СОДЕРЖАНИЕ

1 Лабораторная работа № 9-12

Тема: Профилирование кода средствами инструментальной среды разработки. Разработка тестовых модулей проекта. Выполнение функционального тестирования и тестирования интеграции. Документирование результатов. Тестирование интерфейса пользователя.

Цель: использование технологий и инструментов отладки и тестирования, встроенных в Visual Studio для выполнения тестирования разного типа с целью обнаружения ошибок и их дальнейшее исправление, ведение документации и анализ результатов тестирования.

1. Выполните профилирование кода сайта на Bottle из предыдущей ЛР с помощью команды Отладка > Запустить профилирование Python (Debug > Launch python profiling. . .), по умолчанию в окне настроек должно отобразиться название проекта:

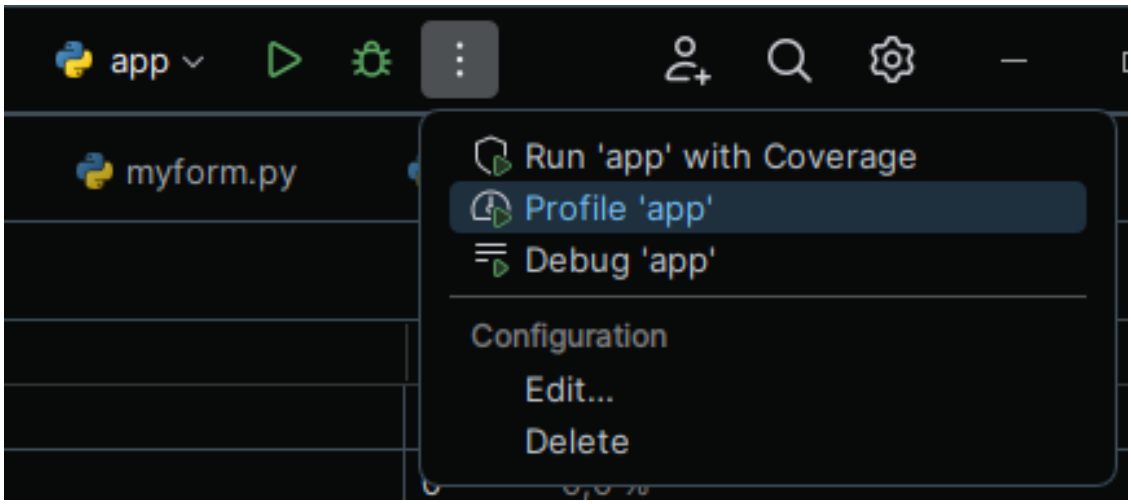
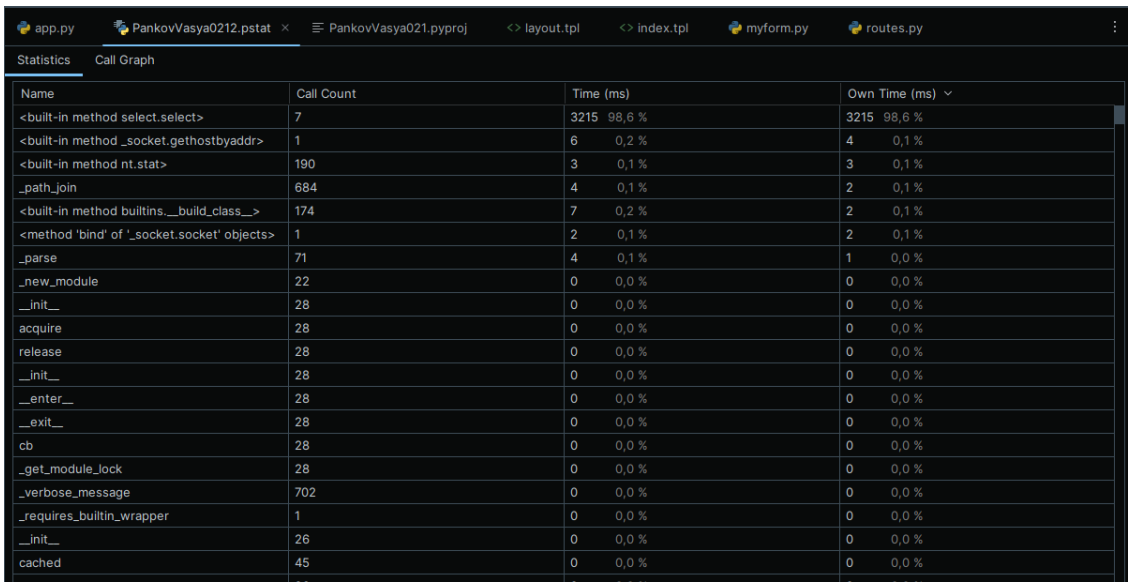


Рисунок 1 – Profiling в PyCharm

Полученный отчёт:



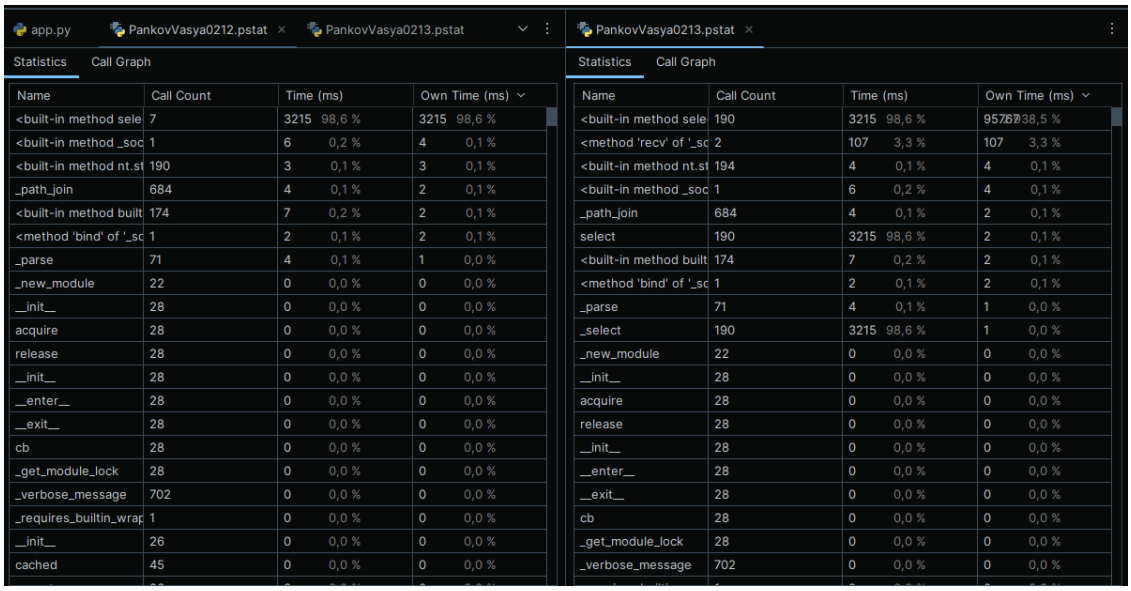
app.py | PankovVasya0212.pstat | PankovVasya021.pyproj | layout.tpl | index.tpl | myform.py | routes.py

Statistics | Call Graph

Name	Call Count	Time (ms)	Own Time (ms)
<built-in method select.select>	7	3215 98,6 %	3215 98,6 %
<built-in method _socket.gethostbyaddr>	1	6 0,2 %	4 0,1 %
<built-in method nt.stat>	190	3 0,1 %	3 0,1 %
_path_join	684	4 0,1 %	2 0,1 %
<built-in method builtins.__build_class__>	174	7 0,2 %	2 0,1 %
<method 'bind' of '_socket.socket' objects>	1	2 0,1 %	2 0,1 %
_parse	71	4 0,1 %	1 0,0 %
_new_module	22	0 0,0 %	0 0,0 %
__init__	28	0 0,0 %	0 0,0 %
acquire	28	0 0,0 %	0 0,0 %
release	28	0 0,0 %	0 0,0 %
__init__	28	0 0,0 %	0 0,0 %
__enter__	28	0 0,0 %	0 0,0 %
__exit__	28	0 0,0 %	0 0,0 %
cb	28	0 0,0 %	0 0,0 %
_get_module_lock	28	0 0,0 %	0 0,0 %
_verbose_message	702	0 0,0 %	0 0,0 %
_requires_builtin_wrapper	1	0 0,0 %	0 0,0 %
__init__	26	0 0,0 %	0 0,0 %
cached	45	0 0,0 %	0 0,0 %

Рисунок 2 – Отчёт профилирования

Сравнение двух отчётов:



app.py | PankovVasya0212.pstat | PankovVasya0213.pstat

Statistics | Call Graph

Name	Call Count	Time (ms)	Own Time (ms)
<built-in method sele	7	3215 98,6 %	3215 98,6 %
<built-in method _soc	1	6 0,2 %	4 0,1 %
<built-in method nt.st	190	3 0,1 %	3 0,1 %
_path_join	684	4 0,1 %	2 0,1 %
<built-in method built	174	7 0,2 %	2 0,1 %
<method 'bind' of '_sc	1	2 0,1 %	2 0,1 %
_parse	71	4 0,1 %	1 0,0 %
_new_module	22	0 0,0 %	0 0,0 %
__init__	28	0 0,0 %	0 0,0 %
acquire	28	0 0,0 %	0 0,0 %
release	28	0 0,0 %	0 0,0 %
__init__	28	0 0,0 %	0 0,0 %
__enter__	28	0 0,0 %	0 0,0 %
__exit__	28	0 0,0 %	0 0,0 %
cb	28	0 0,0 %	0 0,0 %
_get_module_lock	28	0 0,0 %	0 0,0 %
_verbose_message	702	0 0,0 %	0 0,0 %
_requires_builtin_wrap	1	0 0,0 %	0 0,0 %
__init__	26	0 0,0 %	0 0,0 %
cached	45	0 0,0 %	0 0,0 %

PankovVasya0213.pstat

Statistics | Call Graph

Name	Call Count	Time (ms)	Own Time (ms)
<built-in method sele	190	3215 98,6 %	95767 38,5 %
<method 'recv' of '_sc	2	107 3,3 %	107 3,3 %
<built-in method nt.st	194	4 0,1 %	4 0,1 %
<built-in method _soc	1	6 0,2 %	4 0,1 %
_path_join	684	4 0,1 %	2 0,1 %
select	190	3215 98,6 %	2 0,1 %
<built-in method built	174	7 0,2 %	2 0,1 %
<method 'bind' of '_sc	1	2 0,1 %	2 0,1 %
_parse	71	4 0,1 %	1 0,0 %
_select	190	3215 98,6 %	1 0,0 %
_new_module	22	0 0,0 %	0 0,0 %
__init__	28	0 0,0 %	0 0,0 %
acquire	28	0 0,0 %	0 0,0 %
release	28	0 0,0 %	0 0,0 %
__init__	28	0 0,0 %	0 0,0 %
__enter__	28	0 0,0 %	0 0,0 %
__exit__	28	0 0,0 %	0 0,0 %
cb	28	0 0,0 %	0 0,0 %
_get_module_lock	28	0 0,0 %	0 0,0 %
_verbose_message	702	0 0,0 %	0 0,0 %

