровані програми-оболонки для розробки Java програм коштують набагато дешевше ( 70-100$ ) ніж аналогічні продукти C++, Delphi ( ~1000$). А набір інструментарію для пакетної компіляції Java програм JDK (Java Development Kit) є взагалі freeware. Тому платформу Java можна рекомендувати як ідеальну для створення некомерційних програмних продуктів, зокрема для галузі освіти.

### 2.1.4. Перспективи застосування.

Програми на Java можуть знайти різне застосування в навчальному процесі: інтерактивні навчаючі програми (HTML в поєднанні з Java), програми-тести і особливо ділові ігри. Додаткові переваги можна отримати, якщо пистати ці програми у вигляді аплетів, які ініціалізуються з Web сервера внутрішньої мережі Intranet. В такий спосіб можна уникнути інсталяції програми на багатьох комп’ютерах - користувач просто запускає Web-броузер і загружає потрібну сторінку.

## 2.2. Вибір інструменту збірки проекту.

### 2.2.1. Потреба в інструменті для збірки

В середньому, розробник витрачає значну кількість часу на виконання рутинних завдань, таких як складання і розгортання, які включають:

* компіляція коду
* Упаковка бінарних файлів
* Розгортання довічних файлів на тестовому сервері
* тестування змін
* При копіюванні коду з одного місця в інше

Щоб автоматизувати і спростити перераховані вище завдання були створенні інструменти збірки проекту.

### 2.2.2. ANT.

ANT -це Java-інструмент збірки проектів різної складності.

Ant був створений Джеймсом Дунканом Девідсоном (оригінальний автор Tomcat). Спочатку він був використаний для створення Tomcat і був у складі дистрибутива Tomcat.

Ant народився з проблем і складнощів, пов'язаних з Apache.

Ant був призначений в якості самостійного проекту в Apache 2000 року.

Зараз ANT являеться одним із найпошириніших інструментів збірки складних проектів. Але його починають витісняти інші інструменти такі як MAVEN, GRADLE, SBT.

### 2.2.3. MAVEN.

Maven є інструментом управління проектами. Maven надає розробникам повну структуру збірки і керування над життєвими циклами. Команда розробників може автоматизувати створення інфраструктури проекту майже на льоту, так як Maven використовує стандартну структуру директорій і житєві цикли складання за замовчуванням.

У разі праці на середовищом декількох команд розробників, Maven можна налаштувати для зручної розподільної роботи за дуже короткий проміжок часу. Оскільки більшість установок проекту прості і багаторазові, Maven робить життя розробника легким при створенні звітів, перевірок, складання і тестування установок автоматизації.

Maven надає розробникам способи управління над:

* Збіркою;
* генераціею документації;
* звітності;
* керування залежностями;
* СДМ;
* Релізами;

Підводячи підсумок, Maven спрощує і стандартизує процес збірки проекту. Він обробляє компіляцію, поширення документації, взаємодії в команді і інші завдання без проблем. Maven збільшує можливість багаторазового використання і піклується про більшу частину завдань, пов'язаних зі збіркою проекту.

### 2.2.4.Історія Maven

Maven був спочатку розроблений для спрощення будівельних процесів в проекті Turbine Джакарта. Були кілька проектів, і кожен проект містив кілька різних файлів ANT збірки.

Група Apache потім розробили Maven , який може побудувати кілька проектів разом, публікувати проектну інформацію, розгорнути проекти, обмінюватися JARs по декількох проектах і допомогати у співпраці команд розробників.

