МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп‘ютерних наук

Кваліфікаційна робота магістра

Інформаційна технологія забезпечення використовуваності та веб-доступності стримінгової платформи

Здобувач освіти гр. ІК.м-11 Євгенія РУДЕНКО

Науковий керівник Надія ТИРКУСОВА

В. о. завідувача кафедри
доцент, к.т.н. Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

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

Затверджую:
В.о.зав.кафедри _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Руденко Євненія Павлівна
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія забезпечення використовуваності та веб-доступності стримінгової платформи*

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) аналітичний огляд літератури; 2) постановка завдання; 3) вибір методів рішення завдання; 4) програмна реалізація поставленого завдання; 5) висновки*

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/	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Аналітичний огляд літератури		
2.	Постановка задачі та формування завдань дослідження.		
3.	Вибір методів рішення завдання		
4.	Програмна реалізація поставленого завдання		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____

РЕФЕРАТ

Записка: 50 с., 26 рис., 1 таб., 2 схеми, 6 додаткіи, 15 літературних джерел.

Об'єкт дослідження – Інформаційна технологія забезпечення використовуваності та веб-доступності стримінгової платформи

Мета роботи – створення інформаційної технології забезпечення використовуваності та веб-доступності стримінгової платформи. Завдяки цієї платформи користувачі зможуть шукати та дивитися інформацію стосовно улюблених фільмів та сералів, а також додавати їх до листа фаворитів. Платформа матиме зрозумілий, зручний та інтуїтивний функціонал.

Результати – створено інформаційної технології забезпечення використовуваності та веб-доступності стримінгової платформи.

Під час виконання роботи було реалізовано: пошук та дослідження необхідної інформації, яка відповідає темі дипломної роботи; проведений аналіз платформ конкурентів; вибір засобів реалізації додатка; реалізація платформи за допомогою NextJS, ReactJS, JS, SASS, HTML; розгортування; тестування та аналіз результату;

.

.

ІНФОРМАЦІЙНА СИСТЕМА, СТРІМОНГОВА ПЛАТФОМА, ПЛАГІНИ, REACT, HOOKS, ROUTING, HTML, CSS, API, JAVASCRIPT, NEXTJS, NPM, VS CODE, ВИКОРИСТОВУВАНОСТЬ, ВЕБ-ДОСТУПНОСТІ, VOD

ЗМІСТ

ВСТУП

У сучасному світі стрімко зростає кількість вебресурс, які урізноманітнюють наше повсякденне життя, наприклад: соціальні мережі, стрімінгові платформи.

З початком епідемії, все більше і більше людей стали прихильниками сервісів які роблять перебування дома цікавішим. Авжеж якщо з'являється потреба та інтерес, то і з'являється велика кількість пропозицій. Але, на жаль, не всі вони розроблюються за стандартами та практиками які роблять веб сервіси – user friendly. Зручність вебсайт є однією з основних складових ергономіки програмного забезпечення. Цей термін належати до легкості, з якою користувачі можуть отримати доступ до додатка з використанням будь якого пристроюю Легко зрозумілий і швидкий у використанні вебсайт – це те, що має велике значення, коли йдеться про зручність для користувачів.

Також важливе значення має вебдоступність. Коли вебсайт та вебінструменти належним чином розроблені та закодовані, люди з обмеженими можливостями мають можливість з легкістю ними користуватися. Однак наразі багато сайтів і інструментів розроблено з бар'єрами доступності, що ускладнює або робить неможливим їх використання для деяких людей. Розроблення додатків доступними приносить користь усім користувачам, не зважаючи на їх особливості. Міжнародні вебстандарти визначають та узагальнюють, що саме потрібно для вебдоступності.

В даному проєкті було проаналізовано ринок конкурентних стрімінгових платформ, розроблено логотип та створено адаптивний вебдодаток з використанням правил вебдоступності, завдяки якому усі користувачі можуть швидко та зручно знайти улюблені фільми чи програми.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд предметної області

Стрімінгова платформа – це онлайн-джерело розваг, завдяки якому кожен може влаштувати театр, кінотеатр чи подивитися інші віртуальні розваги онлайн, не виходячи з дому.

Ці послуги є альтернативою кабельним і супутниковим послугам, але часто за нижчою ціною та більшим спектром послуг. Замість того, щоб зберігатися на жорсткому диску пристрою, відео або медіа файл надсилається у вигляді стисненого відеоматеріалу через Інтернет для миттєвого відтворення. Замість того, щоб чекати, поки фільм буде завантажено на пристрій, а потім переглядати його, це можна зробити у реальному часі.

Будь-який пристрій, який має підключення до Інтернету та програми, які можуть розпакувати вміст, може використовувати служби відео стрімінгу. Але іноді послуги можуть бути орієнтовані на обмежений тип пристроїв, наприклад тільки для девайсів певного бренду, подібно до Apple TV або Apple Fitness+, які працюють тільки на певних пристроях Apple. Інші додатки можуть підтримуватися рекламою, працювати за безкоштовною моделю, а також пропонувати придбати контент за окрему плату.

Однією з основних причин, що сприятливо впливає на галузь, є розширення використання Інтернету та зростання продажів мобільних пристроїв. На додаток до цього, спостерігається помітне зростання попиту на послуги потокового відео в коледжах, університетах та установах по всьому світу. Це можна віднести до його переваг, які включають покращення процесів навчання завдяки візуальним вебінарам і записам курсу. Крім того, компанії широко використовують прямі трансляції, оскільки це допомагає їм просувати свої товари та послуги, розвивати свої бренди та покращувати взаємодію з клієнтами.

Останнім часом спостерігаються певні фактори росту:

Попит на послуг стрімінгу медіа файлів за запитом (Video on demand) зростає, що сприяє значному розширенню ринку.

За даними звіту Motion Picture Association за 2020 рік, кількість користувачів онлайн VoD зросла приблизно до 1,11 мільярда під час епідемії COVID-19, а до 2023 року, за прогнозами, досягне 2 мільярдів користувачів. Обсяг світового ринку потокового відео в 2021 році оцінювався в 375,1 мільярда доларів США, а до 2030 року очікується, що він досягне приблизно 1721,4 мільярда доларів США[1].

Рисунок 1.1– Зростання ринку стрімінгових послуг 2021 – 2030 роки[1].

Ключові ринкові можливості

Вплив потокового відео в освіті. Відео може допомогти учням запам’ятати матеріал ефективніше, що може мати величезний позитивний вплив на освіту. До цієї категорії входять: вебінари, відеолекції. Освітні організації, включаючи університети, школи та коледжі, тепер створюють інтерактивну інформацію та поширюють її за допомогою відеопрезентацій. Таким чином вони ефективно використовують технології для поширення знань. Обов’язки вчителів суттєво змінюються внаслідок зростання попиту на онлайн формат. Останні тепер можуть викладати за допомогою відео у віртуальних класах по

всьому світу та демонструвати добірку навчальних фільмів у класі, що сприяє навчальному процесу.

Вплив потокового відео у кіноіндустрії. Стрімінгові сервіси надають широкий спектр контенту, не виходячи з дому. Деякі служби безкоштовні, приклад BBC Player, для використання інших платформ вам доведеться платити, будь то підписка, приклад Netflix, чи індивідуальна оренда, як-от SweetTV. Деякі сервіси потокового передавання пропонують суміш безкоштовного перегляду, з можливістю подивитися певний контент — на основі прокату чи купівлі, приклад Amazon Prime. Деякі послуги надійшли від уже існуючих виробничих компаній, прикладом може слугувати Disney+, тоді як інші створили власні виробничі компанії після створення своєї стрімінгової платформи, як приклад - Netflix.

1.2 Поняття зручності, використовності та вебдоступності відносно стрімінгових платформ

Коли пандемія COVID-19 змусила суспільство змінити свої звички та залишитися вдома, багато людей знайшли розраду у віртуальному світі стрімінгових платформ. Це й не дивно, оскільки такі платформи дозволяють користувачам дивитися те, що вони хочуть, як завгодно і де завгодно.

Зручний для користувача – це термін, який ми використовуємо для опису всього, чим легко навчитися користуватися. Іншими словами, навіть якщо це складне призначення, використовувати його просто і зрозуміло. Термін використовується для комп’ютерних інтерфейсів, пристроїв, обладнання, засобів і систем. Термін описує все, що сумісно зі знаннями, навичками, концентрацією та іншими здібностями користувача.

Ми використовуємо цей термін, описуючи, наприклад, програми, у значенні, що й «інтуїтивно зрозумілий». Якщо користування пристроєм, інтуїтивно зрозуміле, це означає, що користувач, можете «відчути» свою поведінку.

Використовність — це міра того, наскільки добре конкретний користувач у конкретному контексті може використовувати продукт/дизайн для ефективного та задовільного досягнення визначеної мети. Щоб забезпечити максимальну зручність використання, дизайнери зазвичай вимірюють зручність використання дизайну протягом усього процесу розробки — від дизайну до кінцевого продукту.

Вебдоступність — це група норм та інструменти веб-розробки, при застосуванні яких люди з обмеженими можливостями можуть користуватися веб додатками на рівні звичайних користувачів[2].

1.2.1 Найголовніші функції стімінгових платформ з перспективи зручності

Користувачі вимогливі до перегляду улюбленого контенту. Часи, коли глядачі сідали перед телевізором в певний час і день, давно минули. Зараз, коли світ швидко змінюється, мінливе робоче середовище та мобільний спосіб життя, глядачі хочуть мати контроль над тим, як вони дивляться та сприймають контент[3].

Які функції шукають користувачі в стімінгових платформах?

Зручний інтерфейс Користувачі хочуть мати легкий доступ до своєї платформи. Зручний інтерфейс повинен бути важливою особливістю кожного додатку. Перед реалізацією додатку варто опрацювати усі візуальні аспекти та провести UX та UI тести, щоб переконатися, що дизайн максимально зручний для користувача;

Безпека та конфіденційність. Сьогодні доступ до візуального та аудіоконтенту є простим і необмеженим. Але це також сприяє зовнішнім загрозам. Тому важливо дбати про стандарти шифрування, сертифікати та протоколи безпеки;

Управління відеоконтентом. Система керування вмістом (CMS) допомагає класифікувати весь вміст і створити організоване середовище для всіх фільмів, а також полегшує глядачам навігацію в системі;

Особливості налаштування. Створення унікального досвіду і надання користувачам контролю над платформою;

Сумісність. Платформа має бути сумісна з іншими пристроями.

Батьківський контроль. Завдяки цій функції батьки можуть контролювати контент, доступний дітям;

Регулярні оновлення контенту.

1.2.2 Найголовніші функції стімінгових платформ з перспективи веб доступності

Більше людей дивляться свої улюблені шоу онлайн, ніж будь-коли раніше. Але багато сервісів потокового передавання все ще відстають, коли йдеться про те, щоб зробити послуги та шоу доступними для людей з обмеженими можливостями[4].

Функції стімінгових платформ з перспективи веб доступності:

- Наявність якісних субтитрів. Це функція не лише для людей із втратою слуху. Багатьом людям так, таким чином легше стежити за тим, що відбувається на екрані;
- Легка навігація;
- Тригерні попередження;
- Звуковий опису. Він забезпечує паралельну розповідь під час перегляду фільму чи серіалу на Netflix. Користувач можете почути про все, що відбувається на екрані, включаючи середовище, в якому перебувають персонажі, опис міміки та рухів персонажа, костюми, зміни сцени, декорація.;
- Зображення контенту без миготінь;
- Поганий колірний контраст;
- Наявність підпису у блоків;
- Голосові команди. Голосові елементи керування можуть допомогти користувачу шукати, переглядати та керувати медіа файлом, який він переглядаєте (старт, пауза, перемотування тощо). Вони корисні людям,

які мають проблеми з пересуванням, і людям із вадами зору. Набагато простіше, не кажучи вже про те, що швидше, сказати те, що ви хочете, ніж шукати потрібні елементи керування.

1.3 Огляд подібних рішень

Зараз спостерігається величезний попит на стрімінгові послуги, тому можна знайти безліч аналогів та платформ. Одними з найпопулярніших в Україні є: Netflix, SweetTV, Megogo, та YouTube. Ці платформи мають як багато спільних так і різних рис.

Додаток WatchOnline було створення з метою аналізу ринку, можливості вдосконалити функції найпопулярніших платформ та застосування понять використовності та вебдоступності.

Перед розробкою стрімінгової платформи “WatchOnline” був проведено огляд вже існуючих аналогів у цій області, таких як:

– Netflix;
– SweetTV;

Кожна з вище приведених платформ має свої характерні особливості та відмінності.

Netflix є лідером серед передплатних відеосервісів. Він поєднує в собі різноманітні телешоу та фільми за відносно дешеву щомісячну плату, і існує вже дуже давно.

