МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Алгоритм автоматизованого тестування інтернет ресурсів на базі фреймворка Yii2»

Завідувач випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Проценко О.Б.**

Студентка групи ІН.мдн-91С **Приходько К.О.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЦЗДВН* Кафедра *Комп'ютерних наук*
Спеціальність *122 «Комп'ютерні науки»*

Затверджую:
зав. кафедри _____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Приходько Катерини Олександрівни
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Алгоритм автоматизованого тестування інтернет ресурсів на базі фреймворка Yii2*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Інформаційний огляд. 2) Вибір методу рішення. 3) Практична реалізація*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1	*Інформаційний огляд*		
2	*Вибір методу рішення*		
3	*Практична реалізація*		
4	*Оформлення кваліфікаційної магістерської роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис

РЕФЕРАТ

Записка: 57 стор., 9 рис., 1 таблиця, 1 додаток, 16 джерел.

Об’єкт дослідження — алгоритм автоматизованого тестування інтернет ресурсів.

Мета роботи — створити алгоритм автоматизованого тестування інтернет ресурсів.

Методи дослідження — в процесі створення застосовуються технології проектування інформаційних веб-систем на мові PHP, фреймворк Yii2 та мова JavaScript. Також використовується аналітичний метод, за допомогою якого здійснюється вибір оптимального програмного забезпечення.

Результати — проведено аналіз матеріалів по даній темі, обрані методи для реалізації веб-системи та алгоритму автоматизованого тестування, розроблена програмна частина є готовою до використання.

HTML, CSS, JAVASCRIPT, ВЕБ-САЙТ, PHP, ВЕБ-СИСТЕМА, БАЗА ДАНИХ, ТЕСТУВАННЯ, ФРЕЙМВОРК

ЗМІСТ

ВСТУП

Сьогодні майже кожна людина використовує інтернет, як в своїх особистих цілях так і в робочих. З кожним роком зростає кількість підприємців які вирішують перенести свій офлайн бізнес в онлайн, тому створюється все більше різноманітних сайтів. Цим самим вони економлять час як собі так і потенційним клієнтам та користувачам їх ресурсів.

Для створення інтернет ресурсу підприємець звертається до організацій які займаються його створенням. Ці організації розробляють саме те рішення яке потребує замовник та видають готовый продукт. Одним із етапів розробки є тестування продукту.

На сьогоднішній день не одна розробка веб-ресурсу не проходить без тестування. Цим етапом займаються спеціально навчені люди. Тестувати можна як окремі частини веб ресурсу так і всього одразу. Дивлячись на тенденцію розвитку інтернету та інтернет ресурсів можно зробити висновок, що тестування та сама розробка сайтів буде розвиватись надалі з геометричною прогресією ще багато десятків років.

Існує декілька видів тестування, але виділяють два основних це мануальне та автоматизоване. Мануальне тестування проходить більш повільніше бо самі люди все перевіряють, а автоматизоване відбувається набагато швидше тому, що все робить алгоритм, який за лічені секунди перевіряє як окремі частини програми так і взаємодії програмних компонентів, тому долучно зауважити, що автоматизоване тестування є більш прогресивним видом.

Отже, автоматизоване тестування є актуальним в сьогоденні, так як воно полегшує саме тестування та робить його швидше. Саме тому метою роботи було обрано створення алгоритму для тестування інтернет ресурсів.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Класифікація веб-систем

Веб-сайт (від англ. «web»-павутина «site»-місце) — це побудована певним образом деяка інформація в структурованій формі, яка розташована на сервері та являється доступною користувачам в мережі як для обмеженого та вільного доступу. Також, веб-сайт є сукупністю пов’язаних веб-сторінок, які мають унікальне доменне ім’я. Для того, щоб отримати доступ до веб-сайта, потрібно ввести його доменне ім’я в адресному рядку веб-переглядача, і браузер відобразить головну веб-сторінку веб-сайту або видкову сторінку цього сайту.

Веб-сторінки, як правило, являють собою поєднання тексту та інших засобів масової інформації. Тим не менш, немає правил, які б диктували форму веб-сайту. Людина може створювати безліч різноманітних веб-сайтів, однак багато веб-сайтів дотримуються стандартного шаблону головної сторінки, що посилається на інші категорії та вміст на веб-сайті.

Кожна сторінка - це єдиний документ HTML, і всі вони пов’язані за допомогою гіперпосилань, які можна зручно поєднати на панелі навігації.

Сукупність всіх сайті створює Всесвітню павутину. Саме ця павутина поєднує в собі шматочки інформації зі всієї світової спілки. Протокол HTTP (Hyper Text Transfer Protocol — протокол передачі гіпертексту) був розроблений спеціально для того, щоб надавати користувачу доступ до сайта який знаходиться на серверах без перешкод.

Веб-сайти розміщуються на серверах і вимагають відвідування веб-браузера, такого як Chrome, Firefox або Internet Explorer на комп’ютері чи мобільному пристрої.

Перший веб-сайт був створений в 1990 році Тімом Бернерс-Лі, британським фізиком з CERN (Європе́йська організація з я́дерних досл́іджень). Через 3 роки, у 1993 році, CERN оголосив, що кожен може отримати доступ до Інтернету та користуватися ним безкоштовно.

Сьогодні можна дати визначення веб-сайту як сукупність інформаційних, програмних та медійних засобів, які є логічно та структуровано пов’язані між собою за певними правилами.

Веб-сайт виконує наступні задачі:

- реклама ідей, продукції та послуг;
- продажа інформації, товарі та послуг;
- безкоштовна подача послуг чи інформації;
- підтримка користувачів та клієнтів веб-сайту.

Веб-система — це клієнт-серверна програма, в якій головна частина розташована на сервері який є віддаленим, а інтерфейс який призначений для користувача показується в інтернет браузері у вигляді веб-сторінки.

Для того, щоб запустити веб-систему користувачу достатньо мати браузер з доступом до Інтернету, який встановлений майже на всіх пристроях за замовчуванням, тому ніяких додаткових програм використовувати та встановлювати не потрібно. Для побудови веб-системи, а саме серверної частини використовують різноманітні мови програмування, яких з кожним роком стає все більше. Найвідоміші мови програмування таких веб-систем на сьогодні є Perl, ASP, C/C++, PHP, Java, Ruby та інші.

Для створення та реалізації запланованої веб-дизайнерами клієнтської частини використовують CSS, HTML, Ajax, та JavaScript.

Існує дуже багато ознак, за якими спеціалісти можуть класифікувати веб-сайти. Класифікація веб-сайту займає важливу роль у його створенні, вона є потрібною для того, щоб людина, яка створює веб-сайт розуміла, що від неї потребує замовник. Отже, були підкреслені основні ознаки кожного типу.

Сайти можна поділити на некомерційні та комерційні. Основна з цілей комерційного сайту принести власнику прибуток, а некомерційного виконати благодійну функцію в наданні певної інформації без потреби відповідної оплати.

Основні види веб-сайтів:

- веб-сайт електронної комерції - це сайт на якому люди можуть безпосередньо купувати товари. Будь-який веб-сайт, що включає кошик для

покупок та спосіб надання інформації про кредитну картку для здійснення покупки підпадає під цю категорію;

- бізнес-сайт - це будь-який веб-сайт, призначений для представлення певного бізнесу. Він має містити логотип, позиціонування та надавати інформацію про типи послуг або продукти, які вони надають;
- розважальний веб-сайт - це сайти які люди відвідують виключно з метою розваги. Більшість таких веб-сайтів мають на меті заробляти гроші, але не з користувачів, а з реклами яку розміщують на сторінках свого сайту;
- портфоліо веб-сайт - це сайт, присвячений демонстрації прикладів минулих робіт автора. Власник такого сайту хоче показати своїм потенційним клієнтам якість роботи, яку він може надати, демонструючи найкращі зразки виконання праці;
- медіа-сайт - це сайт який збирає в купу новини чи певні репортажі. Багато медіа-веб-сайтів є мережевою гілкою засобів масової інформації, яка часто існує в інших формах, таких як телевізійні канали, друковані журнали та газети, але деякі з них є тільки в інтернеті;
- веб-сайт брошура - це спрощена форма бізнес-сайту, яка може складатись лише з однієї сторінки;
- навчальний веб-сайт - це веб-сайт навчального закладу, або тих, хто пропонує онлайн-курси;
- некомерційний веб-сайт - це сайт який створює некомерційна організація, для просування своїх ідей та залучення людей;
- персональний веб-сайт - це сайт конкретної людина на якому вона може ділитись з іншими особистими думками, використовуючи відео, текст та фото;
- веб-портал - це часто веб-сайти, розроблені для внутрішніх цілей у бізнесі, організації чи установі. Вони збирають інформацію в різних форматах з різних джерел в одному місці, щоб зробити всю відповідну інформацію доступною для людей, яким потрібно її бачити;
- wiki-сайт - довідковий ресурс, який поповнюється самими користувачами.

Різновиди веб-систем:

- static Web Applications - це статичні веб-системи на яких користувачі можуть лише читати інформацію;
- dynamic Web Application - це динамічні веб-системи які генерують сторінки або дані в режимі реального часу відповідно до запиту користувача;
- single Page Apps (SPA) - це такі веб-системи, що складаються з однієї сторінки. Вони плавні та швидкі, їх легко розробляти та налагоджувати. Користувачам не потрібно переходити на різні сторінки, весь вміст завантажується на одну сторінку;
- multi-Page Apps (MPA) - це традиційні веб-програми, які перезавантажують всю сторінку та відображають нову, коли користувач переходить на іншу сторінку;
- portal Web App - портали дозволяють підприємствам мати персоналізовані інтерфейси, що відповідають потребам їхніх користувачів;
- animated Web Applications - анімація у веб-програмах корисна для того, щоб довше утримувати увагу людей та виділяти відповідну інформацію;
- web Applications with a Content Management - система управління вмістом (CMS) - це програмне забезпечення, яке допомагає користувачам створювати, керувати та модифікувати вміст на веб-сайті без необхідності технічних знань, веб-програм або мов розмітки;
- rich Internet Apps (RIA) - це веб-програма, яка має всі функції традиційних браузерних програм, але швидша, цікавіша та має кращий обмін даними;
- javaScript-Powered Web Apps - веб-програми на базі JavaScript використовується для створення динамічних та інтерактивних веб-сторінок;
- progressive Web Apps (PWA) - це програма, яка працює в будь-якому браузері чи мобільному пристрої, і має зовнішній вигляд програми на мобільному пристрої.

1.2 Тестування веб-систем

Тестування програмного забезпечення - це перевірка відповідності між реальною та очікуваною поведінкою системи, яка відбувається за допомогою деяких тестів, які вибираються певним чином. В більш широкому сенсі, тестування - це одна з технік контролю якості.

Тестування можна поділити на основні етапи [10]:

- аналіз продукта;
- робота з вимогами;
- розробка стратегії тестування;
- створення тестової документації;
- тестування прототипу;
- основне тестування;
- стабілізація;
- експлуатація.

Тестування можна поділити на мануальне та автоматизоване [7].

При Мануальному тестуванні (manualtesting) тестувальник виконує тести своїми руками, не використовуючи засоби автоматизації. Мануальне тестування - самий низькорівневий та простий тип тестування, який не потребує великої кількості додаткових знань.

Автоматизоване тестування передбачає використання спеціального програмного забезпечення для контроля якості виконання тестів та порівняння очікуваного та фактичного результату роботи програми. Цей тип тестування допомагає автоматизувати задачі, які часто повторюються.