### 2.2.4. Життєвий цикл Maven проекту

Це список названих фаз, що визначає порядок дій при його побудові. Життєвий цикл Maven містить три незалежних порядку виконання:

* clean
* - Життєвий цикл для очищення проекту. Містить наступні фази:
  1. pre-clean
  2. clean
  3. post-clean
* default
* - Основний життєвий цикл, що містить наступні фази:
* 1. validate - виконується перевірка, чи є структура проекту повна й правильна.
* 2. generate-sources
* 3. process-sources
* 4. generate-resources
* 5. process-resources
* 6. compile - компілюються вихідні данні (класи, ресурси ….).
* 7. process-test-sources
* 8. process-test-resources
* 9. test-compile
* 10. test - зібраний код тестується заздалегідь підготовленим набором тестів.
* 11. package - упаковка відкомпільованих класів та інших ресурсів. Наприклад, в JAR-файл.
* 12. integration-test - програмне забезпечення в цілому або його великі модулі піддаються інтеграційного тестування. Перевіряється взаємодія між складовими частинами програмного продукту.
* 13. install - установка програмного забезпечення в локальний Maven-репозиторій, щоб зробити його доступним для інших проектів поточного користувача.
* 14. deploy - стабільна версія програмного забезпечення поширюється на віддалений Maven-репозиторій, щоб зробити його доступним для інших користувачів.
* • site - життєвий цикл генерації проектної документації. Складається з фаз:
* 1. pre-site
* 2. site
* 3. post-site
* 4. site-deploy

Завдякі простоті використання MAVEN набув дуже великої популярності серед розробників. Приклад файлу конфігурації можна побачити на Рис.8. Для конфігурації файлу pom.xml використвоють ключові слова такі як (Properties, dependencies, build, plugins).

Усі необхідні бібліотеки можно підключити просто вказавши їх ім’я, версію та репозиторій у тому випадку якщо немае у цетральному. Далі MAVEN зкачає усі необхідні бібліотеки і розмістить їх у локальному репозиторії.

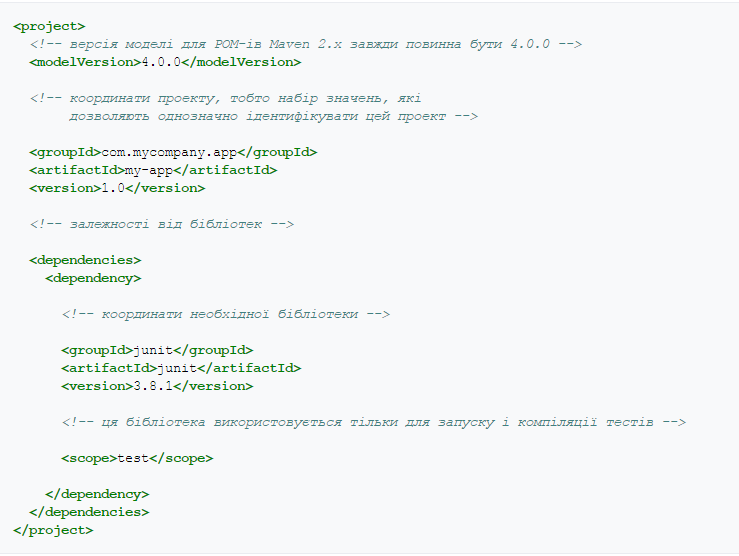


Рис.8. Зразок файлу конфігурації MAVEN (pom.xml)

## 2.3. Вибір системи контролю версій.

На сьогоднішній день є 2 найпошириніши системи контролю версій це GIT та SVN.

Система керування версіями — інструмент, який дозволяє одночасно, не заважаючи один одному, проводити роботу над груповими проектами.

Системи керування версіями зазвичай використовуються при розробці програмного забезпечення для відстеження, документування та контролю над поступовими змінами в електронних документах: у сирцевому коді застосунків, кресленнях, електронних моделях та інших документах, над змінами яких одночасно працюють декілька людей.

Кожна версія позначається унікальною цифрою чи літерою, зміни документу занотовуються. Зазвичай також зберігаються дані про автора зробленої зміни та її час.

Інструменти для контролю версій входять до складу багатьох інтегрованих середовищ розробки.

Існують два основні типи систем керування версіями: з централізованим сховищем та розподіленим.

