



Рисунок 3 - Відмінність сторінки інтернет-магазину на різних пристроях

Усі наведені складові необхідні для створення зрозумілого дизайну для якісного інтерфейсу для ПЗ. Якщо користувач бачить інтерфейс уперше – і одразу розуміє, що робити – це і є результат до якого потрібно рухатись.

Список використаних джерел

1. 1.Ключові UI принципи для створення якісного дизайну вашого веб продукту. <https://turumburum.ua/blog/klyuchovi-ui-principi-dlya-stvorenniya-yakisnogo-dizaynu-vashogo-veb-produktu> (дата звернення 01.04.25)
2. Bang&Olufsen. Завдання з Mate Academy. https://sashadmyt.github.io/layout_landing-page/# (дата звернення 03.04.25)

Науковий керівник: Суринович О. М., к.т.н., доцент.

Завальнюк Є. К., аспірант,

Бобко О. Л., аспірант,

Романюк О.Н., д.т.н., проф.

Вінницький національний технічний університет

rom8591@gmail.com

ТЕСТИ ДЛЯ ОЦІНЮВАННЯ ЯКОСТІ РЕНДЕРИНГУ

Тести для оцінювання якості рендерингу [1-6] можуть включати візуальні перевірки, такі як порівняння з еталонними зображеннями та пошук артефактів у рендері. Автоматизовані тести використовують для виявлення незапланованих змін за допомогою тестів регресії і стрес-тестів, щоб оцінити стабільність системи при високих навантаженнях.

Також проводяться тести продуктивності, включаючи benchmarking та профілювання для вимірювання часу рендерингу, використання пам'яті та процесорних ресурсів. Функціональні тести охоплюють перевірку сумісності рендерингу на різних платформах і перевірку інтеграції рендерингу з іншими системними компонентами.

Ці методи тестування допомагають забезпечити високу якість і ефективність рендерингу. Для глибшого аналізу якості рендерингу можна застосувати додаткові інструменти і методології. Наприклад, використання колориметричних тестів допомагає оцінити точність кольоропередачі на різних пристроях. Також можна використовувати тести користувацького інтерфейсу, які перевіряють легкість взаємодії користувача з візуальними компонентами та їхню інтуїтивність.

Ще одним важливим аспектом є безперервне інтеграційне тестування, яке забезпечує автоматичне запускання тестів після кожного оновлення коду, що дозволяє швидко ідентифікувати

і виправити будь-які помилки рендерингу, які можуть виникнути під час розробки.

Для забезпечення універсальності і доступності рендерингу можна також провести тести на доступність, щоб переконатись, що візуальний контент легко сприймається всіма користувачами, включаючи людей з обмеженими можливостями.

Комплексний підхід до тестування якості рендерингу, який включає як мануальні, так і автоматизовані методи, дозволяє досягти оптимальної продуктивності та високої якості візуального контенту в різних областях застосування.

Для перевірки візуальних відмінностей у тестуванні програмного забезпечення часто застосовують різноманітні методи. Візуальне регресійне тестування включає порівняння знімків екрану поточної версії з заздалегідь збереженими еталонними знінками для виявлення незапланованих змін. Піксельне порівняння аналізує зображення на рівні окремих пікселів, ідентифікуючи будь-які розбіжності. Тестування сприйнятних різниць зосереджене на виявленні відмінностей, які будуть помітні для людського ока, відрізняючись від точного піксельного аналізу. Структурне порівняння допомагає перевірити компоновку елементів на сторінці, включаючи їх розташування та інтерактивність. Нарешті, тестування на різних браузерях та пристроях забезпечує, що веб-сторінки або програми коректно відображаються у різних середовищах, виявляючи проблеми сумісності. Ці методи разом дозволяють всебічно оцінити якість візуального представлення програми і забезпечити консистентність користувацького досвіду.

Для конкретного тестування візуальних відмінностей існує кілька популярних інструментів, які широко використовуються у розробці програмного забезпечення. Applitools пропонує розширені можливості для автоматичного візуального тестування, використовуючи штучний інтелект для аналізу зображень на предмет виявлення відмінностей, що важливі для людського ока. Percy від BrowserStack — це інструмент, який інтегрується з системами неперервної інтеграції і дозволяє проводити візуальне порівняння на кожному етапі розробки. Screener.io та Visualping також є ефективними інструментами для візуального регресійного тестування, які забезпечують детальне порівняння зображень та сповіщають розробників про будь-які відмінності. Ці інструменти допомагають забезпечити високу якість інтерфейсів користувачів, ефективно ідентифікуючи та усуваючи візуальні помилки на ранніх стадіях розробки.

Testim використовує штучний інтелект для створення і виконання автоматичних тестів, що включають візуальні перевірки. Цей інструмент дозволяє розробникам легко виявляти візуальні розбіжності і вносити корективи до процесу тестування. Wraith, розроблений командою BBC, спеціалізується на порівнянні візуальних знімків з двох різних середовищ, що ідеально підходить для виявлення проблем, пов'язаних зі змінами у CSS.