Існує декілька типів автоматизованого тестування:

- автоматизація тестування коду (Code-driven testing) - тестування на рівні програмних модулів, класів та бібліотек;
- автоматизація тестування графічно користувацького інтерфейса (Graphical user interface testing) - спеціальна програма, яка може генерувати користувацькі події та відстежувати реакцію програми на ці дії;

- автоматизація тестування API (ApplicationProgrammingInterface) - тестування програмного інтерфейсу програми.

Веб-тестування - це практика тестування програмного забезпечення для тестування веб-сайтів та веб-додатків на наявність потенційних помилок. Це повне тестування веб-додатків перед тим, як опублікувати. Веб-систему потрібно перевірити повністю від початку до кінця, перш ніж вона з'явиться для кінцевих користувачів. Проводячи тестування веб-сайтів, організація може переконатися, що веб-система працює належним чином і може бути прийнята користувачами в режимі реального часу. Дизайн та функціональність інтерфейсу є головними елементами тестування веб-сайтів.

Основні пункти веб-тестування:

- тестування функціональності;
- тестування юзабіліті;
- тестування інтерфейсу;
- тестування на сумісність;
- тестування продуктивності;
- тестування безпеки.

Алгоритм - це процедура або формування вирішення проблеми, заснована на проведенні послідовності зазначених дій. Кожна людина має справу з алгоритмом в кожній сфері свого життя. Комп'ютерну програму можна розглядати як складний алгоритм. У математиці та інформатиці під алгоритмом зазвичай розуміють невелику процедуру, яка вирішує періодичну задачу.

Алгоритми широко використовуються в усіх сферах IT. Наприклад, алгоритм пошукового рядка в Інтернеті, який бере вхідні дані, а саме рядки пошуку ключових слів, здійснює пошук у відповідній базі даних та повертає результат у вигляді веб-сторінок.

Види алгоритмів:

- лінійний алгоритм - це послідовне виконання інструкцій в строгій черговості їх розташування;
- розгалуження - послідовність дій відповідно до визначених умов;

- циклічний алгоритм - це послідовність дій, яку необхідно повторювати кілька разів для досягнення позитивного результату.

1.3 Сервіси тестування веб-додатків

Аналізуючи вже існуючі сервіси тестування веб-додатків було зроблено висновок, що всі вони спеціалізуються на вузькому профілі тестування. Тому для для розробки нашого алгоритму автоматизованого тестування було прийнято рішення створити більш широкі можливості для тестування в одному алгоритмі. Далі розглянемо декілька вже існуючих сервісів тестування веб-додатків.

PageSpeed Insights API - це сервіс, який дає можливість отримувати звіти про швидкість завантаження сторінки на комп’ютерах та мобільних пристроях, а також надає користі поради, як покращити швидкість.

PSI надає дані як швидко сторінка завантажувалась в реальних користувачів, а також дані отримані в результаті імітації процесу завантаження сторінки. Під час імітації процес виконується в керуючих умовах тому знаходження та усунення проблеми відбувається швидше. З іншої сторони дані від стеження за користувачами відображають реальний стан дій.

HCL AppScan - це потужний інструмент, який виявляє та захищає проти вразливостей та витоку даних. Включає в себе такі види статичного аналізу, як white box та black box. Він надає розширений звіт з помилками, за допомогою якого можна з легкістю зрозуміти, які виникли помилки в безпеці.

Firebase Test Lab - це хмарна інфраструктура для тестування додатків. За допомогою однієї операції ви можете протестувати свій додаток на операційній системі Android чи iOS на різноманітних пристроях з різною конфігурацією та побачити результати, які включають в себе журнали, відео та фото екранів в консолі Firebase. Test Lab використовує реальні робочі пристрої, які працюють в центрі обробки даних Google, для тестування вашого додатку.

Cucumber - інструмент автоматизованого тестування програмного забезпечення з відкритим кодом. Дає можливість створювати тести спільно з іншими людьми та базується на тестуванні поведінки веб-систем. Також має можливість створювати документи зі специфікаціями, тобто після написання

тестів розробником сервіс створює документацію для проекта. Ця система підтримує мови програмування Groovy, Python, PHP, Perl, Scala та C# [11].

1.4 Постановка задачі

Метою роботи є розробка алгоритму автоматизованого тестування який буде використаний для тестування веб-ресурсів. Потенційні користувачі повинні мати можливість використовувати алгоритм автоматизованого тестування для своїх веб-ресурсів.

Також потрібно реалізувати базу даних, побудувати веб-систему, налаштувати та запрограмувати її роботу. За допомогою алгоритму автоматизованого тестування виконати тестування створеної веб-системи та отримати результати. Проаналізувати отримані результати та запропонувати методи їх вирішення.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Обґрунтування вибору методу рішення завдання

Перед автором роботи була поставлена задача: розробити алгоритм автоматизованого тестування, який надасть можливість тестувати веб-ресурси автоматизовано без використання додаткових ресурсів. На думку керівника диплому, алгоритм автоматизованого тестування, що розробляється, повинен мати наступні особливості:

- повинен тестувати користувацьку сторону та серверну;
- повинен відображати помилки в зрозумілому вигляді для користувачів;
- мати змогу виконувати перевірку даних в БД;
- мати можливість змінювати його за потреби користувачів для індивідуальних потреб;
- повине бути швидкий;
- має бути незалежним від фреймворка веб-ресурса;
- повинен бути зрозумілим для програмістів та тестувальників;
- при використанні мінімізовані дії для його запуску.

Враховуючи всі особливості, автором було обрано найбільш відповідні до завдань засоби, а саме для розробки веб-системи використати мови програмування: HTML, CSS разом з фреймворком Yii2. Для створення алгоритму автоматизованого тестування було обрано мову програмування JavaScript. Саме ці засоби допоможуть розв’язати поставлену вище задачу. Завдяки гарній та докладній документації на етапі створення веб-системи та алгоритму автоматизованого тестування не повинно виникнути складнощів.

Етапи створення веб-системи та алгоритму автоматизованого тестування для неї:

- затвердження технічного завдання для створення веб-системи та алгоритму;
- розробка шаблону веб-системи та створення схеми алгоритму;
- створення дизайну за шаблоном;

- розробка програмного коду веб-системи та структури бази даних;
- створення алгоритму автоматизованого тестування;
- тестування веб-системи за допомогою алгоритму.
- розміщення сайту в мережі Інтернет.

2.2 Огляд мови HTML, CSS та PHP

HTML (мова розмітки гіпертексту) та CSS (каскадні таблиці стилів) - дві основні технології побудови веб-сторінок. HTML забезпечує структуру сторінки, CSS - візуальний макет для різних пристроїв. Поряд з графікою та сценаріями, HTML та CSS є основою побудови веб-сторінок. Як правило, HTML завжди представляє зміст, а CSS - зовнішній вигляд цього змісту.

HTML - це мова для опису структури веб-сторінок.

Мова надає авторам можливість:

- створювати в Інтернеті сторінки з заголовками, текстом, таблицями, списками, фотографіями тощо;
- здійснювати переходи між сторінками, натискаючи на них;
- створювати форми для проведення операцій з віддаленими послугами, а також для пошуку інформації в Інтернеті;
- включати до веб-сторінки різноманітні відео та аудіо матеріали;
- використовувати цю мову для різноманітних веб-браузерів однаково.

HTML визначає вміст кожної веб-сторінки в Інтернеті. Позначивши ваш необроблений вміст тегами HTML, ви можете повідомити веб-браузеру, як ви хочете, щоб відображалися різні частини вашого вмісту [12]. Створення документа HTML з правильно розміченим вмістом - це перший крок до створення веб-сторінки.

Мова розмітки HTML має дуже велику популярність у всьому світі. Сьогодні це єдина мова, що допомагає створювати розмітку веб-сторінок.

Файли HTML мають розширення .html або .htm. Для написання простого коду мовою HTML достатньо програми Блокнот, яка є майже на всіх комп’ютерах.

HTML стрімко розвивається, з цим розвитком створюється нові можливості цієї мови, які надають змогу оптимізувати та покращити кінцевий результат роботи.

CSS - це мова для опису зовнішнього вигляду веб-сторінок, включаючи макети, кольори та шрифти. Це дозволяє адаптовувати веб-сторінку до різного розміру на різних пристроях. Відокремлення CSS від HTML робить простішим ведення веб-сайтів, обмін таблицями стилів між сторінками та адаптацію сторінок до різних середовищ. CSS привносить стиль у ваші веб-сторінки, взаємодіючи з елементами HTML. Подібно до HTML, CSS пишеться текстом через текстовий редактор.

Існує три основних способи додати CSS- код до HTML-сторінки [5]. Таблиці стилів можуть бути внутрішніми, зовнішніми та вбудованими. Зовнішні таблиці стилів зберігаються у форматі .css-файлів та можуть використовуватися для всього веб-сайту за допомогою лише одного файлу. Щоб користуватися зовнішньою таблицею стилів, файли .html мають містити в собі розділ заголовка, який посилається на зовнішню таблицю стилів та мають приблизно такий вигляд:

```
<head>
    <link rel = ”styleheet” type = ”text / css” href = sitestyle.css ”>
</head>
```

Саме це зв’яже файл .html до зовнішньої таблиці стилів, та всі сценарії CSS у цьому файлі будуть застосовуватися до зв’язаних сторінок .html. Внутрішні таблиці стилів - це інструкції CSS, які записані в заголовку деякої сторінки .html та має вигляд:

```
<head>
    <style>
        body { background-color:....; }
        p {   font-size:...;
              color:...; }
    </style>
</head>
```

Вбудовані стилі - це фрагмент CSS, який записаний в HTML-код та застосовується лише до одного екземпляра кодування. Наприклад:

<h1 style = ”font-size: 15px; color: red;”> Заголовок! </h1>

Зовнішні таблиці стилів є найбільш ефективним методом реалізації CSS на веб-сайті.

PHP (Hypertext Preprocessor) - це мова програмування, а саме сценаріїв, яка в основному використовується у веб-розробці на стороні сервера [6]. Мови сценаріїв - це підмножина мов кодування, що використовується для автоматизації процесів, які в другому випадку мали б виконуватися покроково в коді сайту кожен раз, коли вони б виникали. Сценарії PHP можна інтерпретувати лише на сервері, на якому встановлений PHP. Комп’ютери користувачів, які мають доступ до скриптів PHP, потребують лише веб-браузера. PHP-файл містить теги PHP та має розширення .php. PHP-код може бути вбудований у HTML-код, або його можна використовувати в поєднанні з різними системами веб-шаблонів, системою управління веб-вмістом та веб-фреймворками.

Переваги PHP [8,9]:

- безкоштовний та відкритий;
- більшість серверів веб-хостингу підтримують PHP;
- регулярно оновлюється з розвитком останніх технологічних тенденцій;
- має вбудовану підтримку, яка працює з MySQL;
- крос-платформна мова.

Що стосується частки ринку, в Інтернеті існує понад 20 мільйонів веб-сайтів та додатків, які розроблені на мові сценаріїв PHP.

Основна різниця між PHP та JavaScript полягає в тому, що другий працює на стороні клієнта (у вашому браузері), тоді як перший працює на вашому сервері, генеруючи HTML, який потім надсилається клієнту.

2.3 Фреймворк Yii2 та мова JavaScript

Фреймворки PHP полегшують розробку веб-додатків, які пишуться на PHP. Вони надають базову структуру, щоб побудувати веб-програму. Завдяки фреймворкам ви зможете економити час, створювати більш стабільні програми

та зменшити кількість повторення в коді. Загальна ідея роботи PHP-фреймворка називається Model View Controller (MVC). MVC - це архітектурний шаблон у програмуванні, який ізолює ділову логіку від інтерфейсу користувача, при цьому дозволяючи змінюваати один окремо від іншого.