### 2.3.1. Централізовані системи контролю версій.

Централізована система контролю версії (клієнт-серверна) — система, дані в якій зберігаються в єдиному «серверному» сховищі. Весь обмін файлами відбувається з використанням центрального сервера. Є можливість створення та роботи з локальними репозиторіями (робочими копіями).

Переваги:

* Загальна нумерація версій;
* Дані знаходяться на одному сервері;
* Можлива реалізація функції блокування файлів;
* Можливість керування доступом до файлів;

Недоліки:

* Потреба в мережевому з'єднанні для оновлення робочої копії чи збереження змін;

До таких систем відносять SVN, Team Foundation Server.

### 2.3.2. Розподілені системи контролю версії.

Розподілена система контролю версії (англ. Distributed Version Control System, DVCS) — система, яка використовує замість моделі клієнт-сервер, розподілену модель зберігання файлів. Така система не потребує сервера, адже всі файли знаходяться на кожному з комп'ютерів.

Переваги:

* Кожний з розробників працює зі своїм власним репозиторієм;
* Рішення щодо злиття гілок приймається керівником проекту;
* Немає потреби в мережевому з'єднанні;

Недоліки:

* Немає можливості контролю доступу до файлів;
* Відсутня загальна нумерація версії файла;
* Значно більша кількість необхідного дискового простору;
* Немає можливості блокування файлів;

До таких систем відносять Git, Mercurial, SVK, Monotone, Codeville, BitKeeper.

### 2.3.3.Збереження файлів

Git, на відміну від Subversion і подібних до неї систем, не зберігає інформацію як список змін (патчів) для файлів. Замість цього Git зберігає дані набором зліпків. Кожного разу при фіксації поточної версії проекту Git зберігає зліпок того, як виглядають всі файли проекту. Але якщо файл не змінювався, то дається посилання на раніше збережений файл (див. рис. 1). Git схожий на своєрідну файлову систему з інструментами, які працюють поверх неї. Для кожного відстежуваного файлу Git зберігає розмір, час створення і останньої зміни. Ці дані зберігаються у файлі index, який знаходиться у теці .git Вся база даних Git зберігається в теці з назвою .git

В Git файли можуть знаходитися в одному із 3-х станів: зафіксованому (файл вже збережено в локальній базі даних), зміненому (файл було змінено, але зміни не зафіксовано) і підготовленому (файли було змінено і відмічено для фіксації).

### 2.3.4.Локальні операції

Більшість дій можна виконувати на локальній файловій системі без використання інтернет підключення. Вся історія змін зберігається локально і при необхідності вивантажується у віддалений репозиторій. На відміну від Subversion, де без підключення до інтернету можна лише редагувати файли, але зберегти зміни в вашу базу даних неможливо (оскільки вона відключена від репозиторія). Будь-який коміт спочатку робиться локально, а потім вивантажується у віддалений репозиторій.

### 2.3.5.Цілісність даних

В своїй базі Git зберігає все по хешам файлів. Як хешуюча функція використовується SHA-1. Перед кожним збереженням файлів Git обчислює SHA-1 хеш файлу і отриманий хеш стає індексом файлу в Git. Використовуючи хеш Git легко відслідковує зміни в файлах.

### 2.3.6.Гілки

Гілки — це розмежування від основної лінії розробки. Git дозволяє створити декілька гілок і перемикатися між ними. Це корисно, оскільки дозволяє працювати декільком розробникам над своїм функціоналом не заважаючи іншим і не псуючи основу гілку. За замовчуванням, Git створює гілку з назвою master. Гілка в Git просто являє собою вказівник на одну із фіксацій. При кожній новій фіксації гілка в Git рухається автоматично (тобто перемикається на фіксацію). Гілка є простим файлом, який містить 40 символів контрольної суми SHA-1 фіксації. Створення нової гілки дуже швидке, оскільки це однаково запису в файл 41 байта (40 символів + символ нового рядка).