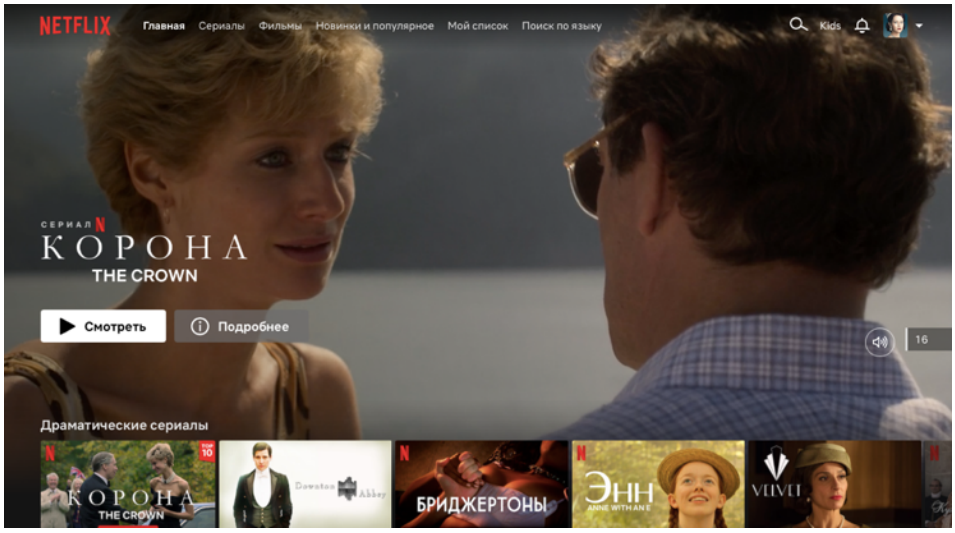

Рисунок 1.2– Головний екран Netflix.

Переваги Netflix:

- Наявність власних шоу, фільмів та серіалів;
- Без реклами;
- Широкий асортимент фільмів;
- Наявність Kids mode;
- Сімейна підписка

Недоліки Netflix:

- Не має можливості дивитися ТВ онлайн;
- Відносно малий вибір озвучки та субтитрів для Українського ринку.

Можливості веб доступності на Netflix[5]:

- Аудіоопис;
- Закриті субтитри [CC];
- Голосові команди;
- Швидкість відтворення.

Аналіз веб доступнасті за допомогою сервісу LightHause

Google Lighthouse – це автоматизований інструмент із відкритим кодом для вимірювання якості веб-сторінок. Його можна запустити на будь-якій веб-сторінці, загальнодоступній або з вимогою автентифікації. Google Lighthouse перевіряє продуктивність, доступність і оптимізацію веб-сторінок для пошукових систем[6].

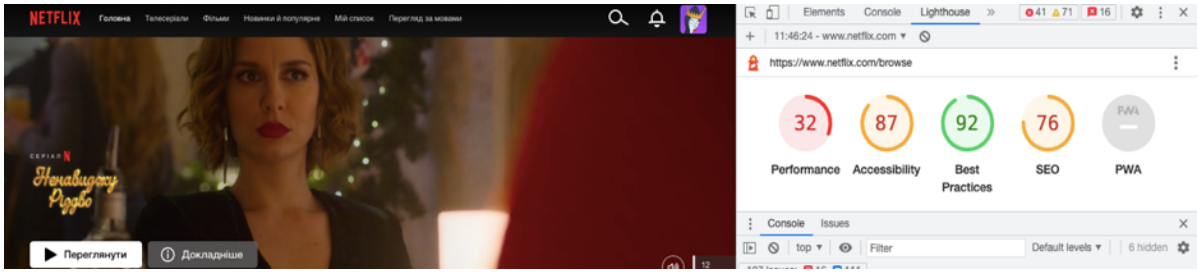

Рисунок 1.3– Показники платформи Netflix за допомогою сервісу Google Lighthouse

SweetTV – це український провайдер відео на замовлення (VOD), який також надає послуги інтернет-телебачення за технологією OTT[7].

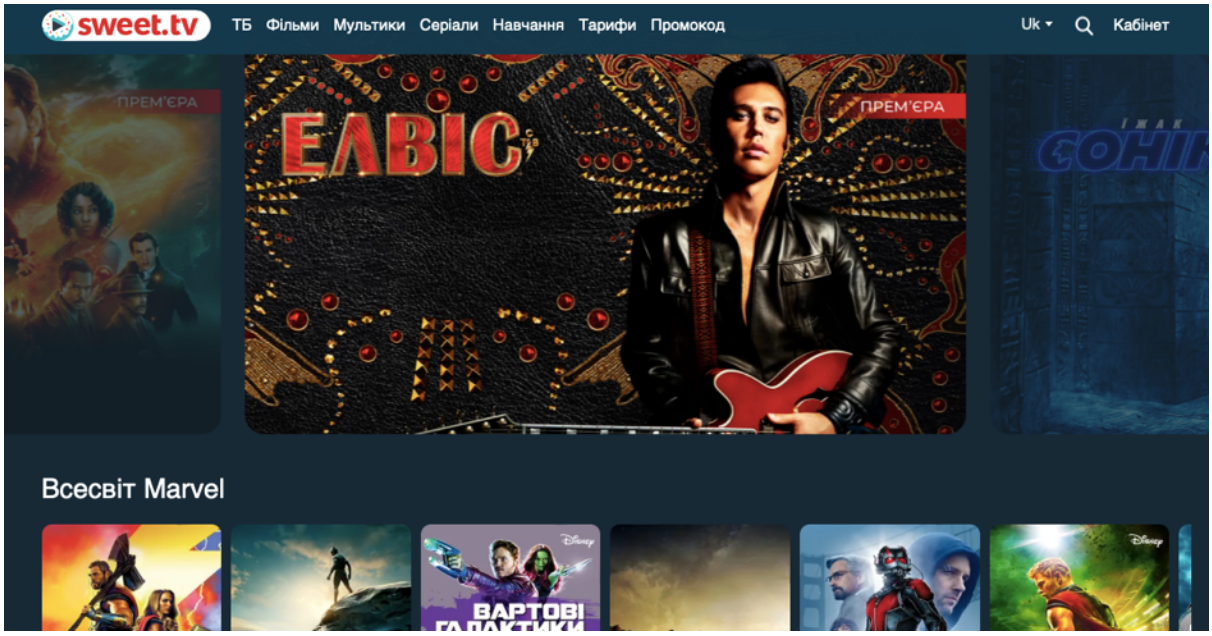

Рисунок 1.4 – Головний екран SweetTV.

Переваги SweetTV:

- Є можливість дивитися ТВ онлайн
- Без реклами;
- Широкий асортимент фільмів;
- Батьківський контроль

Недоліки SweetTV:

- Відсутність власних шоу, фільмів та серіалів;
- Відсутність сімейної підписки;

Можливості веб доступності на SweetTV:

- Закриті субтитри [CC];
- Швидкість відтворення;
- Онлайн підтримка.

Аналіз веб доступнасті за допомогою сервісу LightHause

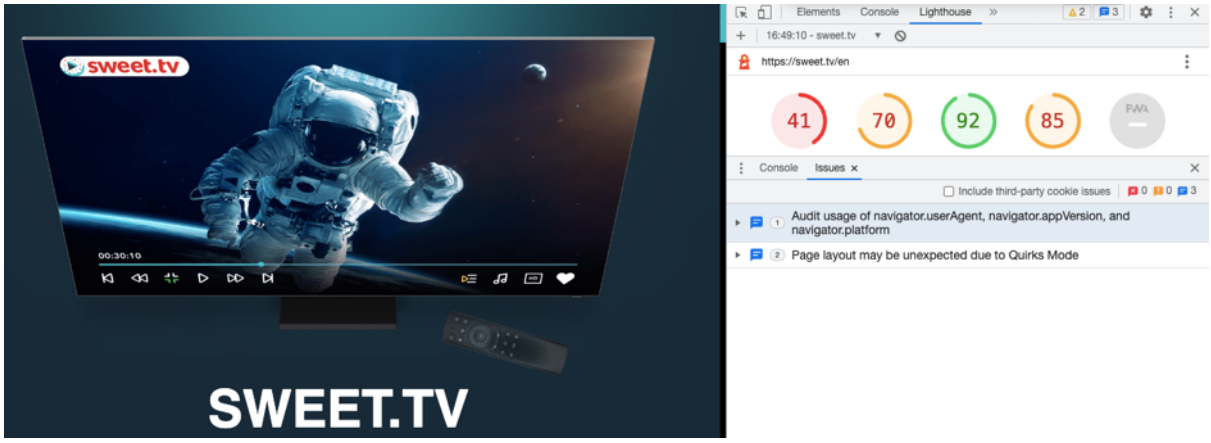

Рисунок 1.5– Показники платформи SweetTV за допомогою сервісу Google Lighthouse

1.4 Порівняльний аналіз аналогів

На основі вищезазначеної інформації була створена порівняльна таблиця з аналізом аналогів (табл. 1.1).

Таблиця 1.1

Порівняльний аналіз популярних стрімінгових платформ Україні

Критерії	Платформа «Netflix»	Платформа «SweetTV»
Багатоплатформність	+	+
Наяність власного продукту	+	-
Можлівість використання Offline	+	-
Kids mode	+	-
Батьківський контроль	-	+
Персоналізація	+	-
Сімейна підписка	+	-
Веб доступності	+	+-

Відсутність реклами	+	+
Відсутність додаткогово взимання коштів	+	+
Можливість безкоштовного перегляду медіа файлів	-	-
Trial period	-	+
Підтримка Української мови	+	+

Згідно з результатами проведеного аналізу популярних стрімінгових платформ України, можна побачити переваги та недоліки, спираючись на які буде створено платформу “WatchOnline”.

Переваги аналогів:

- Велика кількість контенту;
- Відсутність реклами;
- Використання Веб доступності;
- Багатоплатформність;
- Підтримка Української мови;

Недоліки аналогів:

- Ціна послуг;
- Відсутність контенту для безкоштовного перегляду;
- Недоліки у використанні Kids mode;
- Недоліки у використанні персоналізації;

1.5 Постановка задачі

Метою дипломної роботи є створення інформаційної технології забезпечення використовуваності та веб-доступності стримінгової платформи. Платформа матиме зрозумілий, зручний та інтуїтивний функціонал.

Завдяки цієї платформи користувачі зможуть шукати та дивитися інформацію стосовно улюблених фільмів та сералів, а також додавати їх до листа фаворитів.

Реалізація поставленої мети визначає наступні задачі:

- пошук та дослідження необхідної інформації, яка відповідає темі дипломної роботи;
- проведення аналізу аналогів;
- вибір засобів реалізації додатка;
- реалізація платформи за допомогою NextJS, ReactJS, JS, SASS, HTML;
- розгортування;
- тестування та аналіз результату;

Однією з найважливіших задач є розробка зручного, якісного, конкурентоспроможного інтерфейсу, який поєднає в собі всі найкращі практики використовуваності та веб-доступності.

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір мови програмування

Для створення стрімінгової платформи, було використано HTML, CSS, SCSS, JavaScript, ReactJS та NextJS.

HTML

Термін "HTML" розшифровується як "Hypertext Markup Language". Мова розмітки гіпертекстових, за допомогою якої створюється веб-сторінки.

У файлі HTML, програміст веб-сайту пише структуру веб-сайту. За допомогою файлу HTML текст можна структурувати та форматувати (наприклад, розмір і шрифт). Також можливо вставляти посилання на інші веб-сайти через HTML або додавати зображення, відео чи фоновий шум до тексту.

Такі браузери, як Chrome, Safari, Firefox, або Internet Explorer, зчитують файл HTML, а потім відображають його графічно.

CSS

Каскадні таблиці стилів (CSS) — це мова стилізації, яка дозволяє створювати дизайн додатків та сайтів. Такі елементи веб-сайту, як макет, колір і типографіка, можна налаштувати за бажанням за допомогою простих інструкцій, представлених у зрозумілому вихідному коді. Завдяки багаторівневому оформленню аркушів семантична структура та зміст документа залишаються абсолютно недоторканими. CSS з’явився в середині 1990-х років і зараз вважається стандартною мовою таблиць стилів у Всесвітній павутині.

Як і HTML, CSS є однією з основних мов Всесвітньої павутини. У той час як ви заповнюєте веб-сайт текстом за допомогою HTML і структуруєте його семантично, ви визначаєте дизайн цього вмісту за допомогою CSS. Хоча HTML і CSS використовуються разом, таблиця стилів CSS і елементи HTML існують окремо. Це означає, що машина може читати електронний документ

навіть без CSS. За допомогою CSS контент, відкритий у браузері, оптично готується та відображається у привабливому вигляді.