Переваги використання фреймворків:

- код є більш безпечним;
- можна уникнути повторного коду;
- послідовна розробка коду із меншою кількістю помилок;
- полегшує роботу над складними технологіями;
- деякі сегменти коду та функціоналу попередньо створені та протестовані, що робить додатки більш надійними;
- тестування та налагодження коду простіше;
- час для розробки програми скорочується.

Yii2 - це високопродуктивний компонентний PHP-фреймворк для швидкого розвитку сучасних веб-додатків. Назва Yii з китайської мови означає “простота та еволюція” [1]. Цей фреймворк являється загальним, це означає, що його використовують для розробки всіх видів веб-додатків за допомогою PHP. Завдяки архітектурі на основі компонентів та дуже гарній підтримці кешування цей фреймворк підходить для великомасштабних додатків, такі як форуми, портали, проекти електронної комерції та інші.

JavaScript - це крос-платформна, об’єктно-орієнтована мова сценаріїв. Її використовують для створення веб-сторінок інтерактивними. Існують також більш вдосконалені версії JavaScript на стороні сервера, такі як Note.js, які дозволяють додати веб-сайту більше функціональних можливостей.

Основний JavaScript можна розширити для різних цілей, доповнивши його додатковими об'єктами. Наприклад:

- клієнтський JavaScript розширює основну мову, надаючи об’єкти для управління браузером та його об’єктною моделлю документа (DOM);
- серверний JavaScript розширює основну мову, надаючи об’єкти, що мають відношення до роботи JavaScript на сервері.

Коли JavaScript створили, його назвали “LiveScript”. На той час була дуже популярна мова програмування Java, тому вирішили, що позиціонування нової мови як “молодшого брата” Java допоможе. З часом та розвитком JavaScript став повністю незалежною мовою зі своєю специфікацією, і тепер він взагалі не має ніякого відношення до Java.

JavaScript має такі переваги:

- повна інтеграція з HTML та CSS;
- прості речі робляться просто;
- підтримка всіх основних браузерів.

JavaScript - єдина технологія браузера, яка поєднує ці три речі, тому це найпоширеніший інструмент для створення інтерфейсів браузера. Тим не менш, JavaScript також дозволяє створювати сервери, мобільні додатки тощо.

2.4 Додаткові технології та інструменти

База даних, також називають електронна база даних - це будь-який збір інформації та даних, яка спеціально організована задля швидкого пошуку в ній. Бази даних структуровані для полегшення пошуку, зберігання, видалення та модифікації даних в поєднанні з різними операціями з обробки даних [14]. Система управління базами даних допомагає витягнути інформацію з бази даних у відповідь на певний запит. База даних зберігається у вигляді файлів або набору файлів. Інформація в цих файлах розбита на записи, які складаються з одного або декількох полів. Поле є основною одиницею зберігання даних та містить інформацію, яка стосується одного аспекту або атрибута сутності, описаної базою даних. Щоб отримати користувачу інформацію потрібно використати запит до бази даних.

Користувачі великої бази даних повинні швидко обробляти інформацію в ній. Для підтримки саме цієї можливості було розроблено декілька типів СУБД:

- плоскі;
- ієрархічні;
- мережеві;
- реляційні;

- об’єктно-орієнтовані.

Puppeteer - це проект команди Google Chrome, який дозволяє нам керувати браузером на основі протоколу Chrome DevTools і виконувати загальні дії, подібні справжніх браузерів - програмно, за допомогою API. Це надзвичайно корисний і простий інструмент для авторизації, тестування у режимі headless.

PhpStorm - це інтегроване середовище для розробки на мові PHP, побудоване на платформі IntelliJ IDEA. За допомогою PhpStorm можна розробляти програми на версіях PHP 5.3 - 7.4 [2], а також це середовище повністю підтримує HTML, CSS, JavaScript та XML, ці мови обровляються за допомогою плагінів, які входять до складу IDE. Підтримку інших мов програмування можна додати за допомогою спеціальних плагінів.

IDE надає розумне заповнення коду, виділення синтаксису, перевірку коду на льоту, підтримку змішування декількох мов та інше. Він ідеально підходить для роботи з Symfony, Laravel, Drupal, WordPress, Zend Framework, Magento, Joomla!, CakePHP, Yii та іншими фреймворками. Перевагами цього інструмента також є підтримка віддаленого розгортання, інтеграція систем контролю версій, підтримка баз даних, інструменти командного рядка, Docker, Composer та інше.

PhpStorm працює на платформах такі, як Windows, macOS та Linux.

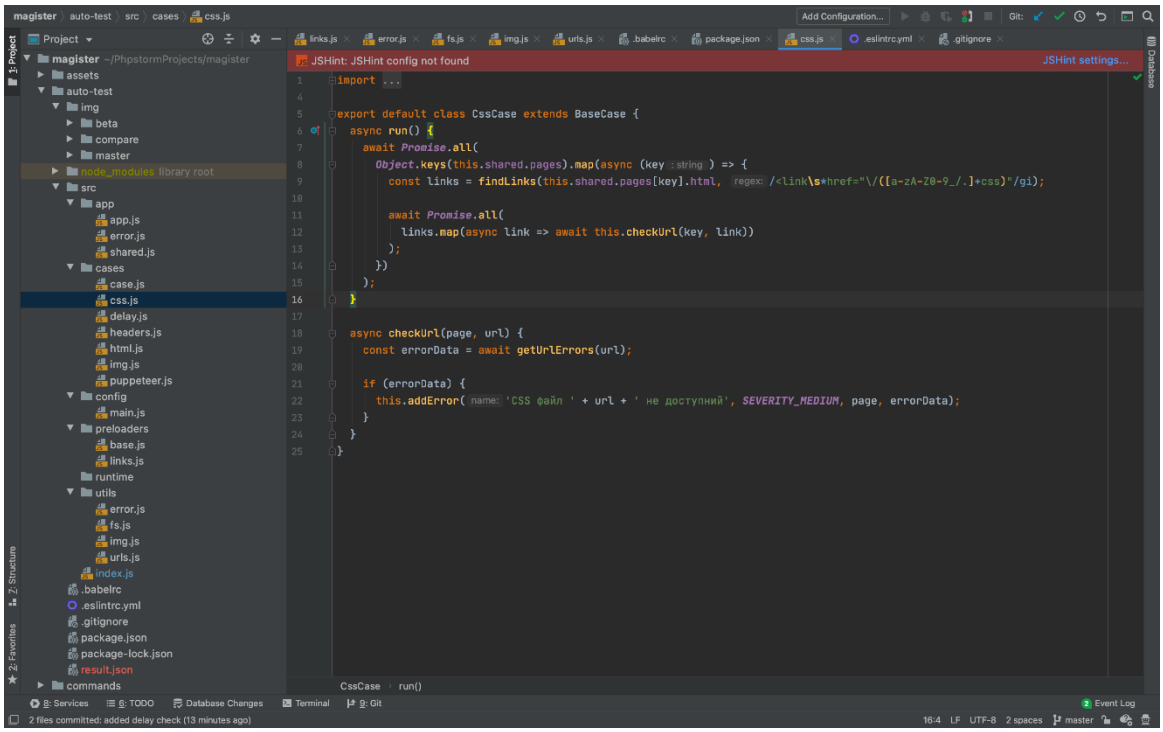


Рисунок 2.1 – Зовнішній вигляд PHPStorm

2.5 Теоретична частина реалізації програми

Щоб створити веб-систему та алгоритм автоматизованого тестування для неї, потрібно пройти по головним частинам проекту.

По перше, для правильного функціонування веб-системі потрібно використовувати базу даних. База даних використовується для постійного зберігання даних та можливості їх отримання у будь-який час. Наша веб-система не може працювати без цієї складової. У базі даних зберігається інформація про товари, які відображаються на сторінках сайту та замовлення які зроблені в цій веб-системі.

По друге для роботи веб-системи потрібен веб-сервер. У нашому випадку було використано вбудований в мову програмування PHP веб-сервер. Він надає більшість можливостей, що й інші веб-сервери, яких нам достатньо для роботи системи. Цей веб сервер обробляє всі запити пов’язані з HTML, CSS та медіа.

По третє для написання алгоритму потрібна базова структура, яка складається з компілятора, лінтера та менеджера пакетів. Компілятор обробляє код та надає можливість виконувати його, лінтер використовується для перевірки помилок при написанні мовою JavaScript та змушує писати в одному стилі та менеджер пакетів дозволяє завантажити компілятор, лінтер, бібліотеку puppeteer та інші залежності потрібні у даному проекті.

Ці всі складові використовуються для створення веб-системи та алгоритму. Веб-система будується на базі фреймворка Yii2(базової версії). Алгоритм автоматизованого тестування написаній мовою JavaScript. Цей алгоритм завантажує HTML зі сторінок та робить перевірку посилань на інші сторінки, посилань на CSS файли, перевіряє помилки у структурі HTML коду, робе фото сторінки та звіряє її з тією яка була в минулий раз за допомогою бібліотеки Puppeteer, це дає можливість перевірити чи змінилась сторінка і, що саме на ній.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Налаштування веб-сервера на базі PHP та фреймворка Yii2

Для роботи веб-системи потрібно налаштувати та встановити фреймворк Yii2. Всі запроси буде обробляти веб-сервер на базі мови PHP.

Веб-сервер на базі мови PHP є вбудованим в мову програмування PHP та може обробляти всі запити до заданного ресурса. Цей сервер може віддавати зображення, файли зі стилями та виконувати скрипти [4].

Фреймворк Yii2 складається з декількох частин:

- вхідні скрипти - завантажують конфігураційні файли та ініціалізують фреймворк;
- маршрутизатор - шукає параметри запросу та знаходить потрібний контроллер;
- базовий контроллер - створення користувацького контроллера;
- моделі - інтерфейс взаємодії з базою даних;
- вигляди - інтерфейси взаємодії з користувачами;
- допоміжні класи - спрощують створення елементів на сайті (меню, списки, поля вводу).

На рисунку 3.1 зображено структуру веб-системи на базі фреймворка Yii2 та послідовність дій при обробці запита користувача.

Щоб перейти до створення веб-системи потрібно створити новий проект в PHPSTORM, перейти в нього та запустити команду:

composer create-project --prefer-dist --stability=stable yiisoft/yii2-app-basic basic

Ця команда завантажує основну структуру веб-системи за допомогою composer, створює конфігураційні файли та відтворює структуру директорій [3].

На рисунку 3.2 зображено основну структуру директорій проекту, яка була створена попередньою командою.