Зливання та перебазовування даних

Git підтримує два способи для інтеграції змін з гілки в гілку: merge (зливання) та rebase (перебазування). Основна різниця полягає в тому, що rebase запам'ятовує фіксації у вигляді патчів, перемотує гілку і застосовує патчі у вигляді фіксацій на відміну від merge, який зливає дві гілки в одну.

## 2.4. BDD.

### 2.4.1. Керована поведінкою розробка

behavior-driven development (BDD) — процес розробки програмного забезпечення, що виникла з керованої тестами розробки (TDD). BDD[5] поєднує основні засади та техніки TDD з ідеями предметно-орієнтованого проектування та об'єктно-орієнтованого дизайну з метою командам розробників та менеджменту спільні інструменти для співпраці під час розробки програмного забезпечення.

ерована поведінкою розробка розширення керованої тестами розробки[1], яка використовує прості предметно-орієнтовані мови програмування. Ці мови перетворюють запити природною мовою у виконувані тести. Результатом є більш тісний зв'язок з критеріями прийнятності для конкретної функції та тестами, які використовуються для перевірки цієї функціональності. Це є природним продовженням тестування TDD в цілому.

BDD фокусується на наступному:

* Коли розпочати процес
* Що тестувати та що не тестувати
* Скільки тестувати за один раз
* Як зрозуміти, чому тести пройшли неуспішно.

### 2.4.2. jBehave.

JBehave являє собою застосунок який уособлює BDD підхід.

Для тестування використовуються story файли які е переробленою версіею User-stories. А кроки в них уособлюють кроки з Use-case огорнутих Геркін стилем приклад можно побачити на рис.9.

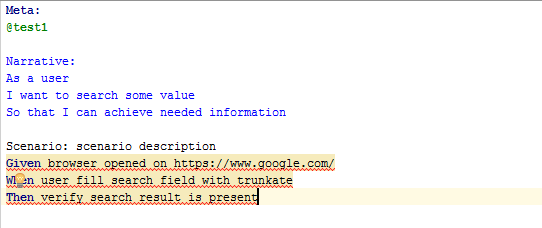


Рис.9. Приклад jBehave story file

Narrative: - описує User-story. Тобто дії юзера для здобуття якоїсь цілі.

Scenario: - описує Use-case. Конкретні дії, кожна з яких призводить до результату.

# РОЗДІЛ 3. Реалізація програмного засобу

## 3.1. Основні характеристики програмного засобу.

3.1.1. Характеристики програмного засобу.

|  |  |
| --- | --- |
| Мова програмування | Java |
| Система збірки | Maven |
| Система контролю версій | Git |
| BDD | jBehave |
| Accessibility застосунок | Wave |
| Reporting system | Serenity |

Для створення якісного продукту потрібно підібрати дизайн патерни які підійдуть як найкраще.

### 3.1.2. Дизайн Патерни.

Так уже прийнято, що усі дизайн патерни поділені на три великі групи, а

саме: породжуючі, структурні та поведінкові. Звичайно, що можна було б

опустити вступ до кожної із груп, але, насправді, поділ на групи має досить

велике значення. Не даремно хлопці із «банди чотирьох» вибрали саме 23

патерни а не більше і не менше, і недаремно вони поділили їх на ці групи. Це ж

була основна мета їхньої роботи – структуризувати та формалізувати вже

існуючі дизайн патерни.

### 3.1.3.Породжуючі патерни.

Отже, породжуючі патерни. Основним завданням таких патернів є спростити створення об’єктів, які необхідні аплікації. Інколи ви працюєте із певним набором об’єктів через групу інтерфейсів. А тоді хочете створювати об’єкти тільки із іншого набору, щоб пристосувати ваш код до інших умов. Звичайно група інтерфейсів, через які ви оперуєте, залишається та ж сама. Спростити створення відповідного набору допоможе Абстрактна Фабрика. А інколи структура деякого об’єкта дуже складна і залежить від багатьох чинників. Щоб спростити створення такого об’єкту зазвичай використовують Будівельника. А щоб зручно вибрати одну реалізацію та інстанціювати її, відштовхуючись від простої умови, можна використати Фабричний Метод. Нерідко постає завдання отримати копію уже існуючого об’єкта, або отримати можливість швидко генерувати багато подібних екземплярів. У такому випадку Прототип якраз згодиться. Вибагливе множення об’єктів не єдине завдання, яке вам слід буде виконувати у роботі, вам часто буде потрібно робити все точно навпаки – мати один-єдиний екземпляр об’єкта і ні за яких обставин не доступатися до йому подібних. Функціональність єдиного екземпляра забезпечуються Синглтоном.