Рисунок 3 – Сравнение двух отчётов

2. Выполните примеры unit-тестов для простейшего модуля Python с функциями калькулятора. Создайте консольное приложение как в ЛР№1. Модуль calc.py будет представлять собой библиотеку, содержащую функции для выполнения основных арифметический действий:

```
def add(a, b):
    return a + b

def sub(a, b):
    return a-b

def mul(a, b):
```

```
    return a * b

def div(a, b):
    return a / b
```

Для того, чтобы протестировать эту библиотеку, создайте отдельный файл с названием $test_{calc.py}$ и поместите туда функции, которые проверяют корректность работы модуля:

```
import calc

def test_add():
    if calc.add(1, 2) == 3:
        print("Test add(a, b) is OK")
    else:
        print("Test add(a, b) is Fail")

def test_sub():
    if calc.sub(4, 2) == 2:
        print("Test sub(a, b) is OK")
    else:
        print("Test sub(a, b) is Fail")

def test_mul():
    if calc.mul(2, 5) == 10:
        print("Test mul(a, b) is OK")
    else:
        print("Test mul(a, b) is Fail")

def test_div():
    if calc.div(8, 4) == 2:
        print("Test div(a, b) is OK")
    else:
        print("Test div(a, b) is Fail")


test_add()
test_sub()
test_mul()
test_div()
```

Запустите $test_{calc.py}$, выбрав в его контекстном меню Запуск без отладки.

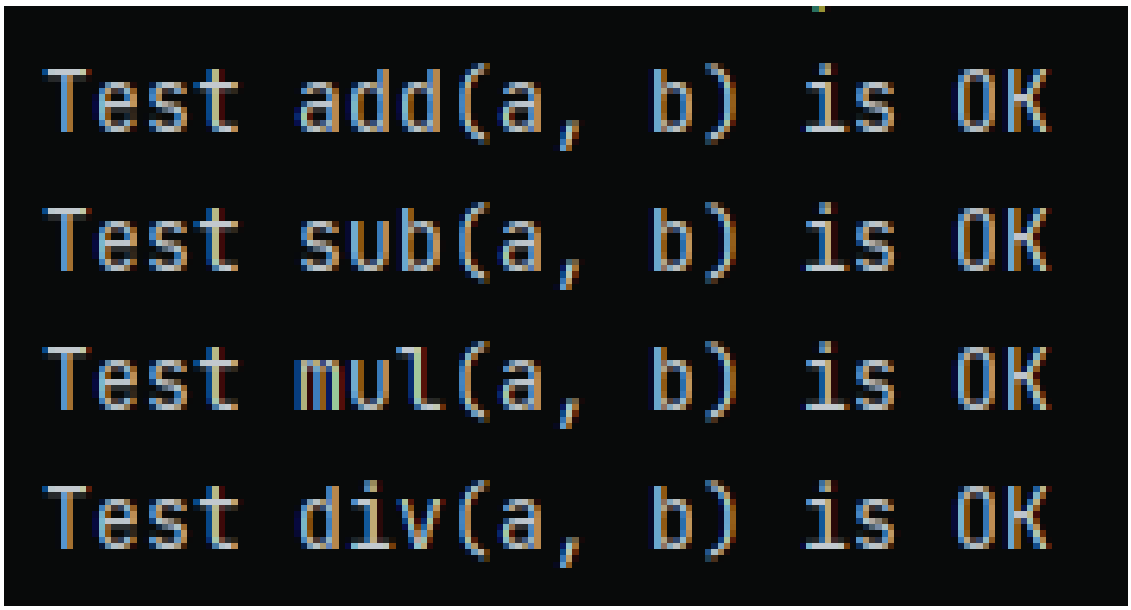


Рисунок 4 – Тестирование `calc`

3. Для тестирования набора функций из calc.py с помощью unittest создайте файл с именем $\text{utest}_{\text{calc.py}}$ и следующим кодом:

```
import unittest
import calc

class CalcTest(unittest.TestCase):
    def test_add(self):
        self.assertEqual(calc.add(1, 2), 3)

    def test_sub(self):
        self.assertEqual(calc.sub(4, 2), 2)

    def test_mul(self):
        self.assertEqual(calc.mul(2, 5), 10)

    def test_div(self):
        self.assertEqual(calc.div(8, 4), 2)

if __name__ == '__main__':
    unittest.main()
```

Рисунок 5 – Запуск тестов

Можно сделать запрос расширенной информации по пройденным тестам, для этого необходимо добавить ключ –v. Запустите скрипт

$utest_{calc.py}$ в терминале посредством контекстного меню. Введите команду python -m unittest -v $utest_{calc.py}$.

```
(venv) PS C:\Users\user\PycharmProjects\testing> python -m unittest -v utest_calc.py
test_add (utest_calc.CalcTest) ... ok
test_div (utest_calc.CalcTest) ... ok
test_mul (utest_calc.CalcTest) ... ok
test_sub (utest_calc.CalcTest) ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s
```

Рисунок 6 – Показ всех тестов

4. Создайте ещё одно консольное приложение. Подготовьте VS для создания unit-тестов с помощью фреймворка. Для этого щелкните проект правой кнопкой мыши в обозревателе решений и выберите платформу unittest на вкладке Тест в области свойств (или нажав на значок свойств в панели его инструментов).

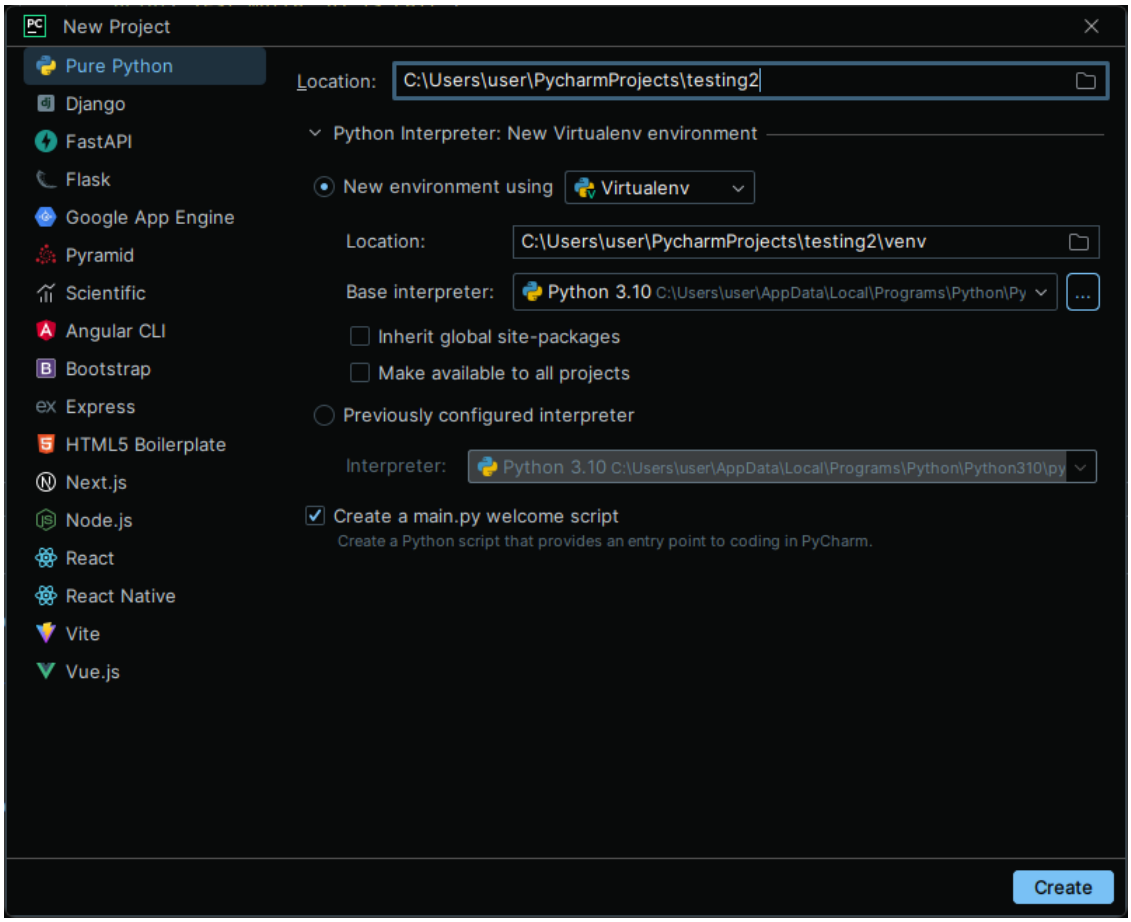


Рисунок 7 – Создание проекта в PyCharm

Создайте два unit-теста для проверки работы регулярного выражения на соответствие email (вынесите формирование и сравнение со строкой

в главный модуль myform$_{\text{mail.py}}$), в одном используя метод assertFalse, а в другом – assertTrue (подробнее о методах в официальной документации или статье `https://tirinox.ru/unit-test-python/`).