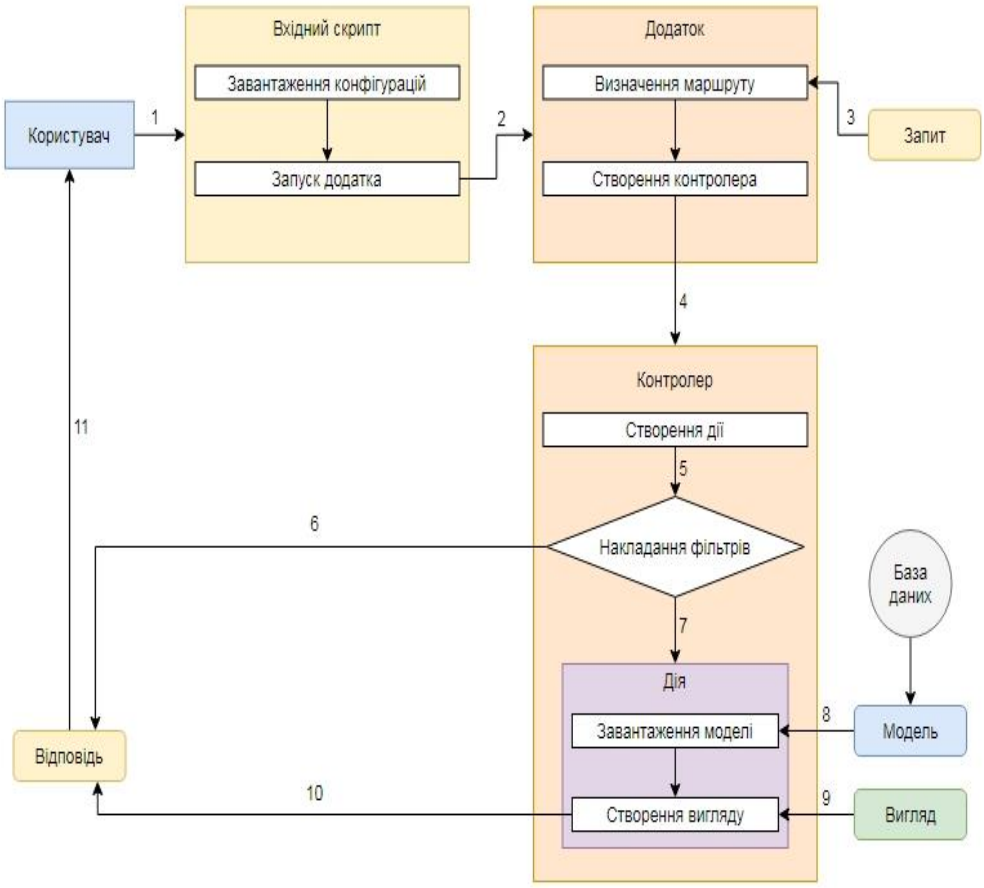


Рисунок 3.1 - Алгоритм при обробці запита користувача фреймворком Yii2

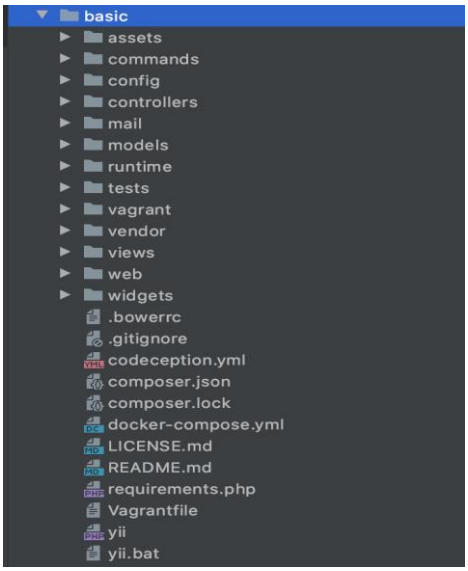


Рисунок 3.2 - Структура директорій проекта

3.2 Створення та налаштування бази даних

Було обрано для роботи базу даних MySQL. Вона наділена такими властивостями:

- швидка;
- легка у використанні;
- поширена;
- типовий SQL синтаксис;
- має драйвери до Yii2.

Для використання бази даних було взято docker образ. Docker - це програма, яка використовується для управління контейнерами ізольованими один від іншого. MySQL було завантажено у docker контейнер за допомогою конфігурації:

```
version: '3.3'
services:
```

Наступний код ініціалізує сервіс із базою даних MySQL 8 версії у контейнері та задає змінні оточення для можливості підключення до цієї бази даних з фреймворка.

```
  db:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: magister
      MYSQL_USER: root
      MYSQL_PASSWORD: root
```

Проставляємо порт 3306 назовні контейнера для можливості підключення до цієї бази даних.

```
    ports:
    - 3306:3306
volumes:
  db_data: {}
```

База даних буде створена за допомогою міграцій. Міграції це спосіб створення структури бази даних за допомогою мови програмування, вони є транзакційно безпечними та мають синтаксис незалежний від бази даних. Також за допомогою міграції легко переносити структуру бази даних між розробниками.

Типовий клас з міграцією на базі фреймворка Yii2 виглядає так:

```
class m180200_483401_create_post_table extends Migration
{
```

Ця функція створює таблицю з двома полями: ідентифікатором та заголовком.

```
   public function up()
   {
      $this->createTable('post', [
         'id' => Schema::TYPE_PK,
         'title' => Schema::TYPE_STRING
      ]);
   }
```

Ця функція видаляє таблицю при поверненні транзакції назад.

```
   public function down()
   {
      $this->dropTable('post');
   }
}
```

В роботі було створено декілька міграцій, а саме:

- міграція створення продукту;
- міграція створення категорій продуктів;
- міграція додавання характеристик продукту;
- міграція з замовленнями.

Далі наведено код міграції категорій:

```
<?php
use yii\db\Migration;
```

class m190413_112238_create_category_table extends Migration

{

Дана функція створює таблицю категорій з двома полями: ідентифікатором та назвою категорій, а таблиця буде мати назву category.

```
    public function safeUp()
    {
        $this->createTable('{{%category}}', [
            'id' => $this->primaryKey(),
            'name' => $this->string()
        ]); }
```

Дана функція видаляє попередньо створену таблицю category та всім її вмістом.

```
    public function safeDown()
    {
        $this->dropTable('{{%category}}');
    }}
```

Щоб міграції були виконані потрібно виконати команду:

php yii migrate

Після виконання міграцій була створена структура даних зображена на рисунку 3.3.

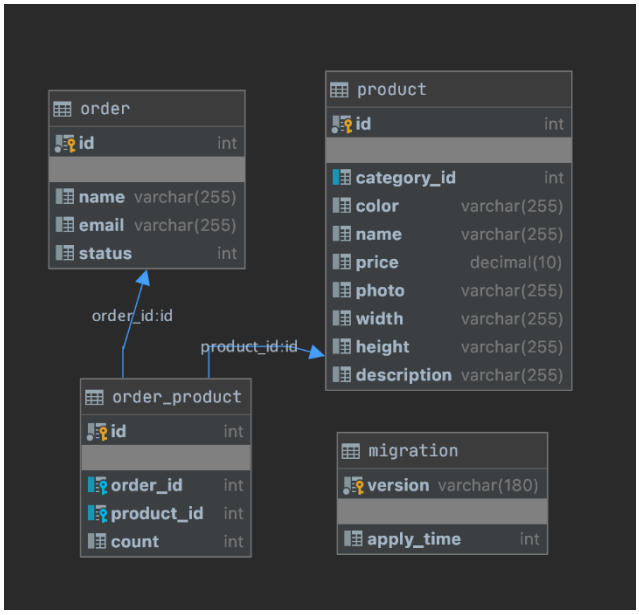


Рисунок 3.3 – Структура бази даних

3.3 Програмування веб-системи

Для побудови веб-системи потрібно написано кілька контролерів. Контролер - це клас, який відповідає за обробку запиту користувача та створення відповіді.

Контролери в нашій веб-системі:

- базовий контролер. Включає в себе головну сторінку, сторінку контактів, сторінку про нас та корзину;
- контролер з замовленнями. Має в собі методи отримання списку замовлених товарів, створення нового товару та їх редагування;
- контролер з товарами. Містить методи отримання списку, сторінки товару.

Нижче наведений приклад одного з методів контролера товарів - отримання списка:

```
public function actionIndex() {
    $searchModel = new ProductSearch();
```

Створюємо провайдер даних, який отримує параметри з запроса та фільтрує товари за декількома полями. Далі цей провайдер повертає фільтрований список товарів та за цим списком генерується HTML сторінки.

```
$dataProvider = $searchModel->search(Yii::$app->request->queryParams);
    return $this->render('index', [
      'searchModel' => $searchModel,
      'dataProvider' => $dataProvider,
    ]);}
```

Другою важливою частиною системи є моделі. Моделі - це класи, які з’єднують додаток з базою даних та можуть містити деяку логіку.

Приклад моделі з товарами в роботі:

```
class Product extends \yii\db\ActiveRecord{
  public static function tableName(){
    return 'product'; }
```

Код наданий нижче валідує поля за декількома ознаками, а саме - ціна повинна бути числом, опис - текстом та поля: ім’я та фото повинні містити текст не більше 300 символів.

```
public function rules() {
    return [
        [['description'], 'string'],
        [['price'], 'number'],
        [['name', 'photo'], 'string', 'max' => 300]];}
```

Третім кроком було створено зовнішній вигляд для сторінок за допомогою HTML, CSS та JavaScript.

Прикладом є HTML код сторінки товару:

```
<div class="col-lg-12">
    <div class="col-lg-4">
```

Код нижче додає зображення товару на сторінку.

```
        <img src="/images/<?= $product->photo ?>" alt="" />  </div>
<div class="col-lg-6">
```

Код нижче додає опис продукту до потрібної секції.

```
        <div class="product-description"><?= $product->description ?></div>
        <div class="product-configuration">
```

Наступний код відображає розміри продукта у відповідній секції.

```
            <div class="cable-config">
                <p class="testimonials">Ширина: <?= $product->width ?></p>
                <p class="testimonials">Высота: <?= $product->height ?></p>
            </div>  </div> </div></div>
```

Після реалізації веб-системи користувач має можливість переходити за посиланнями, дивитися інформацію щодо кожного продукта та інформацію про систему. Приклад однієї зі сторінок веб-системи зазначено на рисунку 3.4.