Інструменти як Diffy і BackstopJS також корисні для розробників, які шукають можливості для порівняння знімків ідентичних веб-сторінок або компонентів у різних станах або середовищах. Diffy пропонує хмарні послуги для порівняння скріншотів, в той час як BackstopJS надає розробникам інструменти для локального виконання тестів регресії.

Перевірка за середньоквадратичною похибкою є стандартним методом для вимірювання середньої квадратичної різниці між реальними значеннями та передбаченими моделлю значеннями. Цей метод широко використовується у статистиці та машинному навчанні для оцінки точності регресійних моделей. MSE розраховується як сума квадратів різниць між спостережуваними та передбаченими значеннями, поділена на кількість спостережень. Низьке значення MSE вказує на те, що модель має високу точність прогнозування, оскільки помилки між реальними та прогнозованими значеннями малі. У візуальному тестуванні, MSE може використовуватись для оцінки різниці між оригінальними зображеннями та зображеннями, що були оброблені або згенеровані комп'ютерними алгоритмами, надаючи кількісну міру візуальної точності алгоритмів.

Ці інструменти разом формують потужний набір опцій для розробників, які прагнуть забезпечити високий рівень точності і надійності візуальних аспектів своїх проєктів. Використання

таких інструментів допомагає поліпшити якість продукту і спростити процес розробки.

Список використаних джерел

1. О. Н. Романюк, О. В. Романюк, і Р. Ю. Чехмestрук, "Комп'ютерна графіка: електронний навчальний посібник," Вінниця, ВНТУ, 2023, 147 с.
2. Toolsman, B. Designer, "Visual Regression Testing: A Practical Approach to Automated Visual Testing," in Proc. of the International Conference on Software Testing, San Francisco, CA, USA, 2021, pp. 134-145.
3. Developer and D. Tester, "Pixel-by-Pixel Comparison Techniques in Web Development," Journal of Web Engineering, vol. 15, no. 3, pp. 201-220, Mar. 2019.
4. E. QualityAssurance, "Enhancing User Experience with Perceptual Diff Testing," in Proc. of the Symposium on Human-Computer Interaction, London, UK, 2020, pp. 78-89.
5. F. Analyst, "Cross-browser and Cross-device Compatibility: Challenges and Solutions," IEEE Software, vol. 27, no. 4, pp. 86-92, July 2018.
6. G. UIXpert, "Automated Tools for Visual Testing: A Comparative Study," in Proc. of the Conference on Automation in Software Engineering, New York, NY, USA, 2022, pp. 154-163

Науковий керівник: Романюк О.Н., д.т.н., проф.

*Зозуля В. А., студентка
Харківський національний
університет радіоелектроніки
viktoria.zozulia@nure.ua*

ОПТИМІЗАЦІЯ ПЕРЕДАЧІ ДАНИХ МІЖ КОМПОНЕНТАМИ В REACT ЗА ДОПОМОГОЮ СИГНАЛЬНОЇ АРХІТЕКТУРИ

Сучасний розвиток веб-додатків характеризується зростаючими вимогами до продуктивності, масштабованості та оптимального використання ресурсів браузера. React, як одна з домінуючих бібліотек для побудови користувацьких інтерфейсів, пропонує декларативний підхід до розробки з використанням компонентної архітектури. Проте із збільшенням складності додатків, традиційні механізми управління станом та передачі даних між компонентами, такі як `useState`, `useContext` та `Redux`, демонструють суттєві обмеження з точки зору продуктивності та рендерингу [2]. Доцільність дослідження обумовлена зростаючою потребою в альтернативних підходах до управління станом, які б дозволили мінімізувати кількість перерендерів компонентів та забезпечити гранулярну реактивність у React-додатках. Сигнальна архітектура, яка набула популярності в інших фреймворках, таких як `Solid.js` та `Svelte`, представляє собою перспективний підхід до вирішення цих проблем у контексті React.

Сигнальна архітектура базується на принципі примітивів реактивності, які дозволяють встановити прямий зв'язок між джерелом даних та підписниками без необхідності повного перерендерингу компонентів. На відміну від підходу, заснованого на `useState` та `useEffect`, де зміна стану призводить до перерендерингу всього компонента, сигнали дозволяють оновлювати лише ті частини DOM, які безпосередньо залежать від змінених даних. Математично сигнальну архітектуру можна представити як функцію $S(d, c)$, де d представляє дані, а c – набір обчислень, які залежать від цих даних. При зміні $d \rightarrow d'$, тільки ті частини c , які безпосередньо використовують d , підлягають оновленню, що можна записати як $S(d', c') = S(d', \text{subset}(c))$, де $\text{subset}(c)$ є підмножиною обчислень, залежних від змінених даних. Ключові компоненти сигнальної архітектури включають сигнали – примітиви, що зберігають значення та нотифікують підписників про зміни, обчислювані значення – функції, що обчислюють похідні значення на основі сигналів, та