CSS є «життєвим стандартом», який постійно розвивається Консорціумом World Wide Web. З цієї причини завжди є нові функції та практичні застосування, які можна відкрити. Широко використовувана мова таблиць стилів виникла в 1990-х роках. Однак ідея використання таблиць стилів для відображення веб-вмісту вже не була абсолютно новою на той час. Однак CSS відрізнявся від інших орієнтованих на презентацію елементів, які вже існували в HTML, в одному важливому аспекті: він давав користувачам можливість визначати правила дизайну для груп елементів у кількох документах і в одній таблиці стилів.

SASS/SCSS

SCSS і SASS — це мови, які розширюють CSS (каскадні таблиці стилів). SASS означає «Syntactically Awesome Stylesheets», що означає чудовий CSS — і ось що це таке. SASS існує з 2007 року та був розроблений Hampton Catlin.

SASS є препроцесором CSS, а це означає, що хоча він спрощує CSS і розширює його багатьма функціями, SCSS і SASS зрештою повинні знову і знову компілюватися в CSS, оскільки браузер може лише інтерпретувати та розуміти CSS. Для цього вам потрібен так званий «компілятор SASS», який компілює мову SASS у звичайний CSS. Такі компілятори існують на всіх платформах і операційних системах; наприклад, як плагіни для більшості редакторів коду або, як професійно використовується в середовищі розробки, як пакет npm. За допомогою такої системи збирання, як Gulp, ви можете автоматизувати компіляцію, а також мінімізувати файли, наприклад, тощо.

Порівняно з CSS, SASS має набагато більше функцій, таких як змінні, цикли, міксини або імпорт. Також важливо, є гніздування. Це надзвичайно корисно й важливо, особливо для великих і складніших проектів. Це дозволяє уникнути необхідності писати класи більше одного разу. SASS також має явну перевагу для медіа-запитів. У той час як у CSS ви повинні згадати всі класи та елементи в медіа-запиті ще раз і таким чином записати їх двічі, ви можете

визначити медіа-запити за допомогою mixins, як у прикладі нижче, і таким чином записати їх безпосередньо в елемент. Цей метод є практичним, а також дуже динамічним, тому що якщо ви змінюєте медіа-запити, вам потрібно налаштувати лише це одне значення.

JavaScript

JavaScript — часто скорочується JS — це мова сценаріїв, яку веб-розробники можуть використовувати для створення інтерактивних веб-сторінок. Поряд з HTML і CSS, JavaScript є однією з найбільш часто використовуваних мов програмування в веб-розробці[8].

У минулому JavaScript в основному використовувався для розробки інтерфейсу. Однак він також набув популярності у розробці бекенда через такі фреймворки, як Node.js.

JavaScript використовується для різних програм в Інтернеті, наприклад для автоматично оновлюваних стрічок новин, форм, функцій пошуку та інших інтерактивних функцій. Майже всі інтерактивні функції веб-сторінок побудовані за допомогою JavaScript.

Підтримуваний усіма основними браузерами, JS широко використовується у розробці інтерфейсу. Однак він також використовується в інших сферах, таких як розробка мобільних пристроїв, розробка ігор та розробка бекенда.

ReactJS

React.js, також просто React, це бібліотека JavaScript, яка використовується для створення інтерфейсів користувача. Кожен веб-додаток React складається з багаторазово використовуваних компонентів, які утворюють частини інтерфейсу користувача — ми можемо мати окремий компонент для панелі навігації, один для нижнього колонтитула, інший для основного вмісту і так далі[10].

Багаторазові компоненти спрощують розробку, оскільки нам не потрібно повторювати код. Нам просто потрібно створити логіку та імпортувати компонент у будь-яку частину коду, де це потрібно.

React також односторінкова програма. Тож замість того, щоб надсилати запит на сервер кожного разу, коли потрібно відобразити нову сторінку, компоненти React завантажують вміст сторінки безпосередньо. Це призводить до швидшого відтворення без перезавантаження сторінки.

Ще однією перевагою React є декларативне програмування. Замість того, щоб покроково (імперативно) вказувати, як DOM має змінитися, ви лише декларативно вказуєте цільовий стан, і React вносить необхідні зміни до DOM.

Центральним елементом React є компоненти. Додаток React складається з одного або кількох компонентів. Основну ідею компонента React можна виразити відносно математично: f(p) = h. Або, кажучи простіше: компонент f приймає props (p) і повертає результат h, який виглядає як HTML, хоча технічно це JavaScript. У випадку рендерера ReactDOM на основі цього створюються фактичні вузли DOM.

Вирази JavaScript можуть з’являтися не лише як вміст компонентів React, а й у props React. Властивості (properties)React — скорочено props — можна порівняти з атрибутами HTML. Prop передається елементу або компоненту React шляхом розміщення його у початковому тегу.

З React 16.8 у лютому 2019 року команда React випустила рішення проблем із згаданими методами життєвого циклу та класами: так звані хуки. Це також дозволяє використовувати стан і побічні ефекти у функціональних компонентах. Завдяки hooks ми отримуємо багаторазову логіку стану без складних шаблонів. І без необхідності жити з недоліками міксинів, такими як неявні залежності, зіткнення імен тощо. Можна також сказати, що хуки дозволяють відокремити рендеринг компонентів інтерфейсу користувача від бізнес-логіки.

Важливо, що React hooks не представляють руйнівні зміни, а наразі представляють лише інший варіант того, як компоненти можуть бути

написані. Команда React також не втомлюється наголошувати, що наразі немає планів щодо видалення компонентів на основі класів із React. Тим не менш, можна передбачити, що одного дня дуже ймовірно буде версія React без класів. Цілком можливо, що вони будуть передані в окремий пакет, подібний до "PropTypes", який був частиною самого React до версії 15.4[10].

React hooks:

- Hook useState(). Як випливає з назви, хук useState() використовується для керування внутрішнім станом компонента. На відміну від класів, стан не об’єднується з попереднім станом; замість цього зазвичай використовуються кілька викликів useState() для різних змінних у стані.
- useEffect(). Для реалізації так званих побічних ефектів існує хук useEffect(). У випадку React побічні ефекти — це все, що безпосередньо не впливає на рендеринг компонентів інтерфейсу користувача, наприклад: консольні виходи, мережеві запити, обробники подій тощо.
- useMemo. Хук React useMemo повертає мемоізоване значення. Подумайте про запам’ятовування як про кешування значення, щоб його не потрібно було перераховувати. UseMemo запускається лише тоді, коли оновлюється одна з його залежностей. Це може покращити продуктивність. Хуки useMemo та useCallback подібні. Основна відмінність полягає в тому, що useMemo повертає мемоізоване значення, а useCallback повертає мемоізовану функцію.
- useCallback. Хук React useCallback повертає мемоізовану функцію зворотного виклику. Подумайте про запам’ятовування як про кешування значення, щоб його не потрібно було перераховувати.

  Це дозволяє нам ізолювати ресурсомісткі функції, щоб вони не запускалися автоматично при кожному рендері. Хук useCallback запускається лише тоді, коли оновлюється одна з його залежностей. Це може покращити продуктивність.

- useContext. React Context — це спосіб глобального керування станом. Його можна використовувати разом із хуком useState, щоб простіше ділитися станом між глибоко вкладеними компонентами, ніж лише з useState.
- useRef. Хук useRef дозволяє зберігати значення між візуалізаціями. Його можна використовувати для зберігання змінного значення, яке не викликає повторного відтворення під час оновлення. Його можна використовувати для прямого доступу до елемента DOM.
- UseReducer. Хук useReducer схожий на хук useState. Це дозволяє використовувати спеціальну логіку стану. Якщо ви виявите, що відстежуєте кілька частин стану, які спираються на складну логіку, useReducer може бути корисним.

Хуки — це багаторазові функції. Якщо у вас є логіка компонента, яка повинна використовуватися декількома компонентами, ми можемо відтворити її за допомогою спеціального кастомізованого хука.

Functional Components and Class Components

Функціональні компоненти: функціональні компоненти є одними з найпоширеніших компонентів, які трапляються під час роботи в React. Це просто функції JavaScript. Ми можемо створити функціональний компонент для React, написавши функцію JavaScript.

Приклад функціонального компоненту:

```
const Home=()=> {
    return <h2>I am at home!</h2>;
}
```

Компонент класу або class components: є простими класами (складаються з кількох функцій, які додають функціональність додатку).

```
class Home extends React.Component {
    render() {
     return <h2> I am at home!</h2>;
     }
   }
```

Arrow function

Arrow function позначається як функціональний вираз зі стрілкою та є компактною альтернативою традиційному функціональному виразу з деякими семантичними відмінностями та навмисними обмеженнями у використанні:

Функції зі стрілками не мають власних прив’язок до this або аргументів , і їх не слід використовувати як методи.

Arrow function не можна використовувати як конструктори. Виклик їх за допомогою нових викидає TypeError. Вони також не мають доступу до ключового слова new.target.

Стрілочні функції не можуть використовувати yield у своєму тілі та не можуть бути створені як функції-генератори.

Приклад:

```
const months = [
    'October',
    "November",
    "Desember",
  ];
  console.log(months.map(month => month.length));
```

Promisis

Promisis є основою асинхронного програмування в сучасному JavaScript. Проміс — це об’єкт, повернутий асинхронною функцією, який представляє поточний стан операції. У той час, коли обіцянка повертається абоненту, операція часто ще не завершена, але об’єкт promise надає методи для обробки можливого успіху чи невдачі операції.

React Router

Як і більшість бібліотек в екосистемі React, React Router доступний як пакет npm. Ви можете встановити його за допомогою будь-якого менеджера пакетів, наприклад npm, yarn або pnpm. У випадку npm необхідною командою є npm install react-router-dom. Тут ви вже можете помітити першу особливість, оскільки пакет, який ви встановлюєте для своєї програми, називається не react-router, а react-router-dom — і це не тому, що ім'я вже зайнято іншим пакетом. Пакет react-router також існує, і ви навіть встановлюєте його, коли встановлюєте react-router-dom.

React роутери дотримується подібного підходу, який реалізує сам React. Ядро бібліотеки не залежить від середовища, в якому працює програма. У конкретних термінах це означає, що ви використовуєте те саме ядро React Router як у React Native, так і в програмі React на основі браузера. Потім React-router-dom додає специфічні для платформи елементи для браузера та забезпечує роботу навігації на основі URL.

Загалом, навігація в SPA може працювати за допомогою двох різних механізмів: хеш-навігації та API історії. React роутер підтримує обидва варіанти. По-перше, він робить URL-шляхи чистішими, а по-друге, це інтерфейс браузера, який насправді призначений для навігації програмою.

Хеш-навігація, яку можна активувати за допомогою компонента HashRouter React роутер, використовує той факт, що браузер підтримує так звані мітки переходів. Це хеш-частина URL-адреси, спочатку призначена для переходу до певного місця в документі HTML. Перехід до такої опорної точки на тій самій сторінці не призводить до перезавантаження документа.

Це надзвичайно важливо для SPA, оскільки перезавантаження призводить до скидання поточного стану програми в пам'яті. Окрім переходу до вказаної точки прив'язки, браузер запускає подію hashchange, на яку в цьому випадку може реагувати програма або маршрутизатор. Неприємним побічним ефектом цього типу маршрутизації є те, що бажаний маршрут додається до URL-адреси після символу решітки.

Компонент BrowserRouter вирішує цю проблему, використовуючи API історії браузера замість хеш-навігації. Маршрутизатор піклується про правильне відтворення відповідних дерев компонентів і представляє поточний стан через URL-шлях браузера. Це можливо, оскільки браузер дозволяє змінювати шлях за допомогою методу History.pushState без перезавантаження браузера. API історії також дозволяє здійснювати навігацію вперед і назад за допомогою відповідних кнопок браузера. Для користувачів межа між односторінковим додатком і багатосторінковим додатком стає ще більш розмитою. З тією різницею, що зміна відображення односторінкової програми зазвичай відбувається набагато плавніше. Незалежно від того, чи ви обираєте HashRouter або BrowserRouter, основні функції React Router в обох випадках однакові. Тож ви можете перемикатися між ними, замінюючи відповідний компонент.

NextS

Next.js — це фреймворк React, який дозволяє вам створювати надшвидкі, дружні до SEO та надзвичайно прості у використанні статичні веб-сайти та веб-додатки. Next.js відомий тим, що забезпечує найкращий досвід розробників під час створення готових до виробництва програм із усіма необхідними функціями. Він пропонує гібридний статичний і серверний рендеринг, підтримку TypeScript, інтелектуальне об’єднання, попередню вибірку маршрутів і багато іншого – без додаткової конфігурації[11].