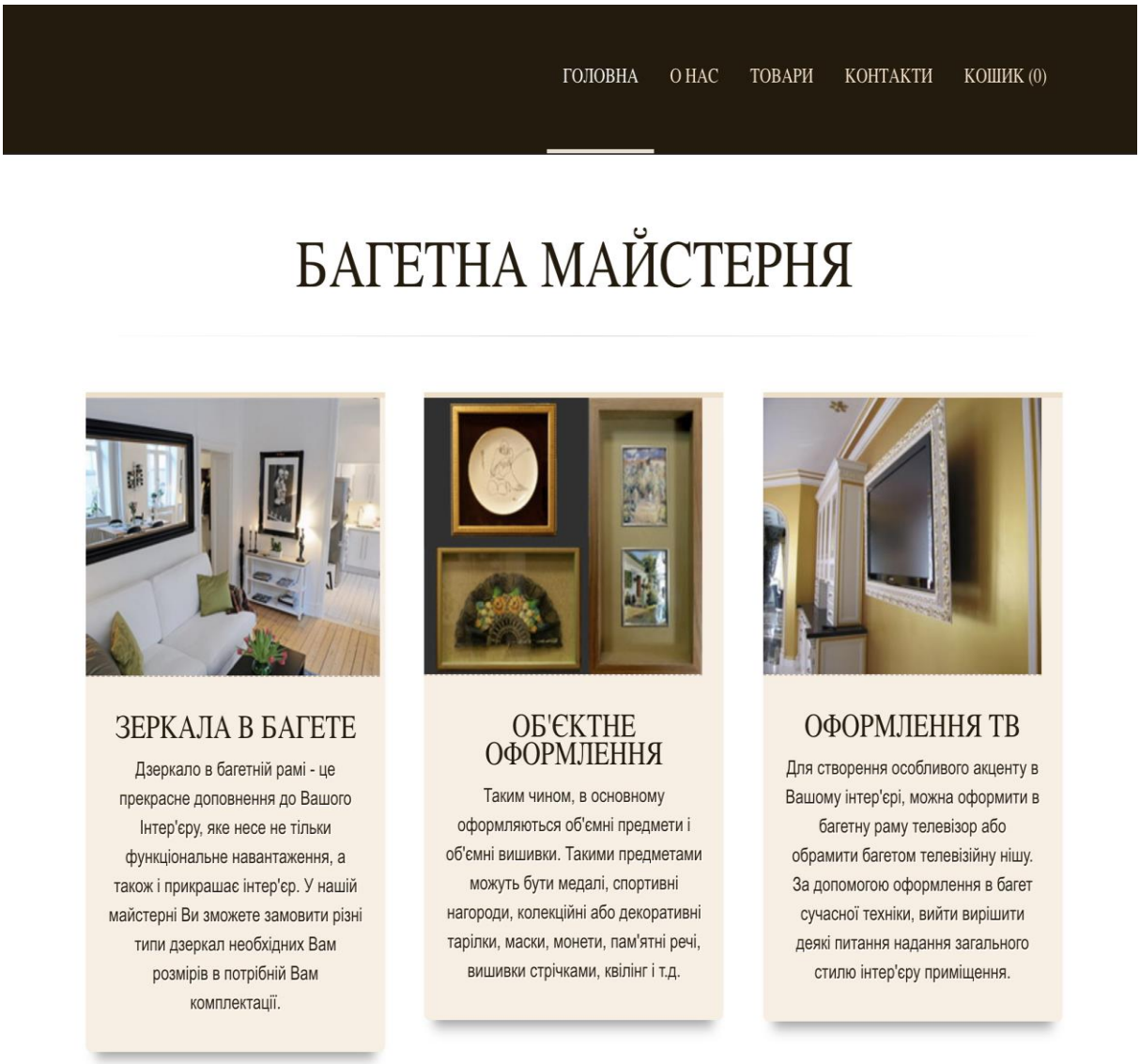


Рисунок 3.4 - Головна сторінка веб-додатку

3.4 Розробка алгоритму автоматизованого тестування

Алгоритм автоматизованого тестування був розроблений за допомогою мови програмування JavaScript. Вона дає можливість розробляти скрипти будь-якої складності, також має широке розповсюдження при створенні веб-систем будь-якої складності.

На рисунку 3.5 можна побачити як працює алгоритм, які дії він виконує, дані які використовує та кроки за якими відбувається тестування та аналіз веб-системи. Кроки перевірки є незалежними один від одного тому можуть виконуватись в іншому порядку, якщо буде така потреба у програміста.

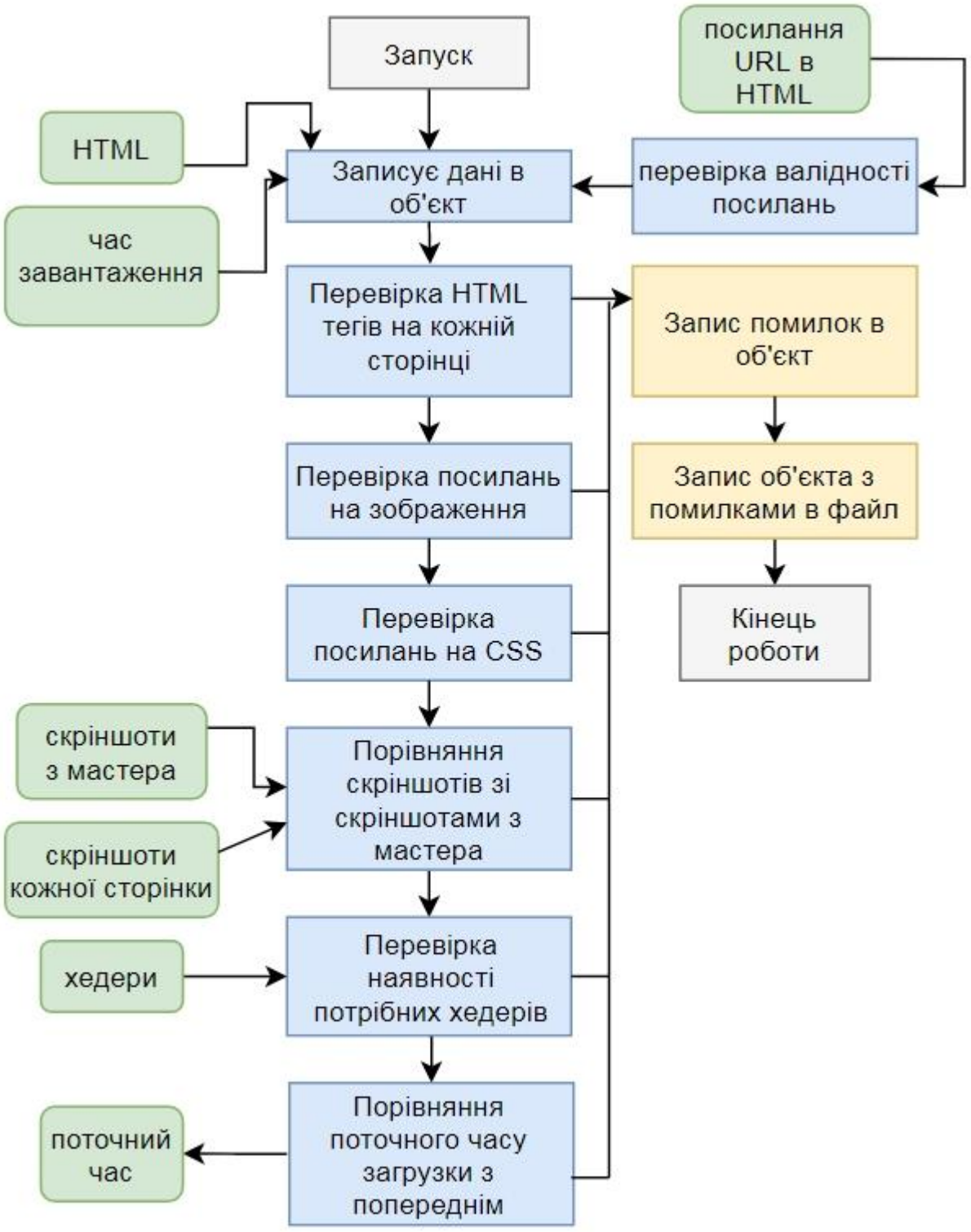


Рисунок 3.5 - Схема роботи алгоритму

Для запуску алгоритму потрібно перейти в консолі до папки з алгоритмом та виконати наступну команду:

npm run babel-node src/index.js

Ця команда запускає роботу нашого алгоритму. Вона компілює весь JavaScript код та запускає скрипт з назвою index.js.

З файлу index.js завантажується app - це головний клас який контролює роботу всього алгоритму та виконує крок за кроком всі потрібні дії.

```
export default class App {
  constructor() {
    this.shared = new Shared();
```

Список класів , які завантажують потрібні дані.

```
    this.preloaders = [LinksPreloader,];
```

Список класів, які перевіряють поведінку веб-додатку. Ці класи включають: перевірку HTML, посилань на зображення та CSS, створюють та порівнюють скріншоти, роблять перевірку хедерів та затримку кожної зі сторінок.

```
    this.cases = [
      HtmlCase,
      ImgCase,
      CssCase,
      PuppeteerCase,
      HeadersCase,
      DelayCase,];}
  run = async () => {
```

Наступний код запускає всі прелоадери - це класи, які завантажують дані попередньо, які потрібні для роботи.

```
    await this.runPreload();
    await this.runCases();
    await this.writeToFile();}
```

Після запуску алгоритм обробляє конфігураційний файл та отримує маршрут за яким потрібно брати вхідні дані для тестування. Нижче наведений цей файл:

```
export default {
  url: 'http://localhost:8001',
```

```
  link_depth: 3,
  time_max_delay_multiplier: 3,};
```

Далі за посиланням на веб-ресурс алгоритм завантажує код HTML сторінки та посилання на інші сторінки. Далі він за кожним з посилань рекурсивно переходить збирає ті ж дані, які були зібрані для першої сторінки і так рекурсивно до параметра вказаного в конфігурації. Також для кожної зі сторінок вимірюється час її завантаження.

```
const timeStart = Date.now();
    const response = await fetch(getFullUrl(url));
```

Перевіряє чи статус відповіді від сторінки не є помилковим та отримуємо код HTML у вигляді тексту.

```
    if (response.status === 200) {
      const text = await response.text();
```

Знаходимо всі посилання на сторінці за допомогою regex - коду. Цей код широко використовуються для пошуку інформації у тексті за заданими параметрами.

```
      const urls = findLinks(text, /<a\s*href="\/([a-zA-Z0-9/]+)"/gi);
```

Записуємо дані з кожної зі сторінок у об’єкт, який далі буде передано до частини перевірки.

```
      if (depth < 2 && url !== baseUrl) {
        this.shared.pages[url] = {
          links: urls,
          html: text,
```

Наступний код отримує час за який було завантажено веб-сторінку.

```
          time: Date.now() - timeStart,};}
      await Promise.all(
        urls.map(async childUrl => {
```

Даний код робить рекурсивну перевірку кожного з посилань - перевіряє доступність посилання, знаходить інші посилання в HTML та вимірює час завантаження.

```
          await this.checkUrl(childUrl, url,depth + 1);}) ) }
```

Наступним кроком є перевірка тегів на кожной HTML сторінці. Під час перевірки, якщо виникає помилка вона записується у відповідний об’єкт з помилками. Для перевірки беруться такі теги:

- div
- p
- h1, h2, h3, h4, h5, h6
- head
- body
- header
- article

Згідно з документацією HTML наведені вище теги повинні бути обов’язково закриті.

```
tags.forEach(tag => {
    const openTagLength = html.split('<' + tag + '>').length - 1;
    const closingTagLength = html.split('</' + tag + '>').length - 1;
```

Цей код підраховує кількість не закритих тегів.

```
    const notClosedSize = openTagLength - closingTagLength;
```

Якщо кількість не закритих тегів більше нуля то записуємо помилку в об’єкт з помилками.

```
    if (notClosedSize > 0) {
     this.addError('Знайдено ' + notClosedSize + ' не закритих тегів ' + tag, SEVERITY_MEDIUM, url); }});
```

Якщо не має тегу !DOCTYPE, записуємо помилку, що цей тег відсутній.

```
if (!html.includes('<!DOCTYPE html>')) {
    this.addError('Doctype повинен міститись у документі', SEVERITY_HIGH, url);}
```

Після перевірки HTML тегів, алгоритм перевіряє посилання на зображення. Наступний код, на кожній сторінці робить перевірку на доступність посилань на кожне із зображень.

```
  await Promise.all(
```

```
Object.keys(this.shared.pages).map(async (key) => {
```

Код нижче знаходить всі посилання з типом .jpg та .png в HTML коді кожної зі сторінок.

```
const links = findLinks(this.shared.pages[key].html, /<img\s*src="\/([a-zA-Z0-9_/.]+(jpg|png))"/gi);
```

Перевірка кожного з посилань та записування помилок, якщо посилання не доступне.

```
await Promise.all(
  links.map(async link => await this.checkUrl(key, link))
); }) );
```

Наступник кроком відбувається перевірка посилань в HTML коді на файли CSS. За допомогою regex знаходимо та записуємо в масив links всі посилання на файли.

```
const links = findLinks(this.shared.pages[key].html, /<link\s*href="\/([a-zA-Z0-9_/.]+css)"/gi);
```

Перевіряємо кожне з посилань за допомогою функції checkUrl.

```
await Promise.all(
  links.map(async link => await this.checkUrl(key, link))
);
```

Ця функція знаходить помилки після запиту до посилання та додає їх до списку всіх помилок.

```
async checkUrl(page, url) {
  const errorData = await getUrlErrors(url);
if (errorData) {
   this.addError('CSS файл ' + url + ' не доступний', SEVERITY_MEDIUM, page, errorData);}}
```