### 3.1.4. Структурні патерни.

Основним завданням структурних патернів є формування найбільш підходящої структури та взаємодії між класами для виконання певних завдань. Якщо потрібно, щоб один об’єкт міг бути зрозумілим під іншим інтерфейсом, використовується Адаптер. Якщо ви хочете розділити абстракцію та імплементацію так, що на одному боці ви матимете абстракцію, а на іншому декілька реалізацій, причому всі доступні до модифікацій, то слід задуматися над поєднанням таких незалежних абстракції та реалізації за допомогою патерну Міст. Якщо елемент містить собі подібні елементи, а вони в свою чергу також можуть містити елементи, то найлегше таку структуру реалізувати за допомогою Компонувальника. Для швидкої та динамічної можливості розширення існуючої функціональності, без зміни її носителя, можна скористатися Декоратором. Якщо ваша система використовує багато об’єктів, що мають спільні дані, то такі дані можна винести та зробити загальнодоступними для економії пам’яті за допомогою патерну Легковаговик. Якщо відсутня можливість працювати із об’єктом напряму, використайте Проксі, що дозволить донести ваші команди до пункту призначення.

### 3.1.5. Патерни поведінки.

Ще однією групою патернів є такі, що акцентують свою увагу на поведінці. Вони або інкапсулюють поведінку, або дозволяють її розподілити. Щоб забезпечити почергову передачу роботи від одного класу до іншого і так дальше, аж до поки робота не буде виконана, використовують Ланцюжок Відповідальностей. Інколи краще запакувати інформацію про дії, які слід виконати, в один об’єкт Команди і переслати на опрацювання, або ж просто виконати в потрібному місці. Багато явищ можна описати за допомогою якоїсь спеціальної мови, наприклад погодні умови можуть бути записані значками, зрозумілими тільки метеорологам, але якщо вам подана граматика цієї мови і пояснення значків, цілком можливо, що ви зможете Інтерпретувати метеорологічне речення і зрозуміти його суть. Колекції об’єктів можуть бути «хитрими» і містити багато підколекцій та поокремих об’єктів. Щоб спростити життя користувачу такої колекції та щоб не викривати логіки колекції, придумали Ітератор, який допомагає легко і грамотно обійти усі об’єкти всередині. Спрощення коодинації роботи між деякою кількістю об’єктів може бути досягнуте виділенням посередника або медіатора. Медіатором може бути ваш бригадир на будівництві або ваш менеджер. Можливість повернутися до попереднього стану системи має велике значення. Така функціональність може бути досягнута Хранителем. Зверху завжди видніше, що коїться внизу. Спостерігач допоможе централізувати огляд роботи декількох класів та генерувати відповідні події. Стан системи та умови переходу між ними можуть бути винесені в окремі класи для легшого контролю над цією системою. Все це досягається за допомогою дизайн патерну Стан. Піти на право чи на ліво, а чи взагалі кудись йти? Відповідь на це питання може залежати від певних параметрів і є нічим іншим як певною Стратегією. Втомились від одноманітної роботи, яка завжди шаблонна, окрім деталей, які час від часу міняються? Віддайте цю робому Шаблонному Методу. Коли потрібно виконати деякі дії над об’єктом, причому вони кожного разу різні, такі дії можуть бути винесені в окремі класи-відвідувачі. Опісля ваш об’єкт може приймати Відвідувачів для виконання конкретних дій.