Next.js прагне надати найкращій досвід розробникові, тому він надає багато вбудованих функцій, таких як:

- Інтуїтивно зрозуміла система маршрутизації на основі сторінок (з підтримкою динамічних маршрутів);
- Попереднє відображення, як статична генерація (SSG), так і відтворення на стороні сервера (SSR) підтримуються на основі кожної сторінки;
- Автоматичне поділ коду для швидшого завантаження сторінок;

- Маршрутизація на стороні клієнта з оптимізованою попередньою вибіркою;
- Вбудована підтримка CSS і Sass, а також підтримка будь-якої бібліотеки CSS-in-JS;
- Середовище розробки з підтримкою швидкого оновлення;
- Маршрути API для створення кінцевих точок API за допомогою безсерверних функцій;

Next.js використовується в десятках тисяч робочих веб-сайтів і веб-додатків, у тому числі багатьох найбільших світових брендів.

NodeJS

Node.js/(Node) — це серверна платформа з відкритим вихідним кодом, побудована завдяки Google Chrome V8 JavaScript. Завдяки своїй здатності працювати на різних платформах і масштабованості Node став популярним варіантом для розробки веб-додатків.

2.2 Середовище розробки, плагіни та сервери

Середовище розробки

Під час розробки стрімінгової платформи, я мала нагоду ознайомитися та аналізувати різні середовище розробки(IDE). Однією з головних умов при обиранні IDE була сумісність програми з платформою MacOS.

Visual Studio Code — це потужний, але легкий редактор коду, який доступний для Windows, macOS і Linux. Він поставляється з вбудованою підтримкою JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов і середовищ виконання[12].

Плагіни

NPM є найбільшим у світі реєстром програмного забезпечення. Розробники застосовують npm для спільного використання та позичання пакетів, а багато організацій також використовують npm для керування приватною розробкою[13]

NPM складається з трьох окремих компонентів, як:веб-сайт, інтерфейс командного рядка (CLI), реєстр.

Веб-сайт використовується щоб знаходити пакети, налаштовувати профілі та керувати іншими аспектами роботи з npm. Наприклад, ви можете налаштувати організації для керування доступом до публічних або приватних пакетів.

CLI запускається з терміналу, і більшість розробників взаємодіють із npm.

Реєстр — це велика загальнодоступна база даних програмного забезпечення JavaScript і метаінформації, що його оточує.

Сервери

Під час розробки проекту ми будемо використовувати localhost. Статичний файловий сервер (localhost), який повертає файли незалежно від методу заголовка [].

Перваги використання localhost для NextJS програм:

- Автоматична компіляція та групування;
- React Fast Refresh;
- Статична генерація та рендеринг сторінок на стороні сервера/ Обслуговування статичних файлів через public/, який зіставляється з базовою URL-адресою (/)
- будь-яка програма Next.js готова до виробництва з самого початку. Дізнайтеся більше в нашій документації з розгортання.

Фінальна робота буде розмішена на Vercel

Vercel — це хмарна платформа, яка дозволяє розробникам розміщувати веб-сайти та веб-сервіси, які миттєво розгортаються, автоматично масштабуються та не потребують контролю. Компанія Vercel, заснована в 2015 році Гільєрмо Раухом, пропонує інтуїтивно зрозумілий інтерфейс користувача з мінімальною конфігурацією для розміщення генераторів статичних сайтів, таких як Gatsby або Hugo, і різних CMS, таких як Contentful, Prismic або WordPress. Vercel також є материнською компанією фреймворку Next.js — і він має багато цікавих функцій.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Налаштування проєкту

Перед початком розробки, створюємо проєкт за допомогою NextJS.

```
Last login: Sun Nov 27 14:58:27 on console
yevrud@Yevheniias-Air ~ % cd PRO
yevrud@Yevheniias-Air PRO % cd WatchOnline
yevrud@Yevheniias-Air WatchOnline % npm create next-app
✔ What is your project named? … wo
? Would you like to use TypeScript with this project? › No / Yes
```

Рисунок 3.1 – Створення NextJS проєкта

Під час розробки проекту планується багато змін та адаптації, тому для нам є дуже важливим можливість відкрити проєкт на локальному сервері — local host. Для того щоб запустити local host, ми використовуємо команду у Terminal – “npm run dev”.

Команда npm run dev — це загальна команда npm, яку можна знайти в багатьох сучасних проектах веб-додатків. Вона використовується для запуску сценарію розробника, визначеного у файлі package.json проекту. Ця команда запустить сервер розробки, щоб ви могли відкрити програму Next.js у своєму браузері[13].

```
yevrud@Yevheniias-Air new_wo-platform % npm run dev

> wo-platform@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
event - compiled client and server successfully in 824 ms (163 modules)
```

Рисунок 3.2 – Запуск команди npm run dev

За замовчуванням NextJS встановлює та створює велику кількість файлів у середні нашого проекту, вони будуть адаптуватися та змінюватися під час розробки. Також для початку роботи нам треба створити файл _documents.js. Спеціальний документ (_documents.js) може оновлювати теги <html> і <body>, які використовуються для відтворення сторінки. Цей файл відображається лише на сервері, тому обробники подій, такі як onClick, не можна використовувати в ньому.

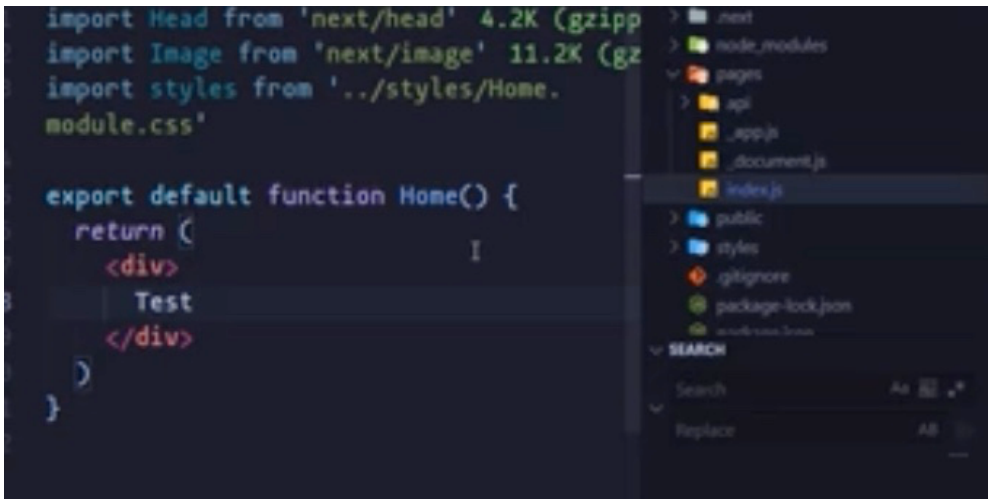

Рисунок 3.3 – Проєкт створений автоматично за допомогою NextJS

Також, перед початком розробки, встановлюємо шрифти які будуть використовуватися при реалізації проєкта . Для цього ми використовуємо бібліотеку Google fonts.

```
import Document, { Html, Head, Main, NextScript } from "next/document";

class MyDocument extends Document {
  static async getInitialProps(ctx) {
    const initialProps = await Document.getInitialProps(ctx);
    return { ...initialProps };
  }
  render() {
    return (
      <Html>
        <Head>
          <link rel="preconnect" href="https://fonts.gstatic.com" />
          <link
            rel="preload"
            as="style"
            href="https://fonts.googleapis.com/css2?family=Nunito+Sans:ital,wght@0,20
          />
```

Рисунок 3.4 – Додавання бібліотеки Google fonts до проєкту

Після додавання шрифтів, ми переходимо до кроку налаштування SASS. Для того щоб налаштувати SASS, для початку треба зупинити наш локальний сервер, це можна зробити командою – Control + C. Наступний крок, це запуск команди “npm install sass” у терміналі. Next.js дозволяє імпортувати Sass за допомогою розширення .scss і .sass. Можливо використовувати Sass на рівні компонента через модулі CSS і розширення .module.scss або .module.sass.

Також, під час розробки, нам знадобиться CSS reset. CSS reset (або «Скидання CSS») — це короткий, часто стислий (зменшений) набір правил

CSS, який скидає стиль усіх елементів HTML до узгодженого базового рівня[14].

Кожен браузер має власну таблицю стилів «агента користувача» за замовчуванням, яку він використовує, щоб зробити веб-сайти без стилів більш читабельними. Наприклад, більшість браузерів за замовчуванням роблять посилання синіми, а відвідані посилання – фіолетовими, надають таблицям певну межу та відступи, застосовують змінні розміри шрифту до H1, H2, H3 тощо та певну кількість відступів майже до всього.

Використовуючи скидання CSS, автори CSS можуть змусити кожен браузер скинути всі його стилі до нуля, таким чином уникаючи відмінностей між браузерами, наскільки це можливо.

Наступним кроком при підготовці до створення нашої стрімінгової платформи є додавання іконок. Font Awesome — це інтернет-бібліотека значків і набір інструментів, якими користуються мільйони дизайнерів, розробників і творців контенту.

3.2 Програмна реалізація

Наш майбутній проєкт буде складатися з таких елементів:

- Сторінка авторизації;
- Сторінка створення нових користувачів;
- Header;
- Бокова навігація сторінками;
- Головна сторінка;
- Окрема сторінка з інформацією про фільм;
- Секція з фаворитами;
- Пошук.

Першим кроком створення стрімінгової платформи буде створення схеми потоку користувача.

Потік користувача або User Flow— це шлях, який проходить користувач — від точки входу до кінцевої мети — для виконання завдання в інтерфейсі

користувача. Точкою входу може бути домашня сторінка вашого сайту та кінцева мета додавання сподобавшихся фільмів до листа фаворитів. Потік користувача визначає всі кроки навігації на шляху користувача для досягнення мети[15].

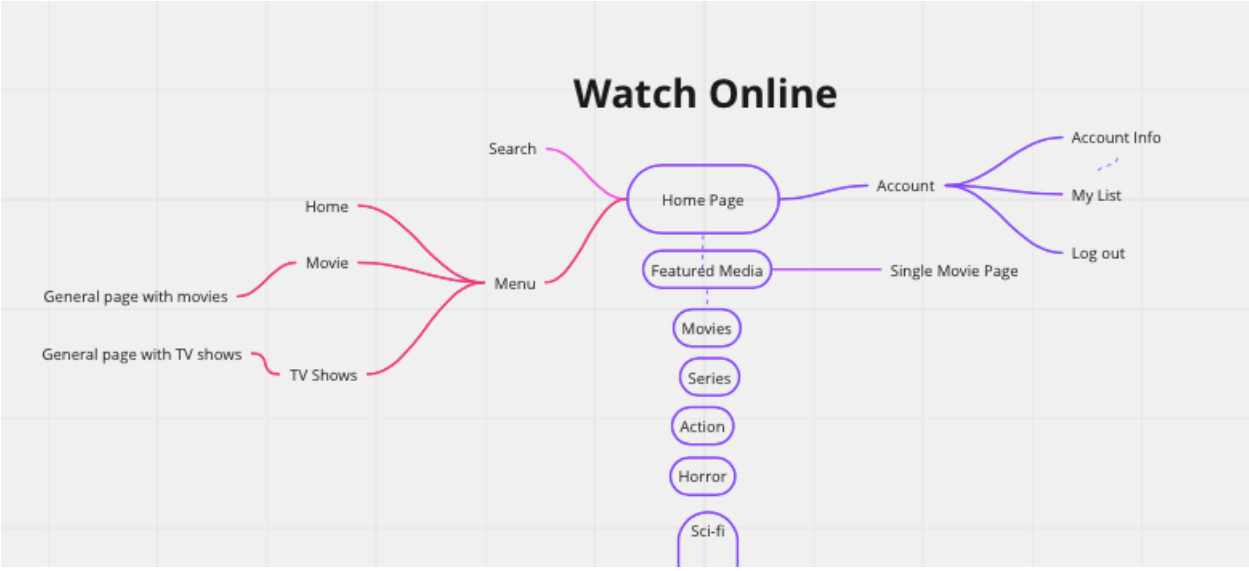

Схема 3.1 - Внутрішня структура веб-сайту

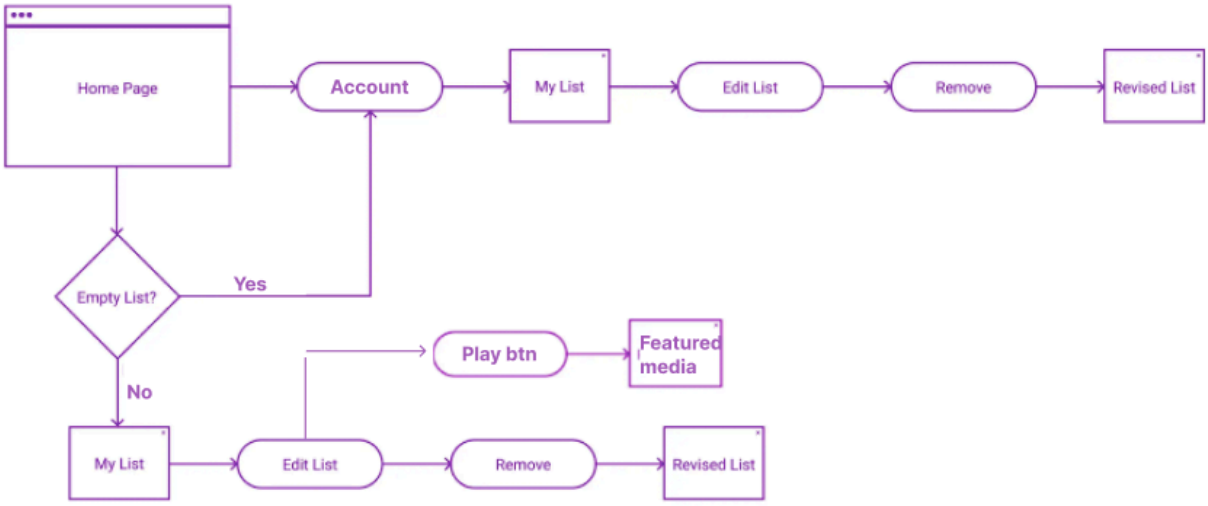

Схема 3.2— Web Flow сторінки My List

3.2.1 Сторінка авторизації

Сторінка авторизації, це сторінка з якої кожен користувач платформи зможе перейти до персонального простору, де він зможе продивитися фільми, інформацію про них, та додати фільми які сподобались до листа фаворитів.