Наступний код створює зображення для кожної зі сторінок та порівнює їх із зображеннями створеними попередньо. За допомогою бібліотеки Puppeteer - це інструмент, який дозволяє оперувати більшістю інструментарія браузера Chromium та створювати за допомогою скриптів сценарії кастомной поведінки.

```
async init() {
```

Створюємо екземпляр об’єкта Puppeteer та задаємо, що браузер буде керуватися скриптами.

```
this.puppeteer = await Puppeteer.launch({
  headless: true,
});
```

Створюємо необхідні дерикторії для майбутнього запису зображень та їх порівняння.

```
this.isEmptyMaster = await isEmptyDir('img/master');
await createDir('img');
await createDir('img/master');
await createDir('img/beta');
await createDir('img/compare');}
```

Наступна функція завантажує кожну зі сторінок та робить скріншот на цій сторінці. Задля збільшення швидкості кожна зі сторінок відкривається у окремій вкладці.

```
async loadPhotos() {
await Promise.all(
  Object.keys(this.shared.pages).map(async url => {
```

Код нижче створює нову сторінку та задає її розміри, як 1920 за шириною та 1080 за висотою.

```
    const page = await this.puppeteer.newPage();
    await page.setViewport({width: 1920,height: 1080,});
```

Далі переходимо на потрібну сторінку, робимо скріншот цієї сторінки та закриваємо вкладку.

```
    await this.go(url, page);
    await this.screenShot(url, page);
    await page.close(); }));}
```

Ця функція порівнює зображення створені попередньо та створені у даний час.

```
async comparePhotos() {
```

Знаходимо всі файли, що розташовані у директорії img/beta.

```
const files = await getFiles('img/beta');
await Promise.all(
 files.map(async file => {try {
    await compareImages('img/beta/' + file, 'img/master/' + file);
  } catch (data) {
```

Записуємо помилку, що файли не збігаються.

```
    const url = file.replaceAll('_', '/').replace('.png', '');
    this.addError('Зображення при порівнянні з мастером для посилання ' + url + ' має розбіжності',
      SEVERITY_HIGH, url, data);
  }}));}
```

Наступний блок перевіряє наявність хедерів у запиті до головної сторінки веб-системи. Ці хедери відповідають здебільшо за безпеку та задають поведінку браузеру.

```
for (const [name, value] of Object.entries(headersConfig)) {
 const headerFromResponse = headers.get(name);
```

Код нижче перевіряє чи відсутній хедер, який заданий у конфігу та додає помилку про це.

```
 if (!headerFromResponse) {
  this.addError(`Відсутній ${name} хедер який може призвести до погіршення безпеки`, SEVERITY_HIGH);}
```

Наступний код звіряє очікуване значення хедера з отриманим та записує помилку, якщо вони різні.

```
 if (value !== null && headerFromResponse !== value) {
  this.addError(`Значення хедера ${name}(${headerFromResponse}) не співпадає з рекомендованим ${value}`, SEVERITY_LOW);}
```

Останньою перевіркою алгоритма є перевірка кількості часу завантаження сторінки. Код нижче знаходить файл з попередніми результатами.

```
const lastResults = await this.getLastResults();
```

```
    for (const [url, {time}] of Object.entries(this.shared.pages)) {
      const lastTime = lastResults[url];
```

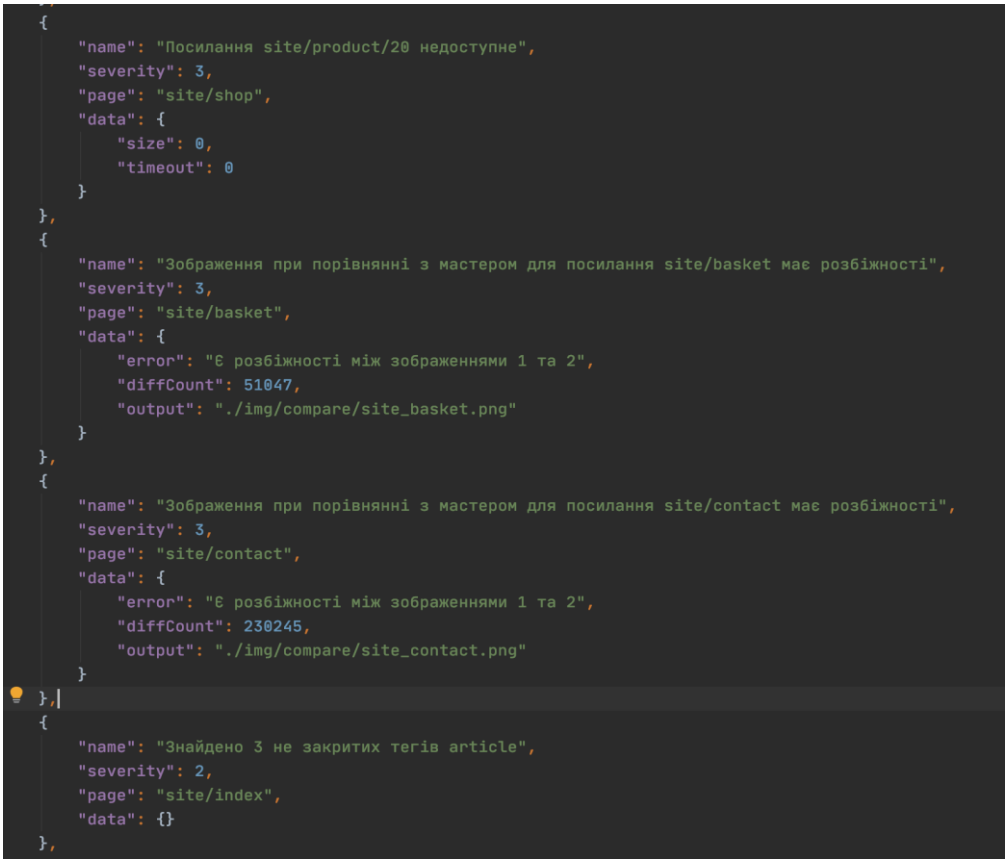
Наступний код порівнює поточний час роботи з часом який був попередньо записаний, якщо різниця мі ними більша ніч число задане в конфігураційному файлі додаємо помилку.

```
      if (time / config.time_max_delay_multiplier > lastTime) {
        this.addError(`Сторінка ${url} стала завантажуватись набагато більше часу!`,
SEVERITY_MEDIUM, url, {lastTime,time,});}}
```

Після всіх дій, які були виконані сортуємо список помилок за важливістю та записуємо об’єкт з помилками у файл result.json.

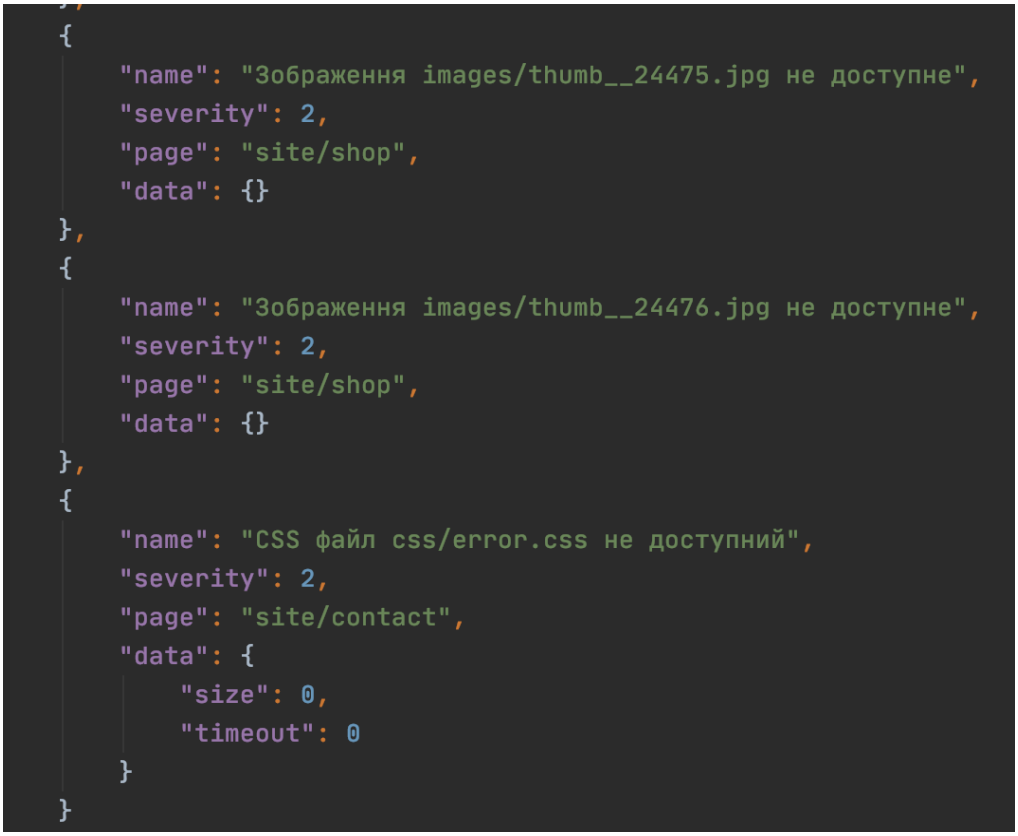
```
  writeToFile = async () => {
    try {
      const errors = this.shared.errors.sort((a, b) => b.severity - a.severity);
      await fs.writeFile('./result.json', JSON.stringify(errors, null, 4));
    } catch (e) {console.log(e);
```

На рисунках 3.6-3.8 зображений результат після тестування.



```
{
    "name": "Посилання site/product/20 недоступне",
    "severity": 3,
    "page": "site/shop",
    "data": {
        "size": 0,
        "timeout": 0
    }
},
{
    "name": "Зображення при порівнянні з мастером для посилання site/basket має розбіжності",
    "severity": 3,
    "page": "site/basket",
    "data": {
        "error": "Є розбіжності між зображеннями 1 та 2",
        "diffCount": 51047,
        "output": "./img/compare/site_basket.png"
    }
},
{
    "name": "Зображення при порівнянні з мастером для посилання site/contact має розбіжності",
    "severity": 3,
    "page": "site/contact",
    "data": {
        "error": "Є розбіжності між зображеннями 1 та 2",
        "diffCount": 230245,
        "output": "./img/compare/site_contact.png"
    }
},
{
    "name": "Знайдено 3 не закритих тегів article",
    "severity": 2,
    "page": "site/index",
    "data": {}
},
```


Рисунок 3.6 - Результат тестування



```
{
    "name": "Зображення images/thumb__24475.jpg не доступне",
    "severity": 2,
    "page": "site/shop",
    "data": {}
},
{
    "name": "Зображення images/thumb__24476.jpg не доступне",
    "severity": 2,
    "page": "site/shop",
    "data": {}
},
{
    "name": "CSS файл css/error.css не доступний",
    "severity": 2,
    "page": "site/contact",
    "data": {
        "size": 0,
        "timeout": 0
    }
}
```