В unit-тесте с assertFalse в цикле проверяйте список с примерами email неверного формата (перечислить возможные тестовые случаи, не менее 12 с разными типами ошибок), например, list$_{\text{mailuncor}}$ = [, "1 "m1@ "@mail . . .], а в unit-тесте с assertTrue – список с корректными email, к примеру, list$_{\text{mailcor}}$ = ["m.m@mail.ru "m1@gmail.com . . .].

```
from re import fullmatch


def mail_is_correct(mail: str):
    """Проверка почты на корректность"""
    return
    ↪  fullmatch(r"[a-zA-Z0-9._&='\-+]{1,256}@[a-zA-Z0-9]{1,100}\.[a-zA-Z0-9]{1,7}",
    ↪  mail) is not None
```

Listing 1: `my_form_mail.py`

```
import unittest

from PankovVasya021.my_form_mail import *


class FormTest(unittest.TestCase):
    """Тесты на правильность проверки электронной почты"""

    # Список с неправильными значениями почты
    list_mail_uncor = ["", "a@", "pa", "pank@a", "@a", "pank@pank", "       @    . ",
    ↪  "pank@pank.", "pank@pank,su",
                       "pank@pank.su, pank@pank.su", "        ", "vasya.pankov@ma",
                        ↪  "pa@pa@.com", "misha....palok"]
    # Список с привильными значениями почты
    list_mail_cor = ["pank@pank.su", "vasya.pankov26@gamil.com", "vasya_@mail.com",

                     ↪  "veryveryveryveryveryveryveryveryveryveryveryveryvery@longlonglonglong.e
                     ↪  "&=_'-+@sym.love",
                     "SENSETIVE@UPPER.CASE"]

    def test_F_mail(self):
        """Проверка некорректных адресов"""
        for mail in self.list_mail_uncor:
            self.assertFalse(mail_is_correct(mail))

    def test_T_mail(self):
        """Проверка корректных адресов"""
        for mail in self.list_mail_cor:
            self.assertTrue(mail_is_correct(mail))
```

Listing 2: `ctest_form_mail.py`

Ошибка:

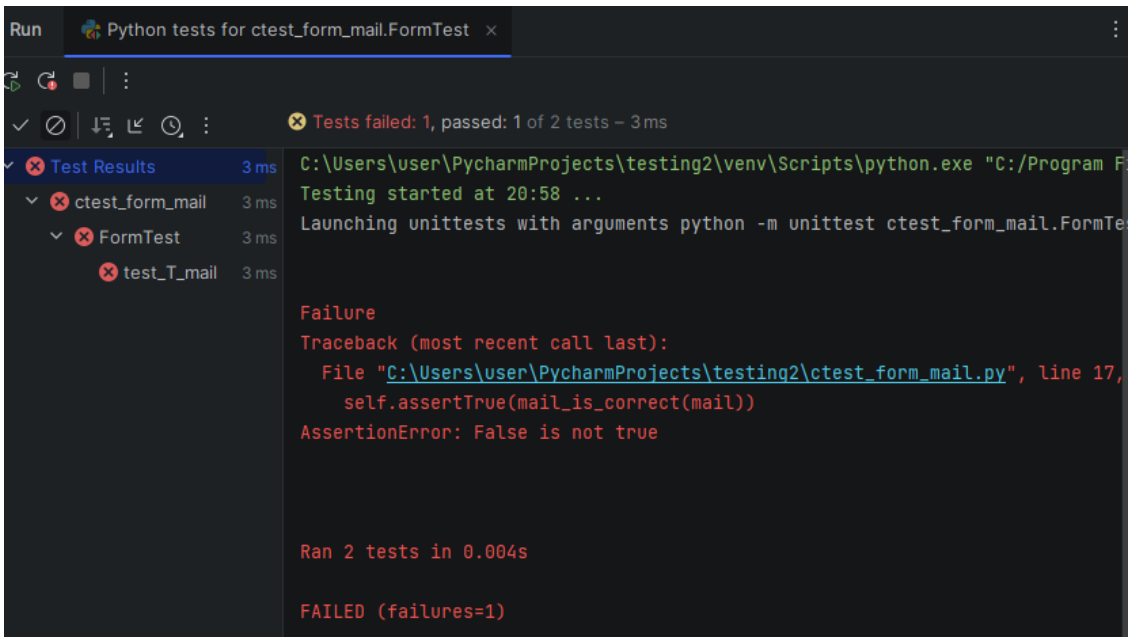


Рисунок 8 – Демонстрация ошибки

Суть ошибки в том, что все не все символы были в проверке, которые могут быть в почте могут быть в почте.

Исправленная версия регулярного выражения:

```
r"[a-zA-Z0-9._&='\-+]{1,256}@[a-zA-Z0-9]
{1,100}\.[a-zA-Z0-9]{1,7}"
```

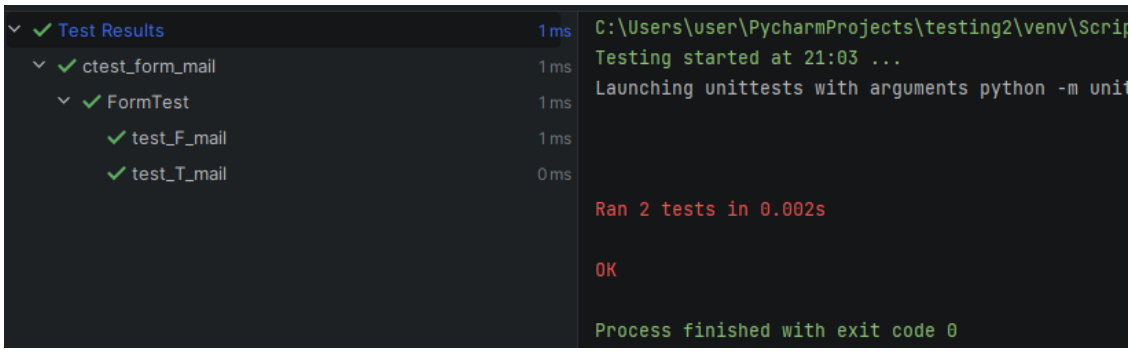


Рисунок 9 – Исправленая версия

5. Вернитесь к приложению Bottle. Интегрируйте модули кода с проверкой работы регулярного выражения и соответствующими unit-тестами в свой проект. Выполните запуск тестов и отладку работы проекта. Зафиксируйте в Git соответствующие изменения (сделайте не менее 2 коммитов). Сделайте push на Github. Задокументируйте все результаты работы.

   Перенёс файлы функции проверки почты и её тестирование в проект, тест выделил в директорию tests.

Рисунок 10 – Первый коммит

Интегрирую функцию в код:

Рисунок 11 – Демонстрация интеграции

Сделал коммит.

Рисунок 12 – История коммитов

6. С помощью инструментария онлайн-сервиса `https://wave.webaim.org/` выполните проверку валидации и тестирование юзабилити следующих сайтов:

   – `http://bottlepy.org/docs/dev/index.html`
   – `https://pypi.org/`
   – `https://python-scripts.com/web-frameworks`.

   Перечислите основные ошибки и предупреждения, назовите структурные элементы интерфейса и особенности (перейти по соответствующим вкладкам раздела «Summary», сделать заключение структуре).

   (a) `http://bottlepy.org/docs/dev/index.html`
   Общий отчёт:

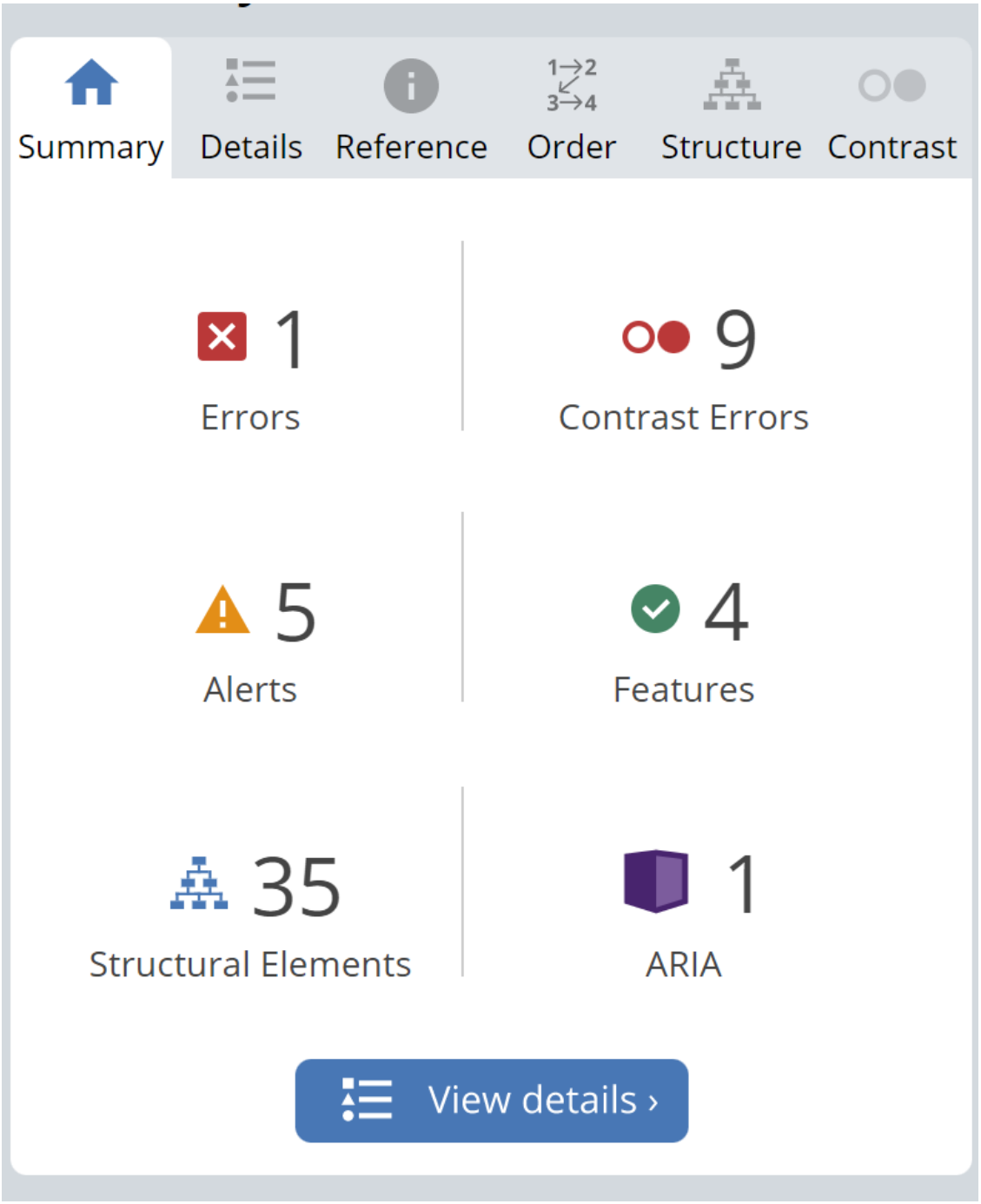


Рисунок 13 – Десонстрация общего отчёта

Демонстрация ошибок:

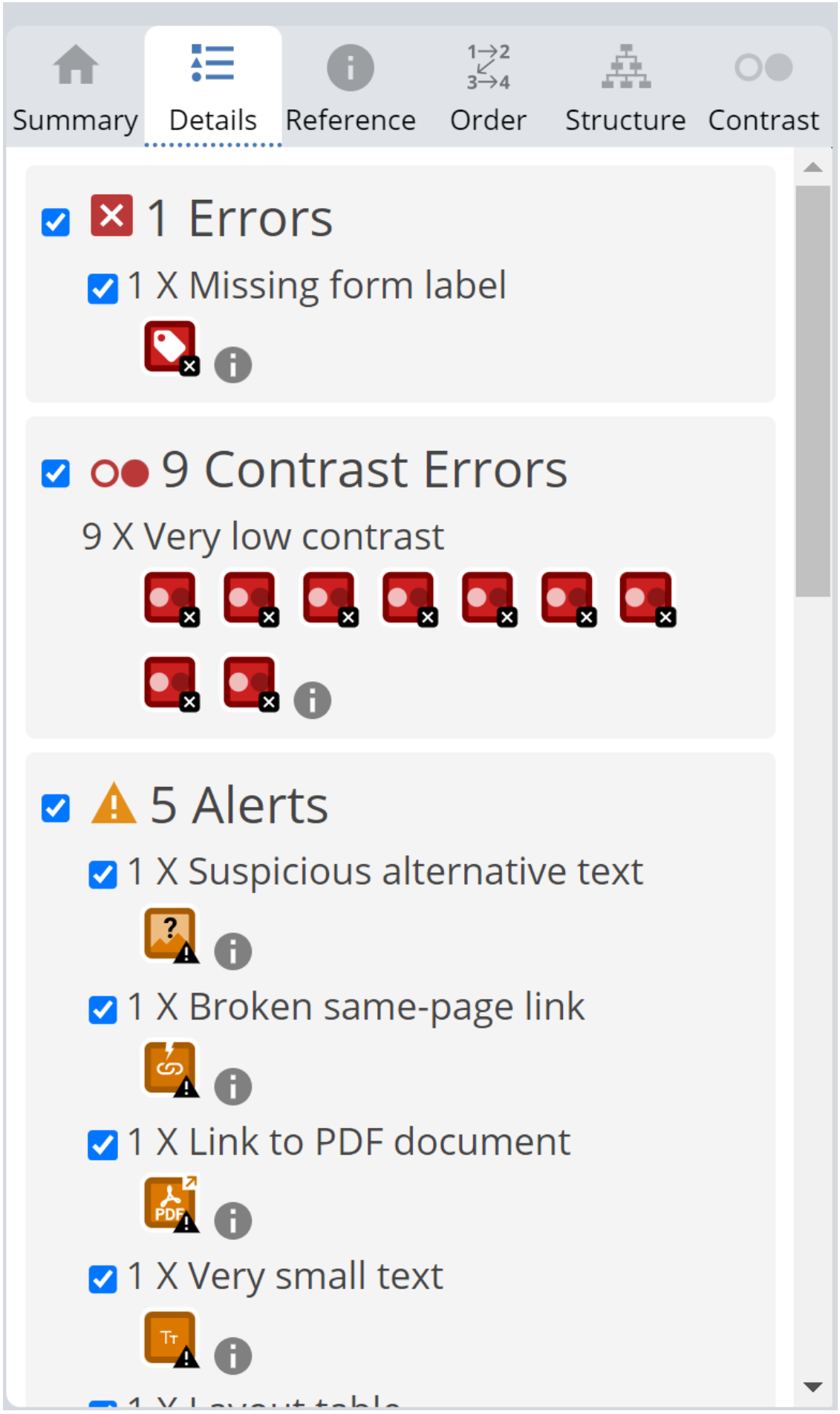


Рисунок 14 – Ошибки подробнее

Главная ошибка, это остутствие подписи у формы:

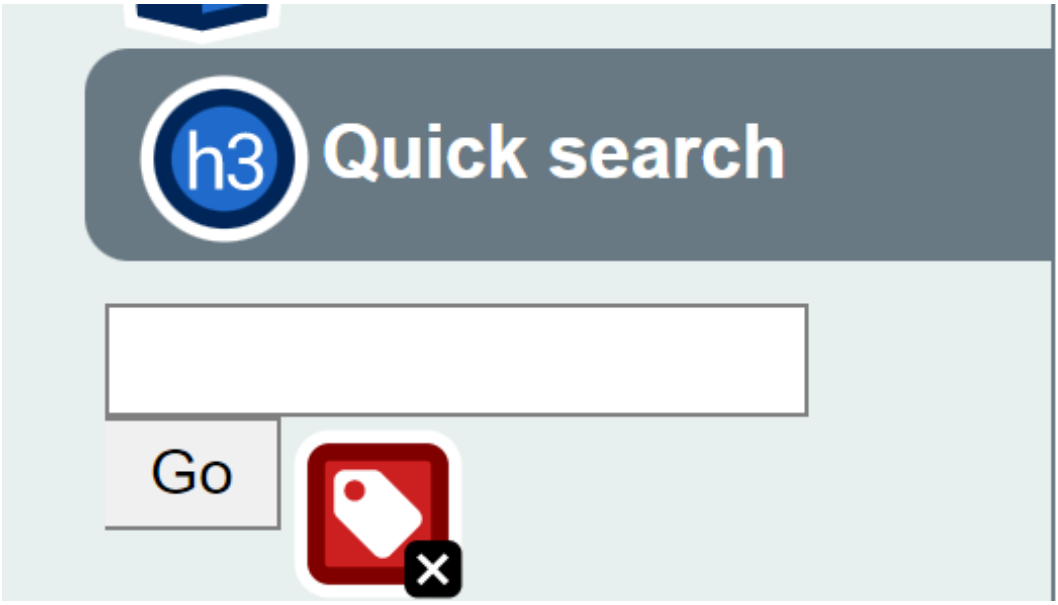


Рисунок 15 – Отсутствие подписи у формы

Фактически подпись есть выше, но есть проблема в том, что данная подпись не лежит в форме, и не помечена как label.

Также было выделено множество ошибок констраста, связаных, с подсветкой кода.

```
from bottle import route, run, template

@route('/hello/<name>')
def index(name):
    return template('<b>Hello {{name}}</b>!', name=name)

run(host='localhost', port=8080)
```

Рисунок 16 – Ошибки констраста

Предупреждения:

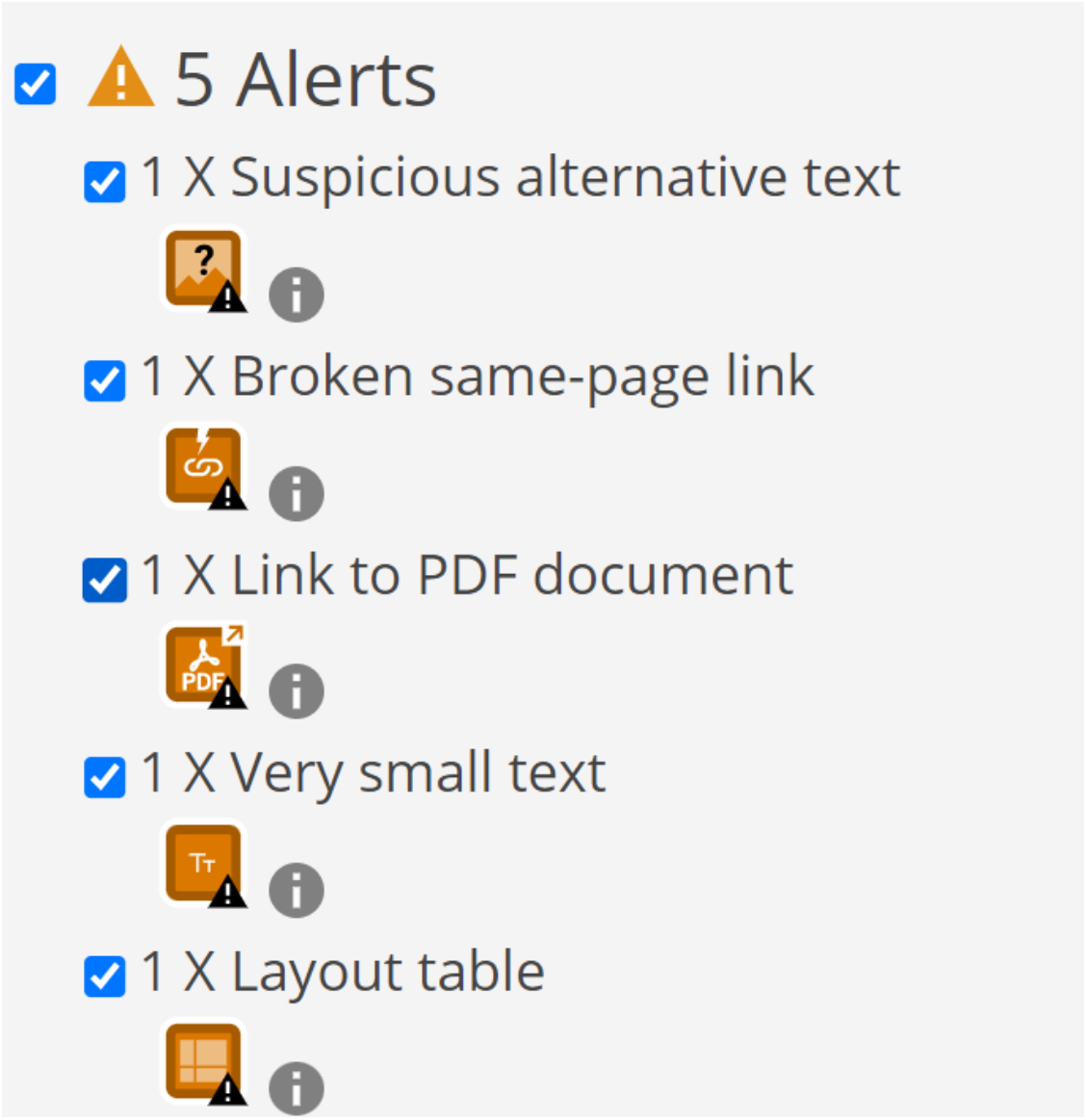


Рисунок 17 – Демонстрация предупреждений

Подозрительный альтернативный текст: у логотипа альтернативный текст не отражает содержание логотипа. Текст для логотипа: “logo”. Понятно, что содержание картинки отражается довольно слабо, хотя при этом при отсутствии изображения можно будет понять, что в этом месте должен был находиться логотип

Сломанная ссылка: создатели сайта хотели сделать переход на конкретное место страницы, где указаны данные о лицензии. Но у них что-то пошло не так, и поэтому ссылка перенаправляет просто на саму страницу

Ссылка на pdf-документ - не на всех устройствах можно с лёгкостью открыть pdf файл, к примеру на Android, необходимо скачать файл и открыть через отдельное приложение.