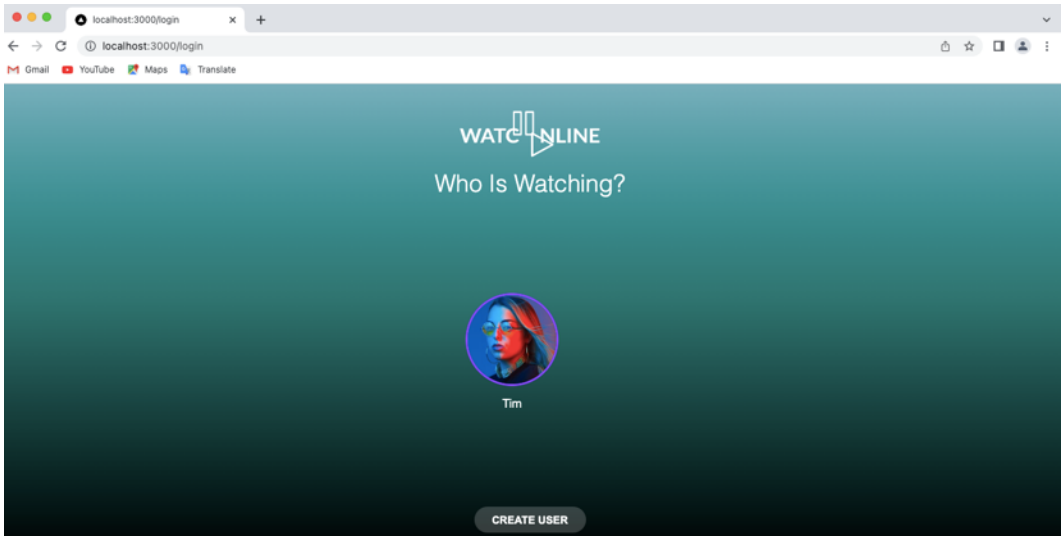

Рисунок 3.5 – Сторінка авторизації

Для початку створюємо папку з компонентами нашого проєкта, таким чином він матиме більш структурований вигляд.

```
import Login from '../components/UI/Login/Login'

const LoginPage = () => {
  return (
    <Login />
  )
}
export default LoginPage;
```

Рисунок 3.6 – файл pages -> login.js

У файлі pages -> login.js встановлюємо посилання на наш компонент Login. Компонент буде складатися з 2 файлів – scss та в js. У яких ми будемо прописувати логіку та зовнішній вигляд компонента.

При запуску цієї сторінки є фон, який буде використовуватись продовж всього проєкта.

```
.login-user{
    background: linear-gradient(180deg,
        rgb(120, 175, 188) 0%,
        rgb(10, 117, 112) 45%,
        rgb(0, 0, 0) 100%);
    height: 100vh;
    display: flex;
    justify-content: space-between;
    align-items: center;
    flex-direction: column;
```

Градієнт було отримано завдяки сервісу CSS gradient.

Також, ця сторінка буде складатися з:

- аватара та ім’я власника акаунту, натиснувши на яку ми зможемо перейти до персональної сторінки;
- Кнопки “Create user”.

3.2.2 Сторінка створення нових користувачів

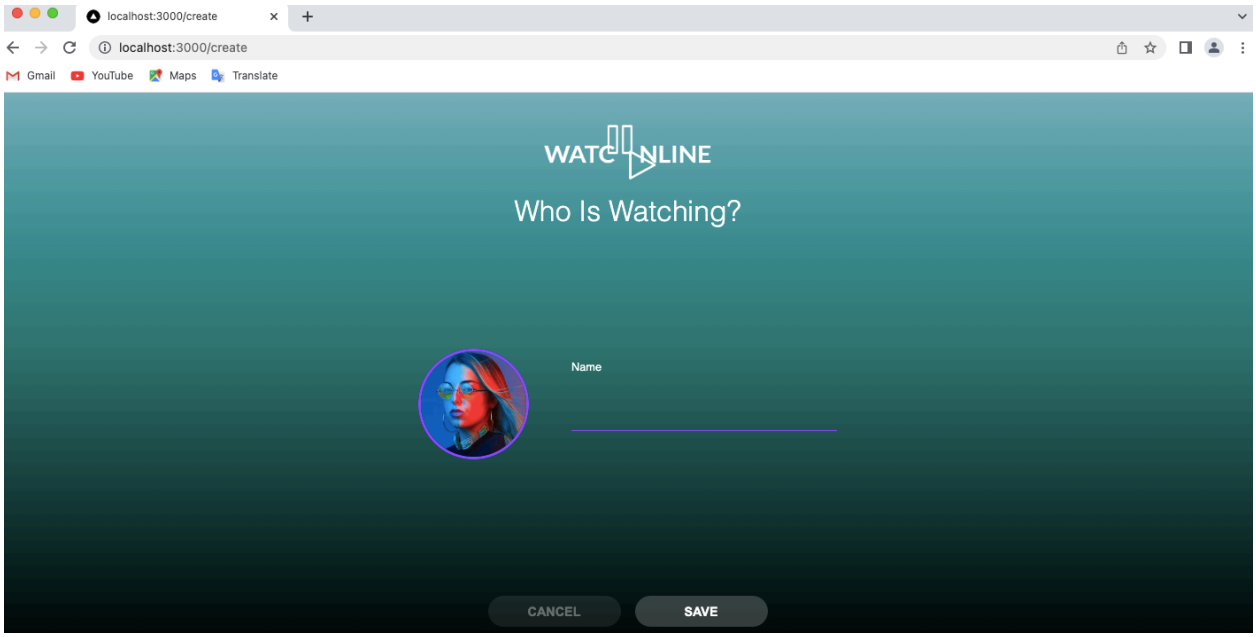

Рисунок 3.7 – Сторінка створення нових користувачів

Перейшовши до вкладки Create user, користувач зможе обрати собі ім'я та у майбутньому кастомізувати фон сторінки. Також, після додавання власної

інформації, користувач зможе зберегти аккаунт або видалити додану інформацію.

Після натискання кнопки “save”, інформацію користувача буде додано до local storage.

Для початку, до файлу create.js імпортуємо local storage. Це можна зробити командою “import ls from ‘local-storage’” та “import {u4} from ‘uuid’”.

Після цього створюємо функцію, яку буде спрацьовувати під час настискання кнопки “save”.

```
const saveUser = () => {
    let users = [],
    user;

    if(ls('users') < 1) {

      user = {
        id: v4(),
        user: globalState.user,
        myListID: []
      }
      users.push(user)
      ls('users', users)
      router.push('/login')

      console.log('users:', users)
      console.log('lsusers', ls('users'))
    } else {
      users = ls('users')
      user = {
        id: v4(),
        user: globalState.user,
        myListID: []
      }
      users.push(user)
      ls('users', users)
      console.log('users:', users)
      console.log('lsusers', ls('users'))
      router.push('/login')

    }
}
```

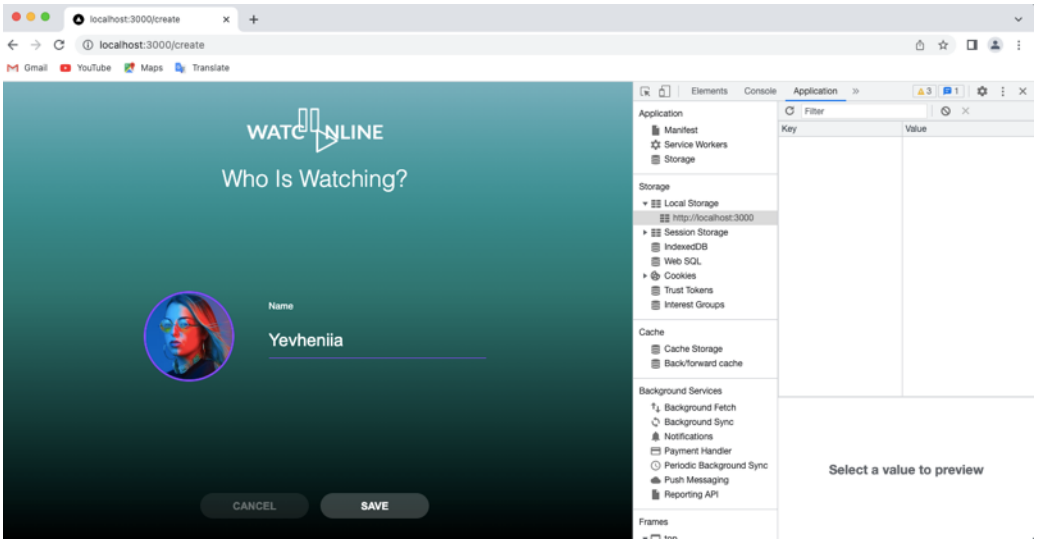

Рисунок 3.8 – Додавання користувача до local-storage

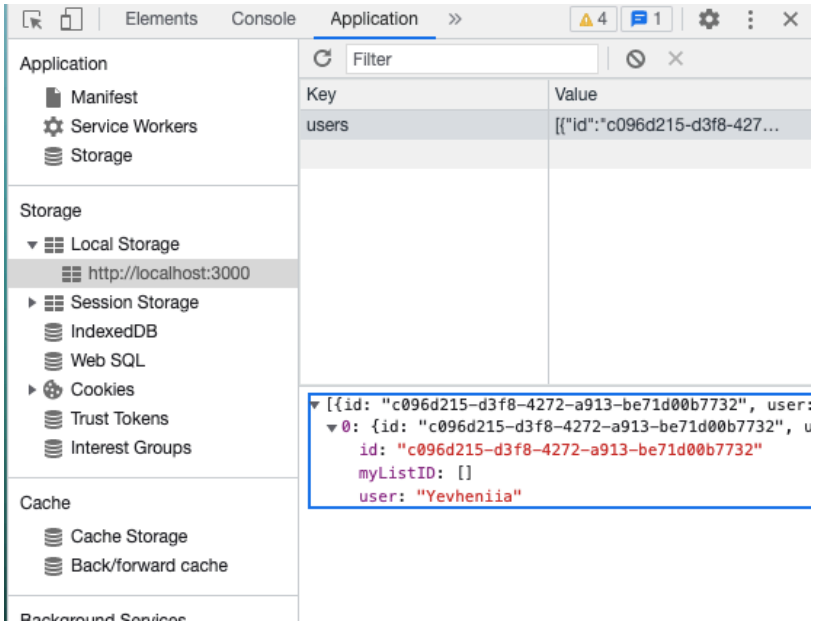

Рисунок 3.9 – Відображення у local-storage

Під час реалізації сайту, ми використовуємо React routing. За допомогою routing та GlobalState, ми перевіряємо якщо користувач вже створив аккаунт, тільки тоді він може перейти до основних сторінок. Якщо ж користувач не був авторизований раніше, то йому буде запропоновано пройти реєстрацію.

```
const AuthCheck = (component) => {
    const router = useRouter();
    const {hasMounted} = useMounted();
    let activeUID = ls('activeUID');
    let users = ls('users') !== null ? ls('users') : [];

    useEffect(() => {
      if( activeUID === null || users.length < 1) {
        router.push('/create')
      }
    }, [])
```

3.2.3 Header

Для початку створюємо компонент header. Сам “заголовок” буде відображатися під час перебування користувача на головній сторінці та на окремій сторінці фільму.