Рисунок 3.7 - Результат тестування

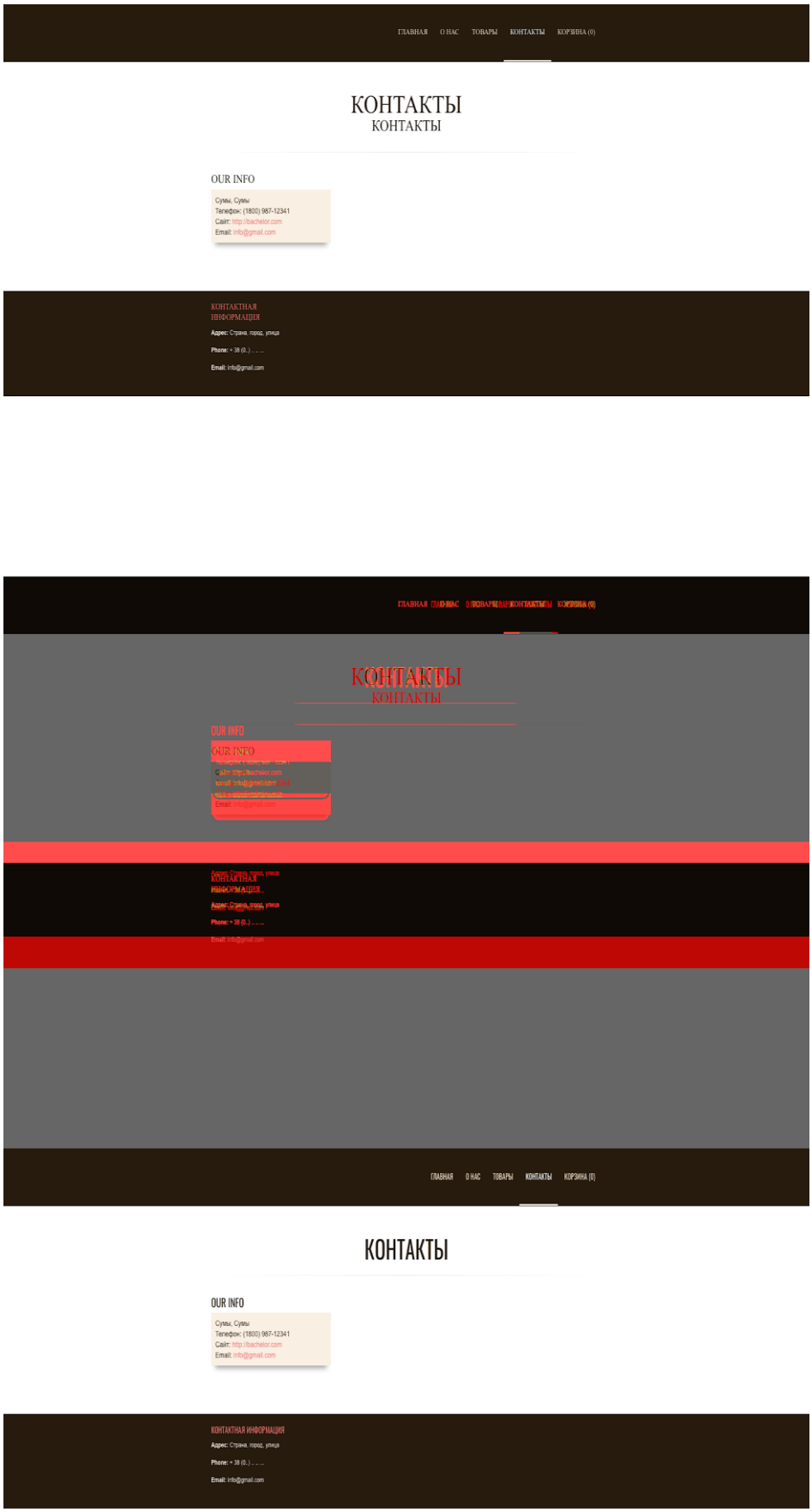


Рисунок 3.8 - Результат тестування

Аналізуючи отримані результати роботи алгоритму автоматизованого тестування, була створена таблиця 3.1, в якій описано кожну з груп помилок та кількість помилок для відповідної групи.

Назва групи помилок	Кількість помилок	Середній пріорітет групи	Кроки для усунення помилок
Посилання на сторінку недоступне	12	3	Потрібно додати відповідні методи до контролерів, чи перевірити наявність інформації в базі даних.
Посилання на файл CSS недоступне	1	2	Потрібно додати відповідні файли до проекту або вилучити посилання на файли CSS.
Посилання на зображення недоступне	7	2	Потрібно додати зображення в відповідне місце або змінити маршрут посилання.
HTML структура не валідна	2	2	Потрібно додати відповідні теги, які відсутні.
Зміна зовнішнього вигляду сайту	2	3	Потрібно змінити стилі та HTML до відповідних або нові зміни записати до мастера.
Відсутність потрібних хедерів	2	3	Потрібно додати необхідні хедери безпеки.
Час завантаження сторінки перебільшує заданий час	0	0	Потрібно перевірити, які зміли призвели до погіршення часу завантаження сторінки.

Таблиця 3.1 – Аналіз результів тестування веб-системи

ВИСНОВКИ

В результаті виконаної роботи був оброблений досить великий обсяг інформації з самих різних джерел. Завдяки отриманій інформації були обрані методи для реалізації веб-системи та алгоритму автоматизованого тестування. Було створено веб-систему на базі фреймворка Yii2, побудовано базу даних та розроблено алгоритм автоматизованого тестування.

Алгоритм автоматизованого тестування включає в себе перевірку таких частин системи, як посилання на інші сторінки, зображення та каскадні стилі, структуру HTML, зовнішній вигляд кожної зі сторінок та час їх завантаження. Алгоритм написано на мові програмування JavaScript. Після реалізації алгоритму було протестовано попередньо створену веб-систему та отримано результати. За відповідними результати було проведено аналіз та запропоновано методи їх вирішення.

Веб-система та алгоритм є готовими для використання.

СПИСОК ЛІТЕРАТУРИ

1. Офіційний сайт документації фреймворку Yii2 - Режим доступу: https://yiiframework.com.ua/uk/doc/guide/2/
2. PHPStorm - Режим доступу: https://www.jetbrains.com/phpstorm/
3. Встановлення Yii2 - Режим доступу: https://yiiframework.com.ua/uk/doc/guide/2/start-installation/
4. PHP офіційна документація - Режим доступу: http://php.net
5. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения: 2-е издание / Б. Хоган. – Санкт-Петербург: Питер, 2014. – 320 с.
6. Маклафлин Б. PHP и MySQL. Исчерпывающее руководство: 2-е издание / Б. Маклафлин – Санкт-Петербург: Питер, 2014. – 544 с.
7. Куликов С.С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — Минск: Четыре четверти, 2017. — 312 с.
8. Зансдстра М. PHP. Объекты, шаблоны и методики программирования: учебное пособие / Зандстра М. – Москва: Вильямс, 2015. – 577 с.
9. David Skla. Learning PHP: A Gentle Introduction to the Web’s Most Popular Language / D. Skla. – O'Reilly Media, 2016. – 416c.
10. Web Application Testing: Step by Step Process to make it Right – Режим доступу: https://tms-outsource.com/blog/posts/web-application-testing/
11. The 5 Best Automation Testing Tools for Web Applications that You Could Use in 2020 – Режим доступу: https://medium.com/@OPTASY.com/the-5-best-automation-testing-tools-for-web-applications-that-you-could-use-in-2020-powerful-and-23135826a569
12. Jennifer Niederst Robbins, Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics / Jennifer Niederst Robbins - O'Reilly Media, 2012 – 623c.

13. Савельева Н.В. Основы программирования на PHP: курс лекций. Учебное пособие для студентов вузов, обучающихся по специальностям в области информационных технологий / Савельева Н.В. - Интернет-Университет Информационных Технологий (ИНТУИТ), 2005. - 264 с.
14. Ларри Ульман MySQL /У. Ларри - М.: ДМК Пресс, 2007. - 352 с.
15. Федорчук Д. А. Разработка WEB приложений на PHP и MySQL /Д.А. Федорчук - СПб. : Корона-принт, 2013. – 340 с.
16. Савин Р. Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах/ Р. Санин — М.: Дело, 2007. — 312 с.

ДОДАТОК

Файл index.js:

```
/**
 * Run the auto testing and test the web site for functionality
 *
 * @type {App}
 */
import App from "./app/app";

const app = new App();

app.run().catch((e) => {
  console.log(e);
});
```

Файл App.js:

```
import Shared from "./shared";
import HtmlCase from "../cases/html";
import LinksPreloader from "../preloaders/links";
import fs from 'fs/promises';
import ImgCase from "../cases/img";
import CssCase from "../cases/css";
import PuppeteerCase from "../cases/puppeteer";
import HeadersCase from "../cases/headers";
import DelayCase from "../cases/delay";

export default class App {
  constructor() {
    this.shared = new Shared();
    this.preloaders = [
      LinksPreloader,
    ];
    this.cases = [
      HtmlCase,
      ImgCase,
      CssCase,
      PuppeteerCase,
      HeadersCase,
      DelayCase,
    ];
  }

  run = async () => {
    await this.runPreload();
    await this.runCases();
    await this.writeToFile();
  }

  runPreload = async () => {
    for (const preloaderClass of this.preloaders) {
      const preloader = new preloaderClass(this.shared);
```

```
      await preloader.run();
    }
  }

  runCases = async () => {
    for (const caseClass of this.cases) {
      const caseObject = new caseClass(this.shared);
      await caseObject.run();
    }
  }

  writeToFile = async () => {
    try {
      const errors = this.shared.errors.sort((a, b) => b.severity - a.severity);

      await fs.writeFile('./result.json', JSON.stringify(errors, null, 4));
    } catch (e) {
      console.log(e);
    }

    try {
      const timeList = {};

      Object.keys(this.shared.pages).map(key => {
        const page = this.shared.pages[key];

        timeList[key] = page.time;
      });

      await fs.writeFile('./runtime/time.json', JSON.stringify(timeList));
    } catch (e) {
      console.log(e);
    }
  }
}
```

Файл error.js:

```
export const SEVERITY_LOW = 1;
export const SEVERITY_MEDIUM = 2;
export const SEVERITY_HIGH = 3;
export const SEVERITY_CRITICAL = 4;
export class AutoTestError {
  constructor() {
    this.name = '';
    this.severity = '';
    this.page = '';
    this.data = {};
  }
}
```

Файл shared.js:

```
export default class SharedData {
  constructor() {
    this.errors = [];
    this.pages = {};
  }
}
```

Файл case.js:

```
import {createError} from "../utils/error";

export default class BaseCase {
  /**
   *
   * @param {SharedData} shared
   */
  constructor(shared) {
    this.shared = shared;
  }

  /**
   * Робить перевірку та визначає де можуть бути помилкі
   * @public
   * @returns {Promise<void>}
   */
  async run() {}

  addError = (name, severity, page, data) => {
    const error = createError(name, severity, page, data);

    this.shared.errors.push(error);
  }
}
```

Файл css.js:

```
import {findLinks, getUrlErrors} from "../utils/urls";
import {SEVERITY_MEDIUM} from "../app/error";
import BaseCase from "./case";

export default class CssCase extends BaseCase {
  async run() {
    await Promise.all(
      Object.keys(this.shared.pages).map(async (key) => {
        const links = findLinks(this.shared.pages[key].html, /<link\s*href="\/([a-zA-Z0-
9_/.]+css)"/gi);

        await Promise.all(
          links.map(async link => await this.checkUrl(key, link))
        );
      })
    );
  }
```

```
  async checkUrl(page, url) {
    const errorData = await getUrlErrors(url);

    if (errorData) {
      this.addError('CSS файл ' + url + ' не доступний', SEVERITY_MEDIUM, page, errorData);
    }
  }
}
```

Файл delay.js:

```
import BaseCase from "./case";
import fs from 'fs/promises';
import config from '../config/main';
import {SEVERITY_MEDIUM} from "../app/error";

export default class DelayCase extends BaseCase {
  async run() {
    const lastResults = await this.getLastResults();

    if (!lastResults) {
      return;
    }

    for (const [url, {time}] of Object.entries(this.shared.pages)) {
      const lastTime = lastResults[url];

      if (lastTime == null) {
        continue;
      }

      if (time / config.time_max_delay_multiplier > lastTime) {
        this.addError(`Сторінка ${url} стала завантажуватись набагато більше часу!`,
SEVERITY_MEDIUM, url, {
          lastTime,
          time,
        });
      }
    }
  }

  async getLastResults() {
    return fs.readFile('runtime/time.json');
  }
}
```