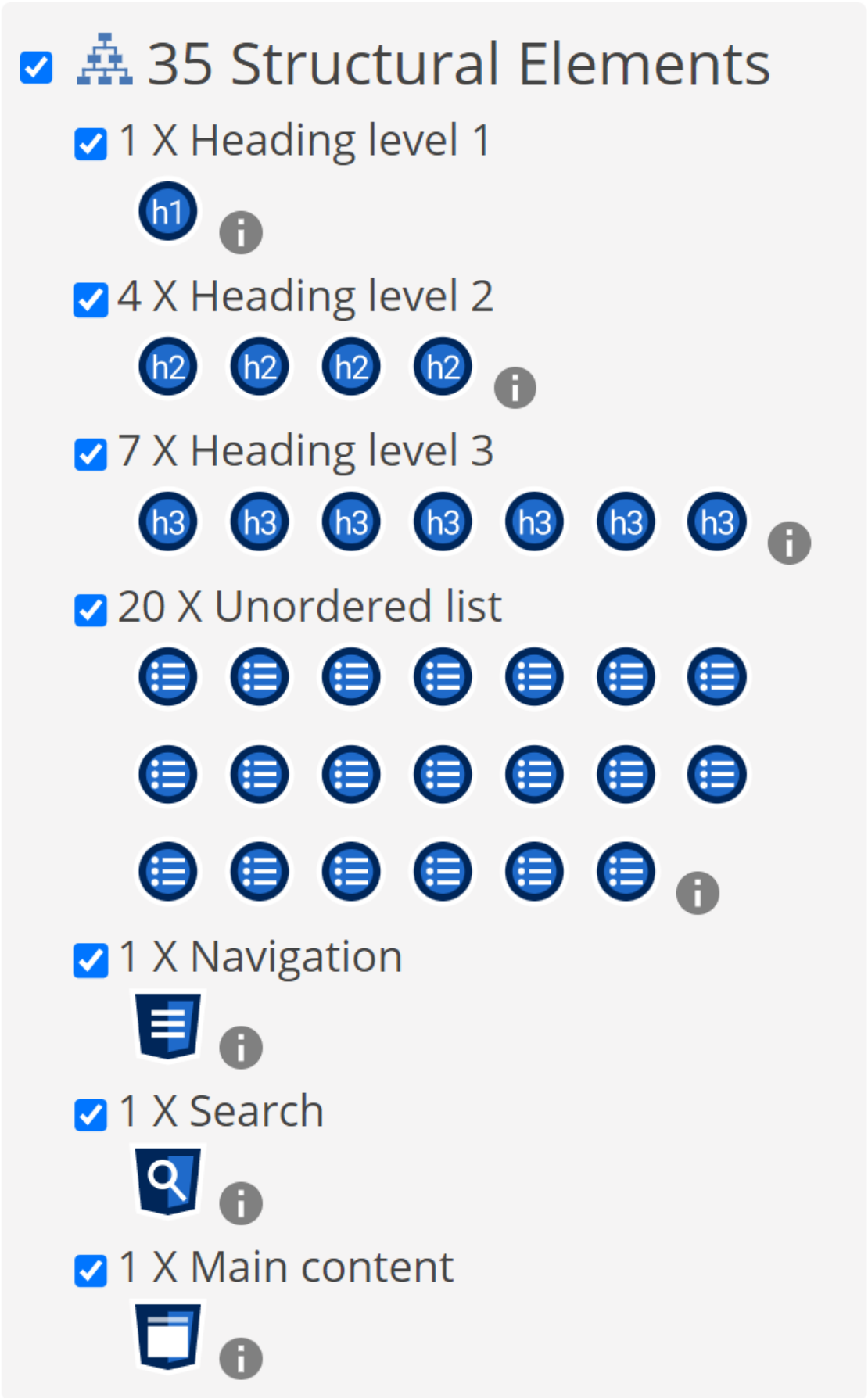


Рисунок 18 – Структурные элементы

Перечисление:

- Заголовки - используются для обозначения заголовков и подзаголовков
- Неупорядоченные списки - используются для организации данных
- Навигация - элемент, используемый для создания навигационных элементов, т.е. элементов для передвижения по сайту
- Поиск
- Main content - главное содержимое страницы. В него входят все остальные элементы

Иерархия:

- Main content:
  - Heading 1:
    * Heading 2:
      · Unordered list
- Navigation:
  - Heading 3:
    * Unordered list
      · Search

Особенности:

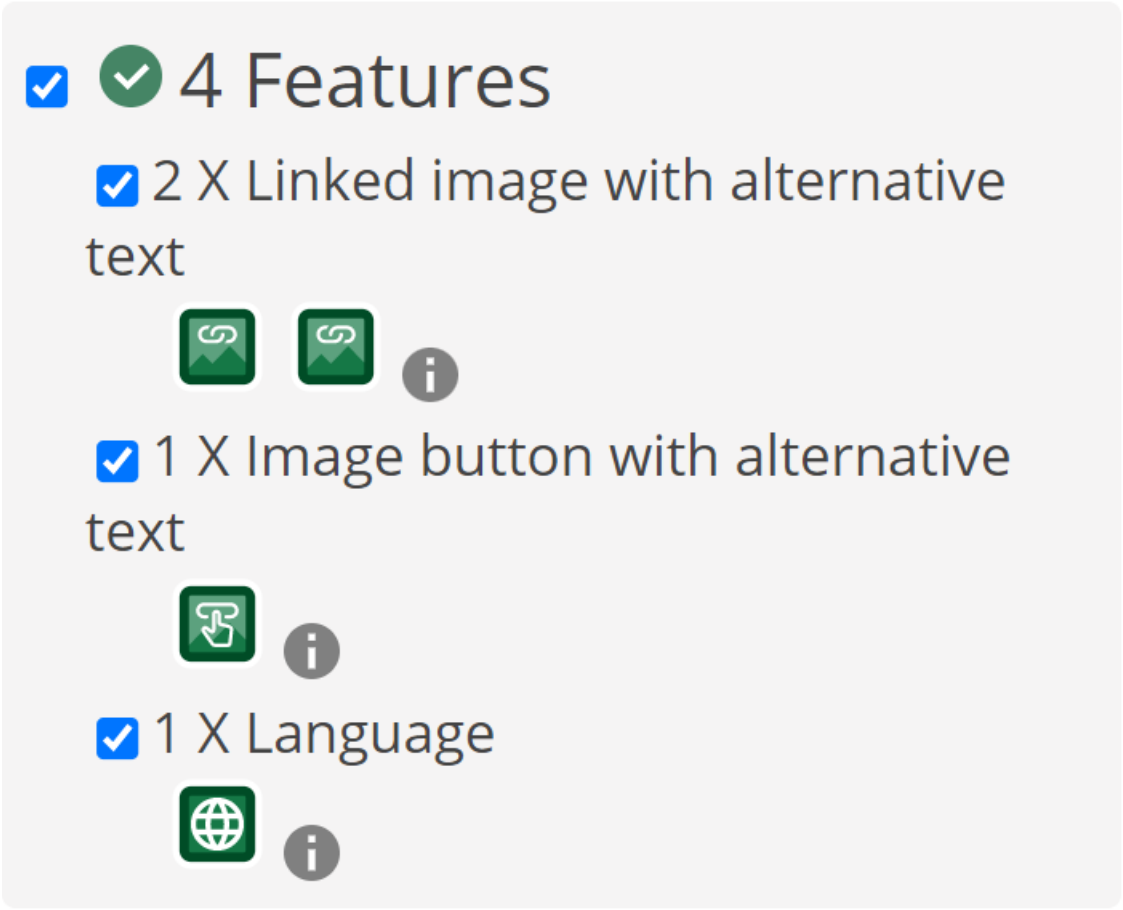


Рисунок 19 – Особенности

- Изображение ссылка
- Кнопка с изображением
- Возможность смены языка

(b) `https://pypi.org/`
Общий отчёт:

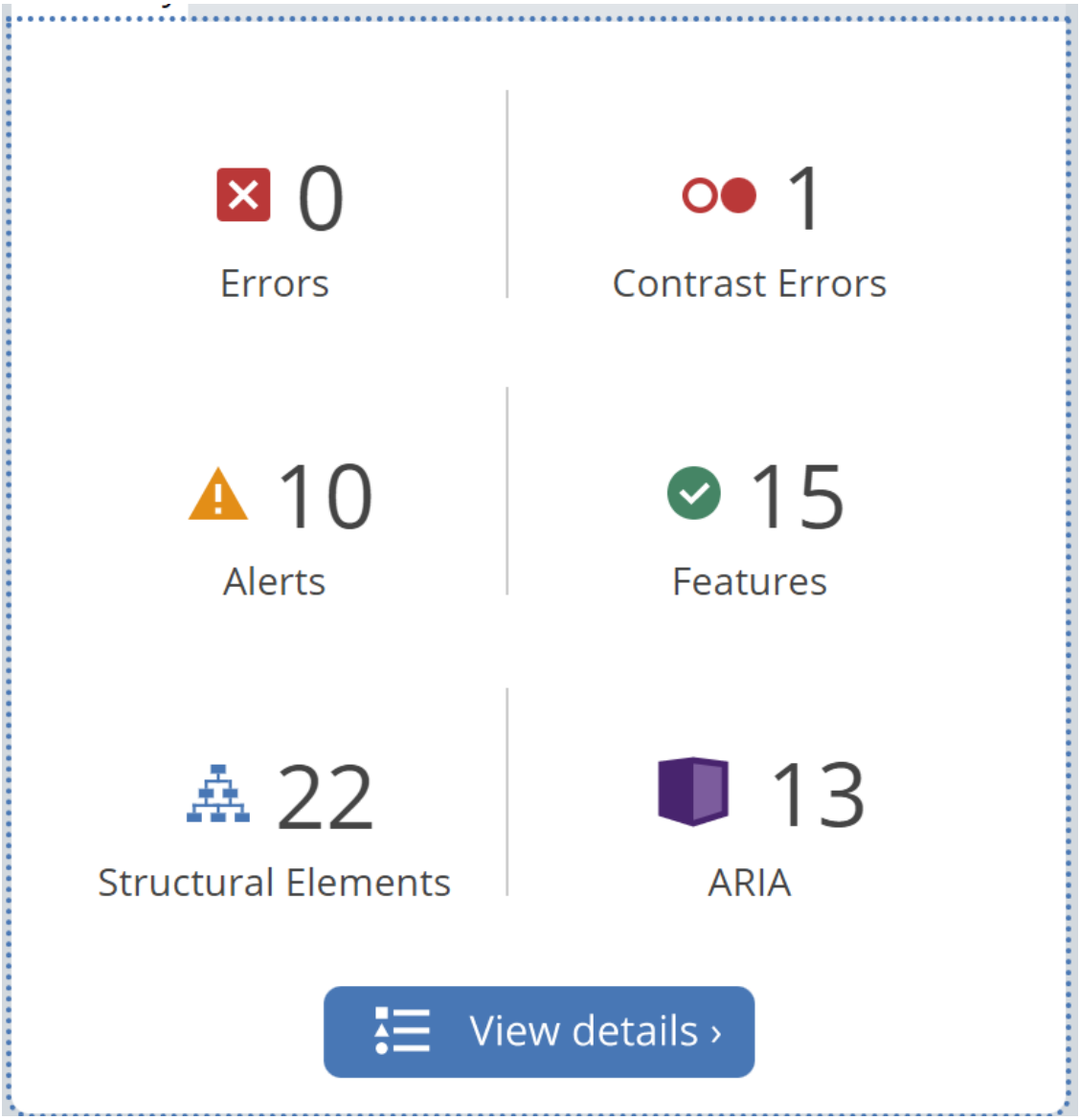


Рисунок 20 – screenshot @ 2023-05-03 10:12:53

Ошибок у данного сайта нет, кроме контрастных и то одна:

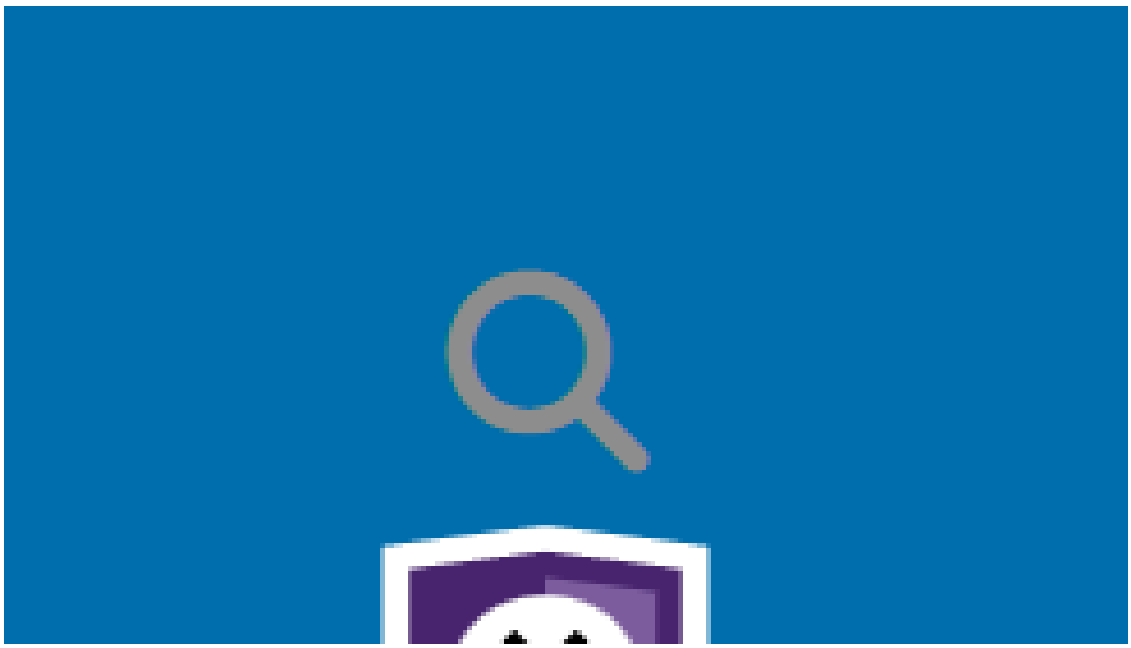


Рисунок 21 – Контрасная ошибка

Предупреждения:

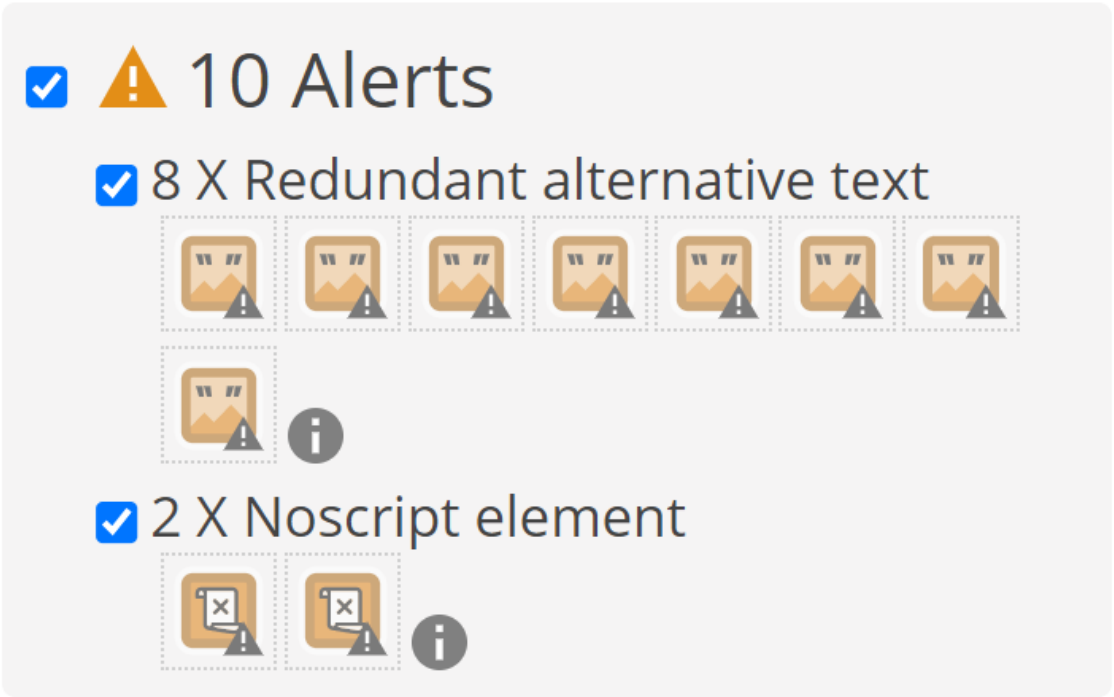


Рисунок 22 – Предупреждения

Reduntant alt text - альтернативный текст дублируется текстом рядом с картинкой

No script - присутствуют элементы, которые будут отображаться, если выключен javascript

Структурные элементы:

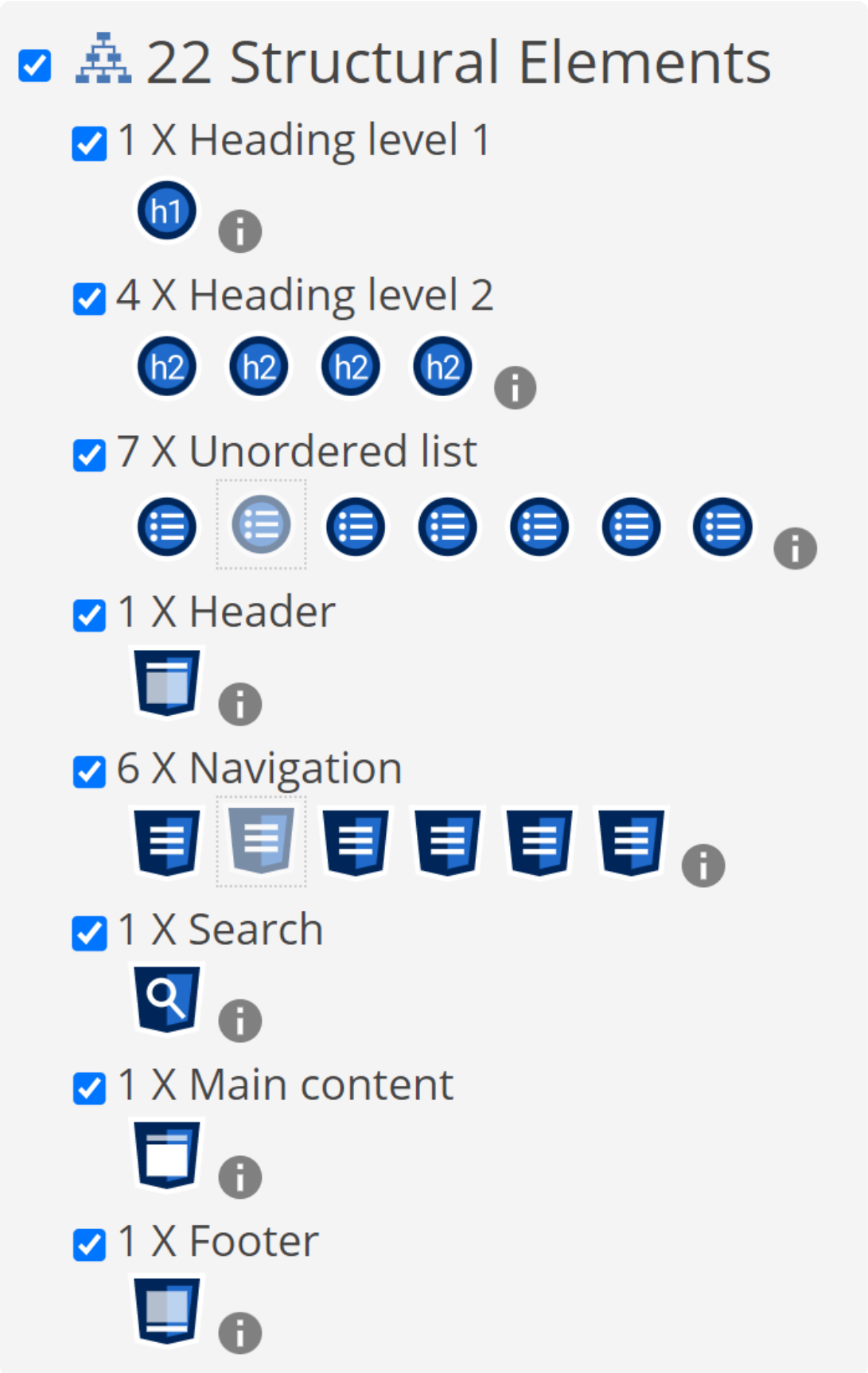


Рисунок 23 – Структурные элементы

Иерархия:

- Header:
  - Heading level 1
  - Search
  - Unordered list
- Navigation
- Main content
- Footer
  - Unordered list
  - Heading level 2

Особенности:

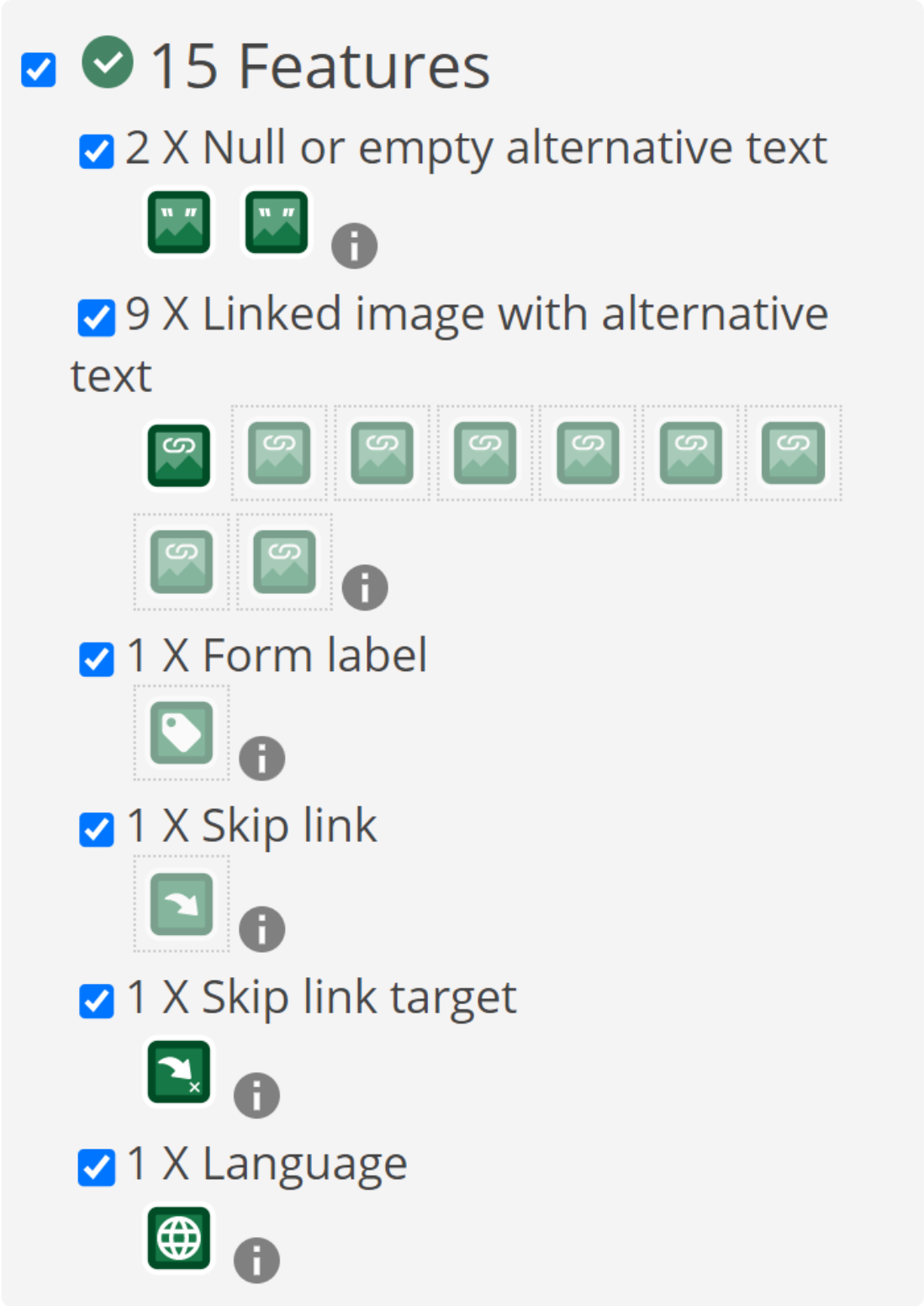


Рисунок 24 – Особенности

- Пустой альтернативный текст
- Изображение-ссылка
- Подпись к форме
- Skip-link - ссылка, которая позволяет перейти к другому месту страницы

- Skip-link target - назначение предыдущего пункта
- Возможность изменения языка

(c) `https://python-scripts.com/web-frameworks`
Ошибки:

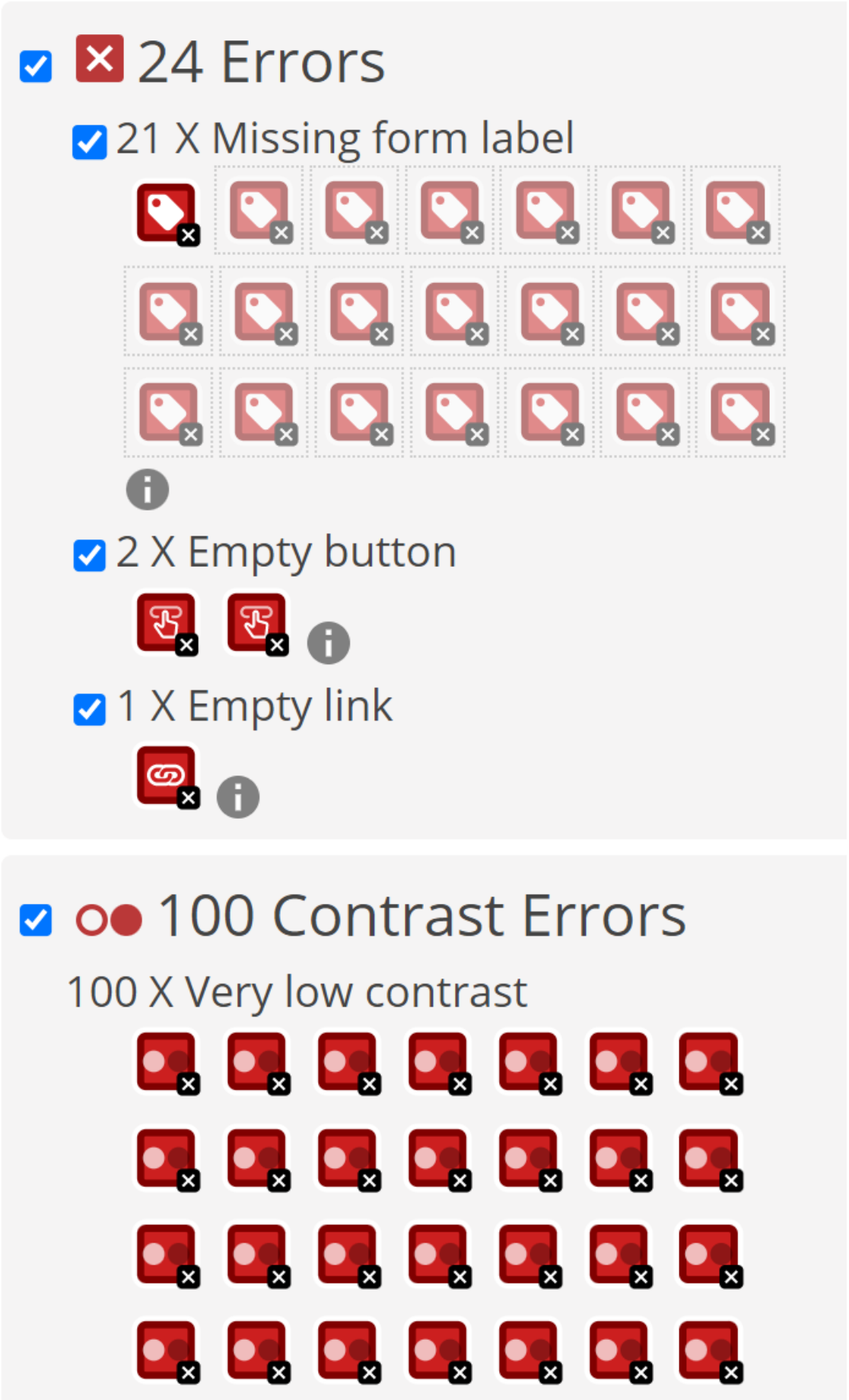


Рисунок 25 – Ошибки

– Отсутствует подпись у формы
– Кнопка не имеет текста
– Ссылка не имеет текста
– Много ошибок контраста

Предупреждения:

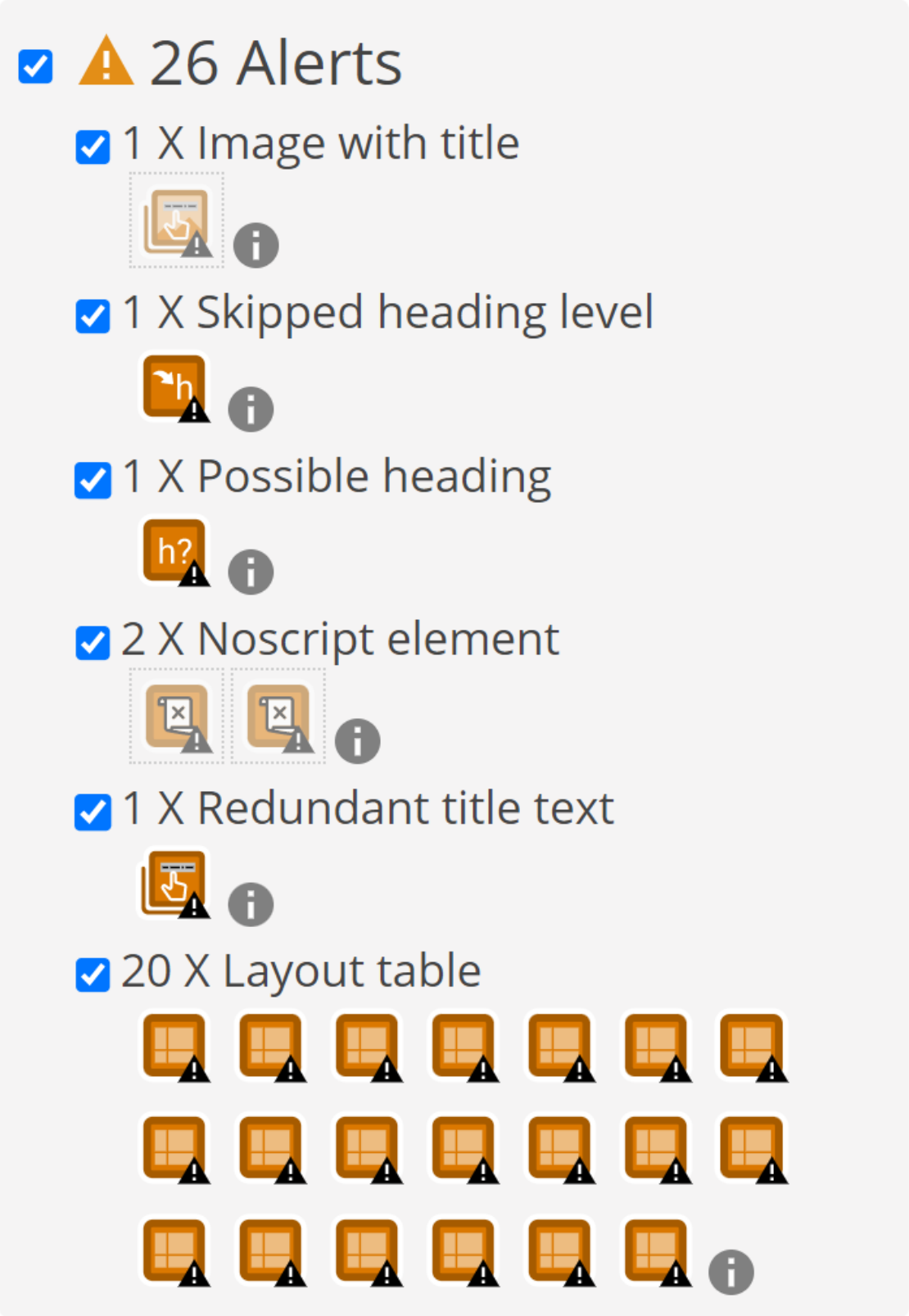
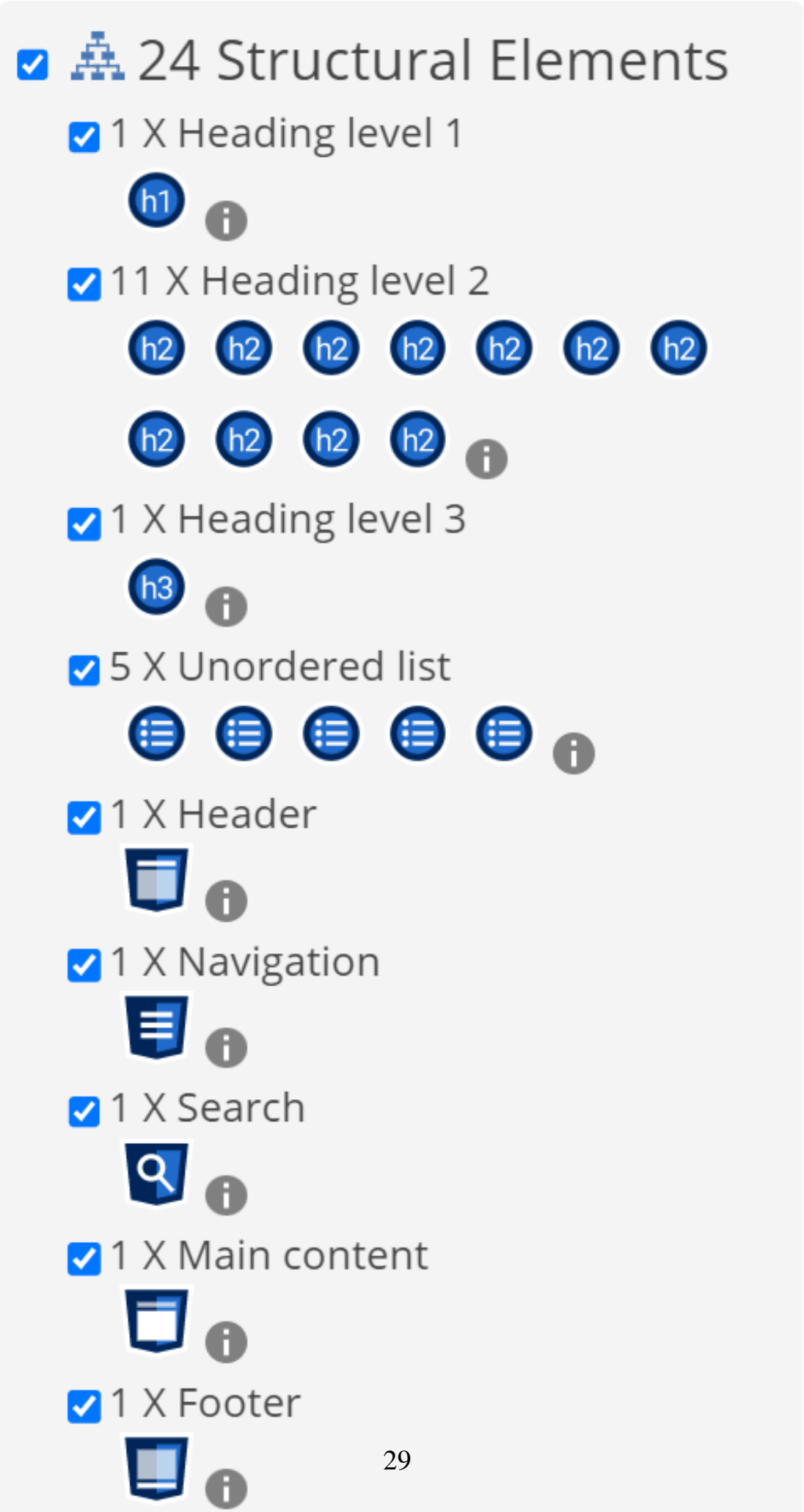