Рисунок 3.10 – Header

Іконка випадаючого списку та іконка пошуку додаються за допомогою Font Awesome:

- Кнопка меню <i className="fas fa-bars" />
- Кнопка пошуку <i className="fas fa-search" />

Також, до обох кнопок додаємо подію onClick та глобальний стан. Основною метою глобального стану є розподіл стану між декількома компонентами.

```
onClick={() => globalState.setSideNavOpenAction(true)}
```

Таким чином, при натисканні кнопки меню на головному екрані з’явиться наше бокове меню. А при використанні події onClick та глобального стану стосовно кнопки пошуку, користувач переде до меню пошуку.

3.2.4 Бокова навігація сторінками

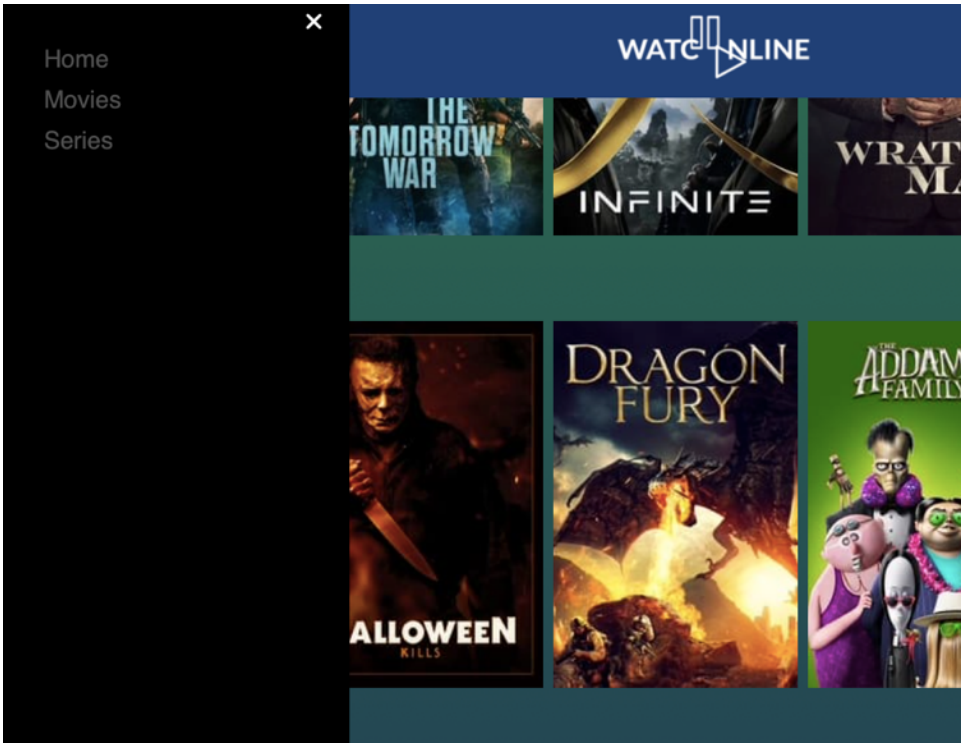

Рисунок 3.11 – Бокова навігація сторінками

За допомогою бокової навігації користувач має змогу:

- Повернутися до головної сторінки;
- Перейти до сторінки з фільмами;
- Перейти до сторінки з серіалами

Для того щоб закрити бокову навігацію, користувач має кнопку close, при натисканні якої глобальний стан бокового меню, отримує статус false.

```
<div className="side-nav__close-button"
onClick={() => globalState.setSideNavOpenAction(false)}>
        <i className="fas fa-times" />
```

3.2.5 Головна сторінка

Головна сторінка стрімінгової платформи “WatchOnline”, буде складатися з 2 частин:

- Featured Media. Це розділ у якому буде розміщено рекламу найпопулярнішого кіно у момент використання;

- MediaRow. Вертикальні компоненти з фільмами розбитими за жанром.

Featured Media

Під час створення додатку, Enola Holms, стала найпопулярнішим фільмом за версією Netflix. Тому рекламу саме цього фільму планується додати до головної сторінки. В майбутньому реклама буде змінюватися автоматично.

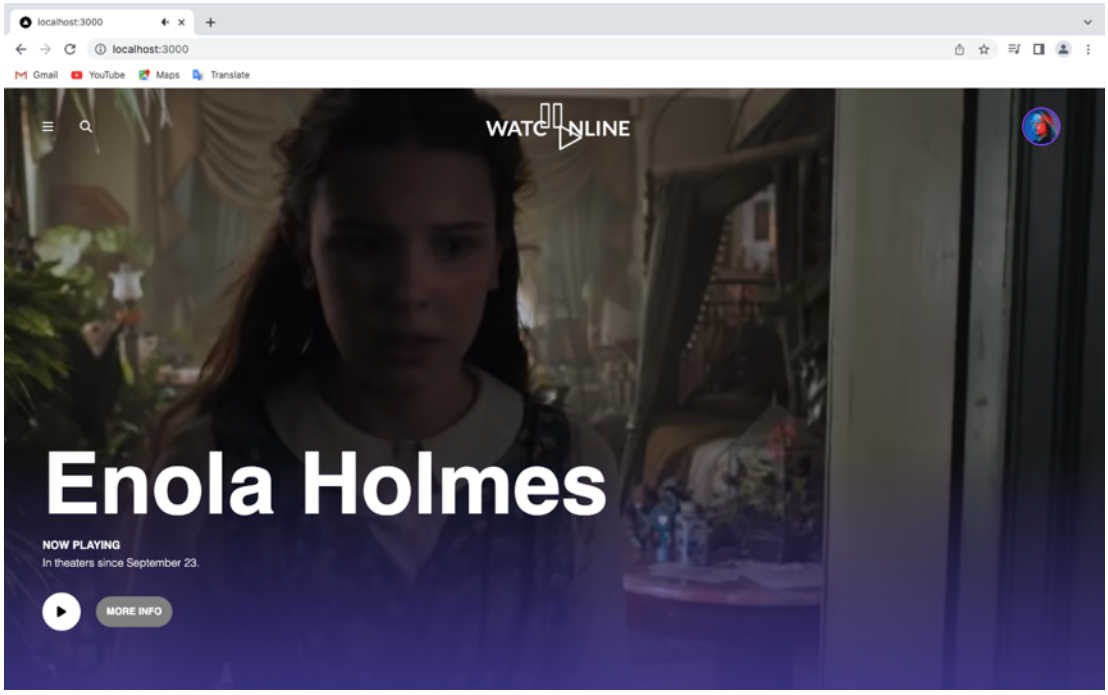

Рисунок 3.12 – Featured Media

Коли користувач дивиться рекламу, також він має змогу перейти до сторінки з інформацією про фільм натиснувши на кнопки “play” чи “more info”.

```
<FeaturedMedia
mediaUrl="https://www.youtube.com/embed/1d0Zf9sXlHk?autoplay=1&loop=1
title="Enola Holmes"
location="In theaters since September 23."
linkUrl="/movie/497582"
type="front"
mediaType={'movie'}
mediaId={497582}
/>
```

MediaRow

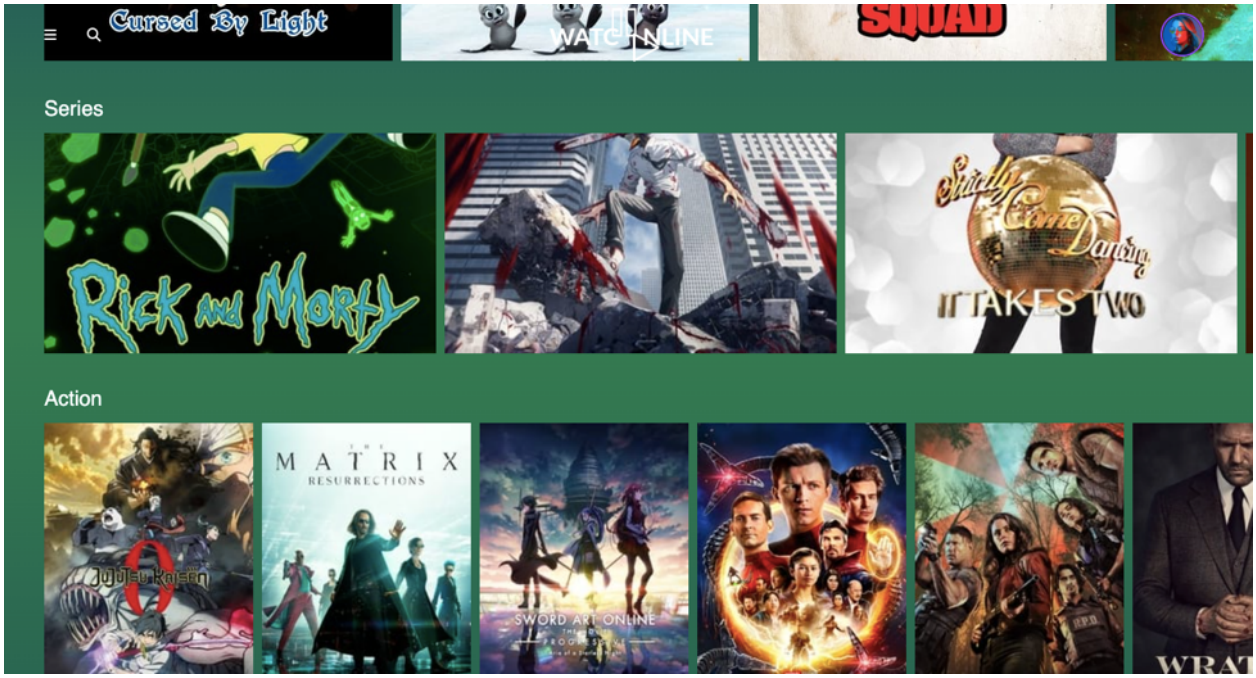

Рисунок 3.13 – MediaRow

MediaRow складається з таких з таких частин як:

- Movies;
- Action;
- Horror;
- Animation;
- Sci-fi

Головною функцією MediaRow є демонстрація афіш фільмів за категоріями. Користувач має змогу продавитися афіши, та перейти на сторінки зацікавивший його фільмів.

Ми створюємо петлі для кожного жанру фільмів. Кожна петля буде відображатися по-різному, залежно від типу.

```
const Poster = (props) => {
    const posterSize = (type) => {
        if (props.type === "large-vertical") {
            return "400";
        }
        if (type === "small-vertical") {
            return "185";
        }
        if (type === "large-horizontal") {
            return "500";
        }
        if (type === "small-horizontal") {
            return "342";
        }
    };
```

Перед тим як під’єднати наш api з фільмами, ми створимо дефолтне зображення, яке буде відображатися при загрузці афіши — “Skeleton”.

```
const showPosters = (type) => {
    return loadingData
        ? loopComp(<Skeleton />, 10)
        : movies.map((movie) => {
            return <Poster key={movie.id}
            movieData={movie}
            type={type} mediaType={props.mediaType} />;
```

Також, під час скролінгу сторіки, ми використовуєсо lasy loading.

Відкладене завантаження або lasy loading — це практика затримки завантаження або ініціалізації ресурсів чи об’єктів, поки вони дійсно не знадобляться для підвищення продуктивності та економії системних ресурсів.

```
<LazyLoad
    offset={-200}
    placeholder={<Placeholders title="Movies"
    type="large-vertical" />}>
    <MediaRow
        title="Movies"
        type="large-vertical"
        endpoint="discover/movie?sort_by=popularity.desc&primary_
    />
</LazyLoad>
```

Підключення API

Для цього проєкта було обрано The Movie DB API.

The Movie DB API — це ресурс для будь-яких розробників, які хочуть інтегрувати дані фільмів, телешоу та акторів разом із плакатами чи фан-артом фільмів. Це безкоштовна база даних, яку редагує спільнота.

Після створення персоналізованого ключа, залишається тільки встановити axios за допомогою команд npm.