Файл headers.js:

```
import BaseCase from "./case";
import config from './../config/main';
import fetch from "node-fetch";
import {SEVERITY_HIGH, SEVERITY_LOW} from "../app/error";
```

```
export default class HeadersCase extends BaseCase {
  async run() {
    const response = await fetch(config.url);
    const headers = response.headers;
    const headersConfig = this.getHeadersConfig();

    for (const [name, value] of Object.entries(headersConfig)) {
      const headerFromResponse = headers.get(name);

      if (!headerFromResponse) {
        this.addError(`Відсутній ${name} хедер який може призвести до погіршення безпеки`,
SEVERITY_HIGH);
      }

      if (value !== null && headerFromResponse !== value) {
        this.addError(`Значення хедера ${name}(${headerFromResponse}) не співпадає з
рекомендованим ${value}`, SEVERITY_LOW);
      }
    }
  }

  getHeadersConfig() {
    return {
      'x-powered-by': null,
      'strict-transport-security': null,
      'content-security-policy': null,
      'x-frame-options': 'DENY',
      'x-content-type-options': 'nosniff',
      'referrer-policy': 'no-referrer',
      'x-xss-protection': '1; mode=block;',
      'content-type': 'text/html; charset=UTF-8',
    };
  }
}
```

Файл html.js:

```
import BaseCase from "./case";
import {SEVERITY_CRITICAL, SEVERITY_HIGH, SEVERITY_MEDIUM} from "../app/error";

export default class HtmlCase extends BaseCase {
  /**
   * Робить перевірку та визначає де можуть бути помилкі
   *
   * @returns {Promise<void>}
   */
  async run() {
    await Promise.all(
      Object.keys(this.shared.pages).map(async (key) => {
        await this.checkUrl(key, this.shared.pages[key]);
      })
    );
```

```
  }

  async checkUrl(url, {html}) {
    if (!html || !html.length) {
      this.addError('HTML пустий для сторінки ' + url, SEVERITY_CRITICAL, url);
    }

    if (!html.includes('<!DOCTYPE html>')) {
      this.addError('Doctype повинен міститись у документі', SEVERITY_HIGH, url);
    }

    const tags = [
      'div',
      'p',
      'h1',
      'h2',
      'h3',
      'h4',
      'h5',
      'h6',
      'head',
      'body',
      'header',
      'article',
    ];

    tags.forEach(tag => {
      const openTagLength = html.split('<' + tag + '>').length - 1;
      const closingTagLength = html.split('</' + tag + '>').length - 1;
      const notClosedSize = openTagLength - closingTagLength;

      if (notClosedSize > 0) {
        this.addError('Знайдено ' + notClosedSize + ' не закритих тегів ' + tag,
SEVERITY_MEDIUM, url);
      }
    });
  }
}
```

Файл img.js:

```
import BaseCase from "./case";
import {SEVERITY_MEDIUM} from "../app/error";
import {findLinks, getUrlErrors} from "../utils/urls";

export default class ImgCase extends BaseCase {
  async run() {
    await Promise.all(
      Object.keys(this.shared.pages).map(async (key) => {
        const links = findLinks(this.shared.pages[key].html, /<img\s*src="\/([a-zA-Z0-
9_/.]+(jpg|png))"/gi);

        await Promise.all(
```

```
        links.map(async link => await this.checkUrl(key, link))
      );
    })
  );
 }

 async checkUrl(page, url) {
  const errorData = getUrlErrors(url);

  if (errorData) {
   this.addError('Зображення ' + url + ' не доступне', SEVERITY_MEDIUM, page, errorData);
  }
 }
}
```

Файл puppeteer.js:

```
import BaseCase from "./case";
import Puppeteer from "puppeteer";
import {getFullUrl} from "../utils/urls";
import {createDir, getFiles, isEmptyDir} from "../utils/fs";
import {compareImages} from "../utils/img";
import {SEVERITY_HIGH} from "../app/error";

export default class PuppeteerCase extends BaseCase {
 async run() {
  await this.init();
  await this.loadPhotos();
  await this.comparePhotos();
  await this.puppeteer.close();
 }

 async init() {
  this.puppeteer = await Puppeteer.launch({
   headless: true,
  });

  this.isEmptyMaster = await isEmptyDir('img/master');

  await createDir('img');
  await createDir('img/master');
  await createDir('img/beta');
  await createDir('img/compare');
 }

 async loadPhotos() {
  await Promise.all(
   Object.keys(this.shared.pages).map(async url => {
    const page = await this.puppeteer.newPage();
    await page.setViewport({
     width: 1920,
     height: 1080,
    });
```

```
        await this.go(url, page);
        await this.screenShot(url, page);

        await page.close();
      })
    );
  }

  async comparePhotos() {
    const files = await getFiles('img/beta');

    await Promise.all(
      files.map(async file => {
        try {
          await compareImages('img/beta/' + file, 'img/master/' + file);
        } catch (data) {
          const url = file.replaceAll('_', '/').replace('.png', '');
          this.addError('Зображення при порівнянні з мастером для посилання ' + url + ' має
розбіжності',
            SEVERITY_HIGH, url, data);
        }
      })
    );
  }

  async go(url, page) {
    await page.goto(getFullUrl(url), {
      waitUntil: 'networkidle2',
    });
  }

  async screenShot(url, page) {
    const path = 'img/'
      + (this.isEmptyMaster ? 'master' : 'beta')
      + '/'
      + url.replaceAll('/', '_')
      + '.png';

    await page.screenshot({
      path,
    });
  }
}
```

Файл config.js:

```
export default {
  url: 'http://localhost:8001',
  link_depth: 3,
  time_max_delay_multiplier: 3,
};
```

Файл preloaders/base.js:

```
import {createError} from "../utils/error";

export default class BasePreloader {
  /**
   * @param {SharedData} shared
   */
  constructor(shared) {
    this.shared = shared;
  }

  /**
   * Завантажує дані для подальшої їх обробки
   */
  async run() {}

  addError = (name, severity, page, data) => {
    const error = createError(name, severity, page, data);

    this.shared.errors.push(error);
  }
}
```

Файл links.js:

```
import BasePreloader from "./base";
import config from '../config/main';
import {SEVERITY_HIGH} from "../app/error";
import fetch from 'node-fetch';
import {findLinks, getFullUrl} from "../utils/urls";

export default class LinksPreloader extends BasePreloader {
  async run() {
    this.checkedUrls = [];

    await this.checkUrl(config.url, config.url,0);
  }

  async checkUrl(url, baseUrl, depth) {
    if (depth > config.link_depth) {
      return;
    }

    if (this.checkedUrls.includes(url)) {
      return;
    }

    this.checkedUrls.push(url);

    let errorData = null;
    try {
      const timeStart = Date.now();
```

```
      const response = await fetch(getFullUrl(url));

      if (response.status === 200) {
        const text = await response.text();
        const urls = findLinks(text, /<a\s*href="\/([a-zA-Z0-9/]+)"/gi);

        if (depth < 2 && url !== baseUrl) {
          this.shared.pages[url] = {
            links: urls,
            html: text,
            time: Date.now() - timeStart,
          };
        }

        await Promise.all(
          urls.map(async childUrl => {
            await this.checkUrl(childUrl, url,depth + 1);
          })
        )
      } else {
        errorData = response;
      }
    } catch (e) {
      errorData = e;
    }

    if (errorData) {
      this.addError('Посилання ' + url + ' недоступне', SEVERITY_HIGH, baseUrl, errorData);
    }
  }
}
```

Файл utils/error.js:

```
import {AutoTestError} from "../app/error";

const createError = (name, severity, page, data) => {
  const error = new AutoTestError();

  error.name = name;
  error.severity = severity;
  error.page = page ? page: 'index';
  error.data = data ? data : {};

  return error;
}

export {createError};
```

Файл fs.js:

```
import fs from "fs/promises";

const createDir = async (path) => {
```

```
  try {
    await fs.mkdir(path);
  } catch (e) {}
}

const isEmptyDir = async (path) => {
  try {
    const files = await fs.readdir(path);

    return !files.length;
  } catch (e) {}

  return true;
}

const getFiles = async (path) => {
  return await fs.readdir(path);
}

export {createDir, isEmptyDir, getFiles};
```

Файл utils/img.js:

```
import BlinkDiff from "blink-diff";

const compareImages = async (img1, img2) => {
  const img1Name = img1.split('/').pop().replace('.png', '');
  const output = './img/compare/' + img1Name + '.png';

  return new Promise((resolve, reject) => {
    let diff = new BlinkDiff({
      imageAPath: img1,
      imageBPath: img2,

      thresholdType: BlinkDiff.THRESHOLD_PERCENT,
      threshold: 0.02,

      imageOutputPath: output,
    });

    diff.run(function (error, result) {
      if (error) {
        reject({
          error,
          diffCount: 0,
          output,
        });
      } else {
        if (diff.hasPassed(result.code)) {
          resolve(output);
        } else {
          reject({
            error: 'Є розбіжності між зображеннями 1 та 2',
```

```
            diffCount: result.differences,
            output,
          });
        }
      }
    });
  });
}

export {compareImages};
```

Файл urls.js:

```
import config from "../config/main";
import fetch from "node-fetch";

const findLinks = (html, regex) => {
  const links = [];

  let link = regex.exec(html);
  while (link !== null) {
    links.push(link[1]);

    link = regex.exec(html);
  }

  return links.filter((elem, pos, arr) => {
    return arr.indexOf(elem) === pos;
  });
}

const getUrlErrors = async (url) => {
  try {

    const response = await fetch(getFullUrl(url));

    if (response.status !== 200) {
      return response;
    }
  } catch (e) {
    return e;
  }
}

const getFullUrl = (url) => {
  if (!url.includes('http')) {
    if (url.startsWith('/')) {
      url = config.url + url;
    } else {
      url = config.url + '/' + url;
    }
  }
```

```
  return url;
}

export {findLinks, getUrlErrors, getFullUrl};
```

Файл .gitignore:

```
/node_modules
/dist
./result.json
./runtime
```

Файл package.json:

```
{
  "name": "auto-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "babel-node src/index.js",
    "babel-node": "babel-node",
    "build": "NODE_ENV=production babel src --out-dir dist --source-maps inline",
    "prepublishOnly": "npm run build"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "blink-diff": "^1.0.13",
    "node-fetch": "^2.6.1",
    "puppeteer": "^5.4.1"
  },
  "devDependencies": {
    "@babel/cli": "^7.12.1",
    "@babel/core": "^7.12.3",
    "@babel/node": "^7.12.6",
    "@babel/plugin-proposal-class-properties": "^7.12.1",
    "@babel/preset-env": "^7.12.1",
    "babel-eslint": "^10.1.0",
    "eslint": "^7.13.0",
    "eslint-config-airbnb-base": "^14.2.1",
    "eslint-plugin-babel": "^5.3.1",
    "eslint-plugin-import": "^2.22.1"
  }
}
```

Файл .eslintrc.yml:

```
plugins:
  - babel
env:
  node: true
parser: babel-eslint
```

```
extends:
  - 'airbnb-base'
rules:
  no-console: 0
```

Файл .babelrc:

```
{
  "presets": [
    ["@babel/preset-env",
      {
        "targets": {
          "node": "current"
        }
      }
    ]
  ],
  "plugins": [
    "@babel/plugin-proposal-class-properties"
  ]
}
```