Рисунок 26 – Предупреждения

– Изображение имеет заголовок, но не имеет альтернативного текста
– Пропущен уровень заголовка

- Текст возможно сделать заголовком, но он не в заголовочном теге
- No script
- Дублирующийся заголовок
- Layout table

Структурные элементы:


24 Structural Elements
1 X Heading level 1
h1
11 X Heading level 2
h2 h2 h2 h2 h2 h2 h2
h2 h2 h2 h2
1 X Heading level 3
h3
5 X Unordered list
1 X Header
1 X Navigation
1 X Search
1 X Main content
1 X Footer

Иерархия:

- Header
  - Unordered list
  - Search
- Navigation
  - Unordered list
- Aside
  - Unordered list
- Main content
  - Heading level 1
  - Heading level 2
    * Unorderd list
- Footer
  - Unordered list

Особенности:

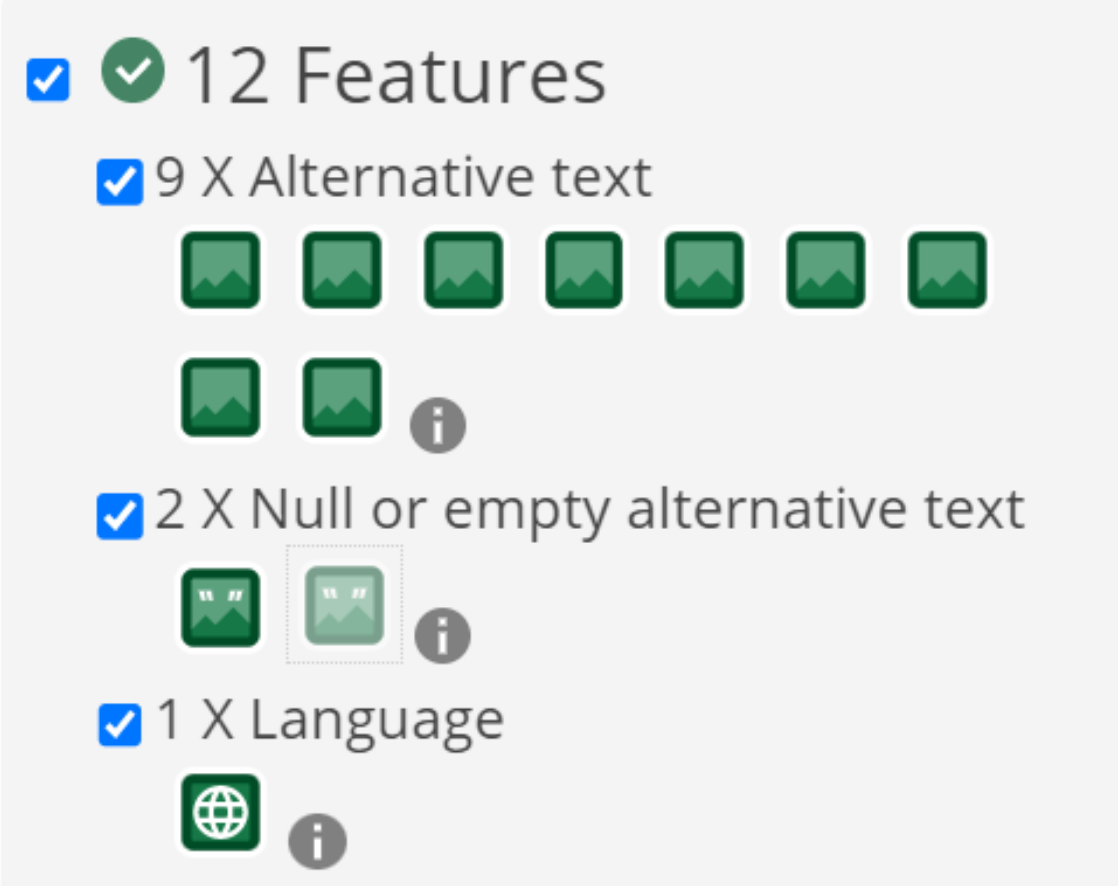


Рисунок 28 – Особенности

- Пустой альтернативный текст

– Возможность смены языка

7. Установите на виртуальную машину TestComplete для автоматизированного тестирования интерфейса (можно скачать с сайта `https://smartbear.com/product/testcomplete`). Изучите работу встроенного демонстрационного примера для веб-интерфейса, запустив тест (к нему можно перейти со стартовой страницы):

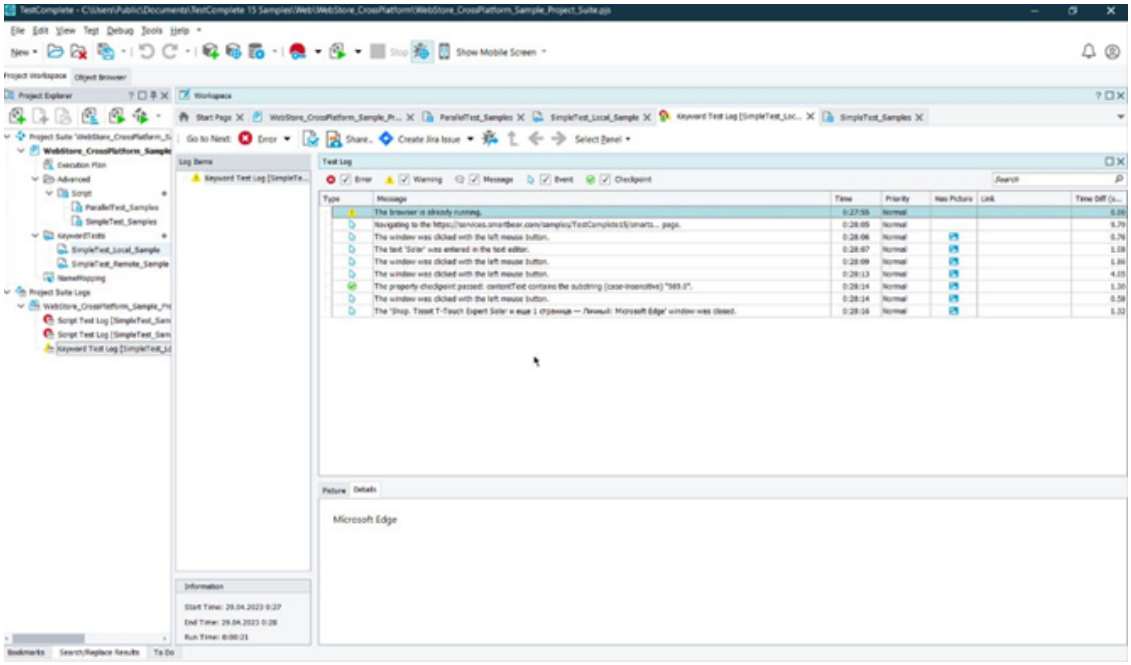


Рисунок 29 – TestComplete

Контрольные вопросы:

1. Что означает протестировать интерфейс?

   Проверить, соответствует ли пользовательский интерфейс программного обеспечения требованиям, и удобно ли пользователям работать с программным продуктом.

2. Какие разновидности тестов вам известны?

   – Модульное
   – Интеграционное
   – Системное
   – Автоматизированное
   – Ручное
   – Тестирование производительности
   – Тестирование безопасности

– Тестирование совместимости

3. Назовите основные аспекты для документирования при ручном и автоматическом тестировании? При ручном и автоматическом тестировании ПО необходимо документировать следующие аспекты:

   (a) Цель тестирования

   (b) Тестовые сценарии

   (c) Результаты тестирования

   (d) Описание среды тестирования

   (e) Информация о тестировщике

   (f) Информация о версии ПО

   (g) Информация об автоматизированных тестах

   (h) Описание проблем и ошибок

   (i) Информация о статусе тестирования

   (j) Рекомендации и доработки