Axios js — це HTTP-бібліотека на основі promisis, яка дозволяє нам використовувати службу API. Вона пропонує різні способи створення HTTP-запитів, наприклад GET, POST, PUT і DELETE. Axios можна використовувати в будь-якому фреймворку JavaScript, і після встановлення він дає змогу вашій програмі взаємодіяти зі службою API.

```
useEffect(() => {
    axios
        .get(
            `https://api.themoviedb.org/3/${props.endpoint}
            &api_key=64e0f967a0e15cb2abcefb00f295a079&language=en-US`,
        )
```

Рисунок 3.14 – Результати запиту до API

Окрема сторінка з інформацією до фільма

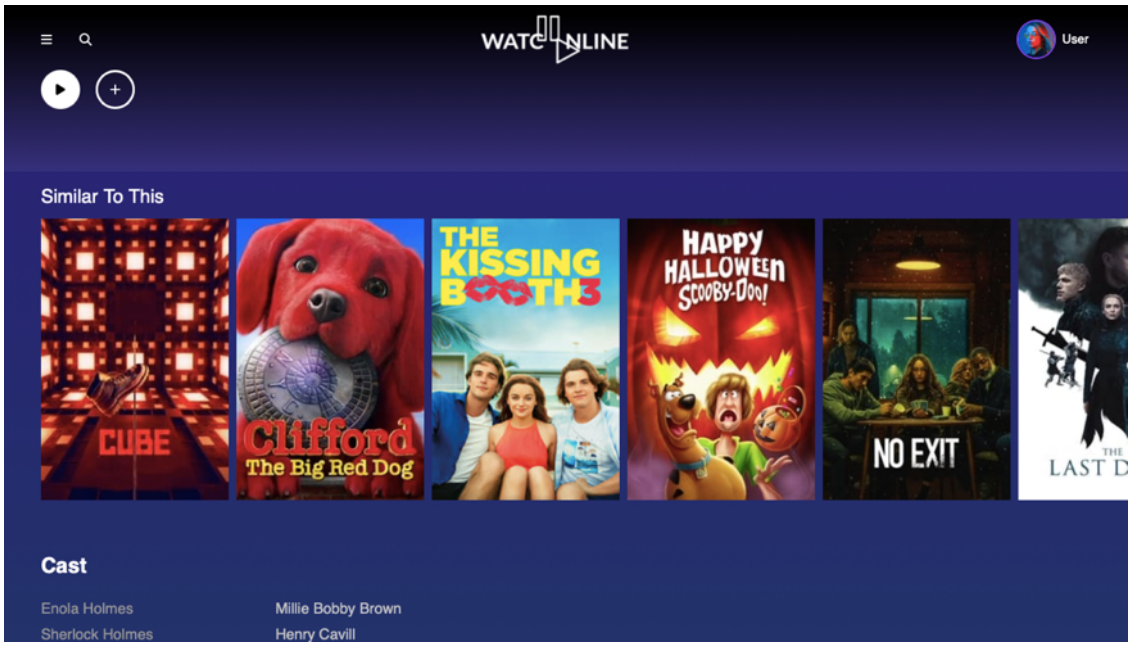

Рисунок 3.15 – Окрема сторінка з інформацією до фільма

Окрема сторінка складається з таких частин, як:

- Обкладинка фільму;
- Кільце з схожими фільмами;
- Cast and crew

```
<MainLayout>
        <FeaturedMedia
            title={props.query.mediaType === 'movie' ? props.mediaData.title : props.mediaData.name}
            mediaUrl={`https://image.tmdb.org/t/p/original${props.mediaData.backdrop_path}`}
            location="In theaters since September 23."
            linkUrl="/movies/id"
            type="single"
            mediaType={props.query.mediaType}
            mediaId={props.query.id}
        />
        <LazyLoad
            offset={-400}
            placeholder={<Placeholders title="Movies" type="large-vertical" />}>
            <MediaRow
                updateData={props.query.id}
                title="Similar To This"
                type="small-vertical"
                mediaType={props.query.mediaType}
                endpoint={`${props.query.mediaType === 'movie' ? 'movie' : 'tv'}/${props.query.id}/similar?`}
            />
        </LazyLoad>
        <CastInfo mediaId={props.query.id} mediaType={props.query.mediaType} updateData={props.query.id}/>
    </MainLayout>,
);
```

Секція з фаворитами

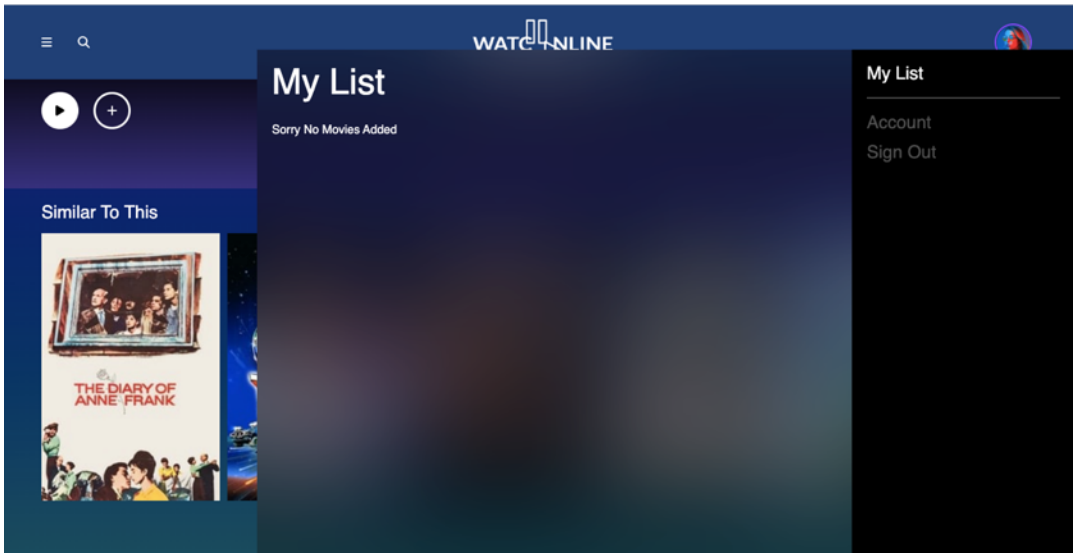

Рисунок 3.16 – Секція з фаворитами без доданих фільмів

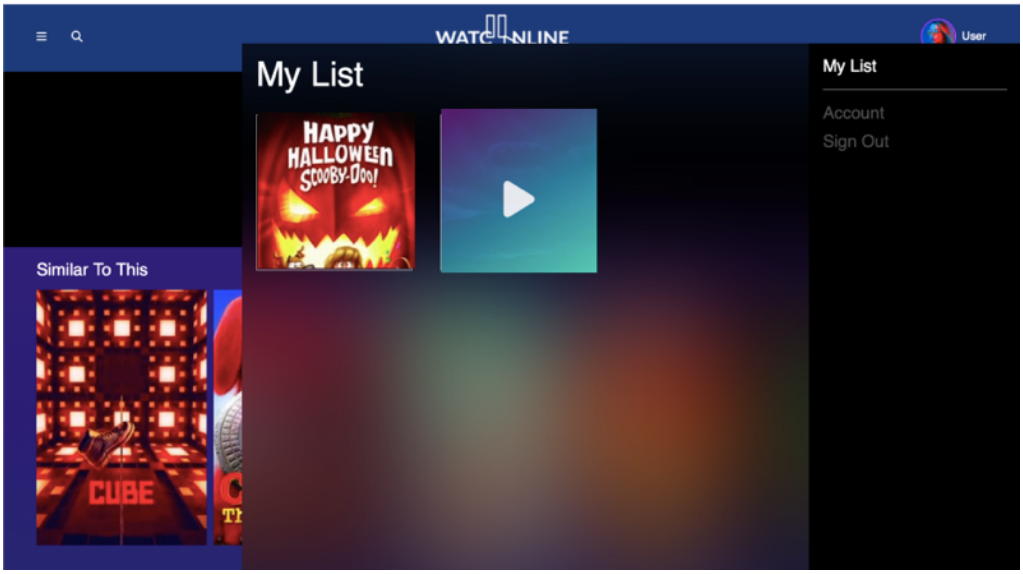

Рисунок 3.17 – Секція з фаворитами з доданими фільмами

Після того як користувач перейшов до сторінки з окремим фільмом, він має змогу додати його свого листа. Це можна зробити за допомогою кнопки “+”.

```
const clickedAdd = (props) => {
    globalState.addToList({mediaId: props.mediaId, mediaType: props.mediaType, mediaUrl: props.mediaUrl})
  }
…
const addToList = (video) => {
    let myList;
    if(ls('myList') !== null) {
      myList = ls.get('myList')
      myList.push(video)
      ls.set('myList', myList)
      setWatchList(myList)
    } else {
      ls.set('myList', [video])
    }
  const removeFromList = (videoId) => {
    let myList = ls('myList')
    myList = myList.filter((item) => item.mediaId != videoId)
    ls.set('myList', myList)
    setWatchList(myList)
  }
```

Після того як користувач додав фільм до листа фаворитів, він має змогу перейти улюблено фільма чи видалити його з листа фаворитів.

Пошук

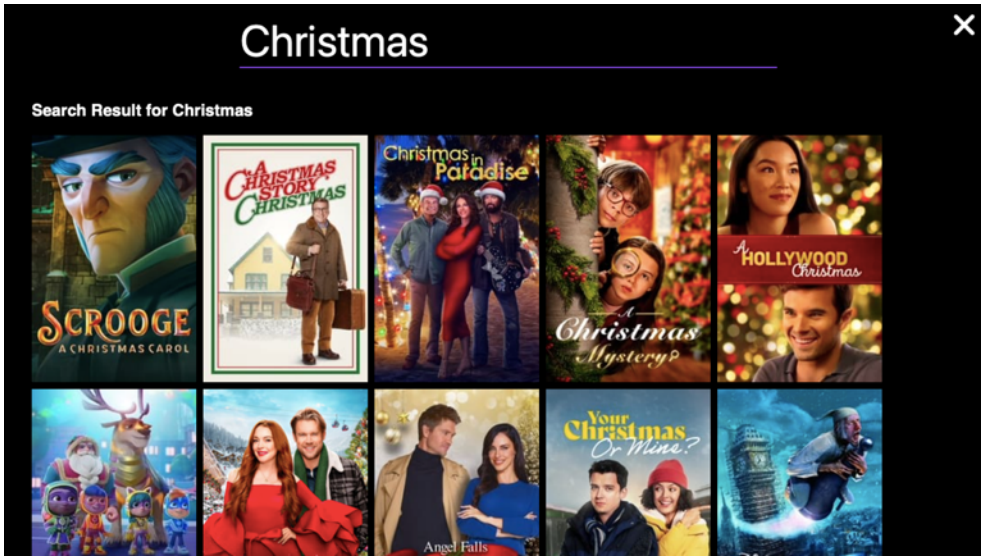

Рисунок 3.18 – Пошук

Користувачі мають можливість не тільки знати улюблений фільм завдяки переліку, а й завдяки функції “Пошук”.

```
const SearchModal = (props) => {
    const globalState = useStateContext();
    const [popData, setPopData] = useState([]);
    const [searchData, setSearchData] = useState([]);
    const [showResults, setShowResults] = useState(false);
    const [text, setText] = useState("");
    const router = useRouter();

    useEffect( () => {
        const fetchData = async () => {
        try {
            let popData = await axios.get(
                `https://api.themoviedb.org/3/discover/movie?primary_release_year=2022&api_key=64e0f967a0e15cb2abcefb
            );
            setPopData(popData.data.results.filter((item, i) => i < 14));

            setShowResults(false);
            console.log("popdata", popData.data.results);
        } catch (error) {
            console.log(error);
        }
    }
    }, []);
```

Перед початком пошуку, користувач має нагоду побачити список найпопулярніших фільмів. Також користувач моє змогу знайти фільми за ключовими словами:

```
useEffect(() => {
    if (globalState.searchOpen) {
        document.body.style.overflowY = "hidden";
    } else {
        document.body.style.overflowY = "auto";
    }
}, [globalState.searchOpen]);

const handleInput = async (e) => {
    try {
        setText(e.target.value);
        let searchData = await axios.get(
            `https://api.themoviedb.org/3/search/multi?query=${e.target.value}&api_key=64e0f967a0e15cb2abcefb00f295a
        );
        setSearchData(
            searchData.data.results.filter(
                (item, i) => item.media_type === "tv" || item.media_type === "movie",
            ),
        );
        setShowResults(true);
    } catch (error) {
        console.log(error);
    }
};
```

Відображення даних у полі Пошук

```
const PopularResults = (props) => {
    return props.popData.map((item, index) => {
    return (
    <div
    key={index}
    className="search-modal__poster"
    onClick={() => props.clickedposter("popular", item.id)}>
    <img src={`https://image.tmdb.org/t/p/w185${item.poster_path}`} />
    <div className="search-modal__top-layer">
    <i className="fas fa-play" />
    </div>
    </div>
```

3.3 Розгортання

Для розгортання було обрано сервіс Versel. Платформа Vercel створена творцями Next.js, та призначена для розгортання програм Next.js.

Vercel CLI дозволяє розгортати ваші проекти безпосередньо з інтерфейсу командного рядка (CLI). Ви можете використовувати цей метод розгортання незалежно від того, підключено ваш проект до репозиторію Git чи ні.

Для нового проекту перше розгортання попросить зв’язати ваш локальний каталог із проектом Vercel. Це робиться за допомогою команди vercel у вашому локальному каталозі проекту.

```
[yevrud@Yevheniias-Air new_wo-platform % vercel                                ]
> UPDATE AVAILABLE Run `npm i -g vercel@latest` to install Vercel CLI 28.8.0
> Changelog: https://github.com/vercel/vercel/releases/tag/vercel@28.8.0
Vercel CLI 28.7.0
🔍  Inspect: https://vercel.com/evgrud9-gmailcom/watchonline/ERgFNbyHvTUewGi1TWo
JS8MUPKqd [4s]
✅  Preview: https://watchonline-evgrud9-gmailcom.vercel.app [22s]
📝  To deploy to production (watchonline.vercel.app), run `vercel --prod`
yevrud@Yevheniias-Air new_wo-platform %
```

Рисунок 3.19 – Розгортання на сервісі Vercal за допомогою CLI

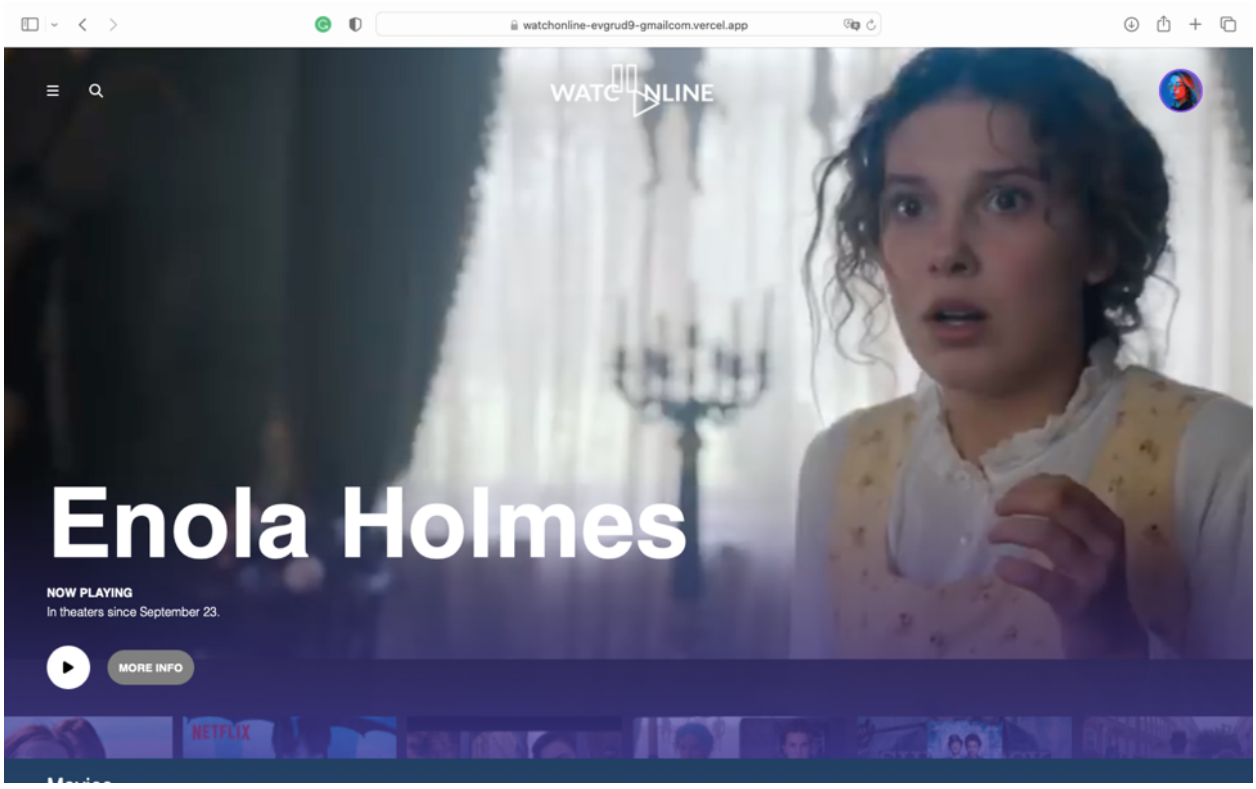

Рисунок 3.20 – Розгорнений сайт Watch Online

3.4 Тестування за допомогою сервісу Google Lighthouse

Як зазначалося вище, Google Lighthouse – це автоматизований інструмент із відкритим кодом для вимірювання якості веб-сторінок.

Рисунок 3.21 – Тестування за допомогою сервісу Google Lighthouse

Після тестування за допомогою сервісу, ми бачимо що наш додатк має достатньо високий рівень роботи та доступності.

ВИСНОВКИ

В процесі роботи, було розроблено інформаційну технологію забезпечення використовуваності та веб-доступності стримінгової платформи. Платформа має зрозумілий, зручний та інтуїтивний функціонал.

Під час виконання роботи було реалізовано:

- пошук та дослідження необхідної інформації, яка відповідає темі дипломної роботи;
- проведення аналізу аналогів;
- вибір засобів реалізації додатка;
- реалізація платформи за допомогою NextJS, ReactJS, JS, SASS, HTML;
- розгортування;
- тестування та аналіз результату;

Однією з найважливіших задач є розробка зручного, якісного, конкурентоспроможного інтерфейсу, який поєднає в собі всі найкращі практики використовуваності та веб-доступності.

Розроблений платформа має безліч переваг:

- адаптивність;
- використовуються практики використовуваності;
- використовуються практики веб-доступності;
- наповненість

Завдяки цієї платформи користувачі можуть шукати та дивитися інформацію стосовно улюблених фільмів та сералів, а також додавати їх до листа фаворитів.

СПИСОК ЛІТЕРАТУРИ

1. Video Streaming Market [Електронний ресурс]. — Режим доступу: https://www.precedenceresearch.com/video-streaming-market
2. Ендрю Кіркпатрік, Майкл Р. Беркс Web Accessibility, 2006 — 354 с.
3. User Friendly Features [Електронний ресурс]. — Режим доступу: https://www.bsgroup.eu/blog/top-12-user-friendly-features-for-your-streaming-platform/
4. Девід Ардіті Streaming Culture: Subscription Platforms And The Unending Consumption Of Culturея, 2021. —. 174 с.
5. Netflix accessibility features [Електронний ресурс]. – Режим доступу: https://www.makeuseof.com/how-to-use-netflix-accessibility-features/
6. Google LightHause [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Google_Lighthouse
7. SweetTV [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Sweet.tv
8. Джон Резвіг, Беєр Бібо, Іосип Марас Secrets of the JavaScript Ninja. Друге видання, 2017 — 418с.
9. Anthony Accomazzo, Nate Murray, Ari Lerner Full stack React, 2020 — 648 с.
10. Даніель Багл Learn React Hooks, 2019 — 426 с.
11. Давід Чой Full-Stack React, TypeScript, and Node: Build cloud-ready web applications using React 17 with Hooks, 2020 — 648 с.
12. Гектор Уріель Перез Рохас Hands-On Visual Studio, 2022 — 350 с.
13. Кирило Коньшин Next.js Quick Start Guide: Server-side rendering done right, 2018 — 166с.
14. Кейт Дж. Грант CSS in Depth, 2018 — 472с.
15. Better Data Visualizations: A Guide for Scholars, Researchers, and Wonks, , 2021 — 449с.

ДОДАТКИ

Додаток А. index.js

```
import Head from "next/head";
import { useEffect } from "react";
import { useStateContext } from "../components/WOprovider";
import { useRouter } from "next/router";
import MainLayout from "../components/Layouts/MainLayout";
import FeaturedMedia from "../components/UI/FeaturedMedia/FeaturedMedia";
import MediaRow from "../components/UI/MediaRow/MediaRow";
import AuthCheck from "../components/AuthCheck";
import LazyLoad from "react-lazyload";
import Placeholders from "../components/UI/Placeholders/PlaceHolders";

export default function Home() {
    const globalState = useStateContext();
    const router = useRouter();
    useEffect(() => {}, []);
    return AuthCheck(
        <MainLayout>
            <FeaturedMedia
                mediaUrl="https://www.youtube.com/embed/1d0Zf9sXlHk?autoplay=1&loop=1&start=16"
                title="Enola Holmes"
                location="In theaters since September 23."
                linkUrl="/movie/497582"
                type="front"
                mediaType={'movie'}
                mediaId={497582}
            />
            <LazyLoad
                offset={-200}
                placeholder={<Placeholders title="Movies" type="large-vertical" />}>
                <MediaRow
                    title="Movies"
                    type="large-vertical"
                    endpoint="discover/movie?sort_by=popularity.desc&primary_release_year=2021"
                />
            </LazyLoad>

            <LazyLoad
                offset={-200}
                placeholder={<Placeholders title="Series" type="small-horizontal" />}>
                <MediaRow
                    title="Series"
                    mediaType="series"
                    type="small-horizontal"
                    endpoint="discover/tv?primary_release_year=2022"
                />
            </LazyLoad>
```

```
            <LazyLoad
                offset={-200}
                placeholder={<Placeholders title="Movies" type="small-vertical" />}>
                <MediaRow
                    title="Action"
                    type="small-vertical"
                    endpoint="discover/movie?with_genres=28&primary_release_year=2022"
                />
            </LazyLoad>

            <LazyLoad
                offset={-200}
                placeholder={<Placeholders title="Movies" type="small-vertical" />}>
                <MediaRow
                    title="Horror"
                    type="small-vertical"
                    endpoint="discover/movie?with_genres=27&primary_release_year=2022"
                />
            </LazyLoad>

            <LazyLoad
                offset={-200}
                placeholder={<Placeholders title="Movies" type="large-horizontal" />}>
                <MediaRow
                    title="Animations"
                    type="large-horizontal"
                    endpoint="discover/movie?with_genres=16&primary_release_year=2022"
                />
            </LazyLoad>

            <LazyLoad
                offset={-200}
                placeholder={<Placeholders title="Movies" type="large-vertical" />}>
                <MediaRow
                    title="Sci-fi"
                    type="small-vertical"
                    endpoint="discover/movie?with_genres=878&primary_release_year=2022"
                />
            </LazyLoad>
        </MainLayout>,
    );
}
```

Додаток Б. component login.js

```
import Head from 'next/head'
import {useState, useEffect} from 'react';
import { useStateContext } from '../../WOprovider';
import { useRouter } from 'next/router';
import ls from 'local-storage';
import { useMounted } from '../../Hooks/useMounted';


const Login = () => {
  const globalState = useStateContext();
  const router = useRouter();
  const [loadingUsers, setLoadingUsers] = useState(false)
  let users = ls('users') !== null ? ls('users') : []
  let {hasMounted} = useMounted();

  useEffect(() => {
    if(users < 1) {
      setLoadingUsers(false)
    }
    console.log('load effect', users)
  }, [])

  console.log('declared users', users)
  const selectUser = (id) => {
    console.log(id)
    ls('activeUID', id)
    router.push('/')
  }
  const showUsers = () => {
    if(!loadingUsers) {
      return users.map((user) => {
        return(
          <div onClick={() => selectUser(user.id)} className="login-user__user-box" key={user.id}>
            <img className="login-user__user-image" src="https://images.unsplash.com/photo-1511485977113-f
            <div className="login-user__user-name">{user.user}</div>
          </div>
        )
      })
    }
  }
  const createUser = () => {
    router.push('/')
  }

  return (
      <div className="login-user">
        <div className="login-user__top">
          <div className="login-user__logo"/>
          <span className="login-user__title">
            Who Is Watching?
          </span>
        </div>

        <div className="login-user__form">
          {hasMounted ? showUsers() : '' }
        </div>
        <div className="login-user__buttons">
          <button className="login-user__kid" onClick={createUser}>Create User</button>
        </div>
      </div>
  )
}
export default Login;
```

Додаток B. component Featured Media

```
import { useRouter } from 'next/router'
import { useStateContext } from '../../WOprovider'
const FeaturedMedia = (props) => {
  const globalState = useStateContext();

  const router = useRouter()
  console.log('props featured', props)

  const clickedPlay = () => {
    router.push(`${props.linkUrl}`)
    console.log('send user to media page ' + props.mediaUrl)
  }
  const clickedAdd = (props) => {
    globalState.addToList({mediaId: props.mediaId, mediaType: props.mediaType, mediaUrl: props.mediaUrl})
    console.log('Clicked TO ADD MOVIE')
  }
  const showMedia = () => {
    if(props.type === 'front') {
      return(
        <iframe
        className="f-media__video"
        width="100%"
        height="100%"
        src={props.mediaUrl}
        allow="accelerometer; autoplay; clipboard-write;encrypted-media; gyroscope; picture-in-picture"
        allowFullScreen />
      )
    } else {
      return(
        <img src={props.mediaUrl} className="f-media__img" />
      )
    }
  }
   .   ,
   return(
     <div className={`f-media ${props.type === 'single' ? 'f-media--single' : ''}`}>
       {showMedia()}

       <div className="f-media__background">
         <div className="f-media__container">
           <div className="f-media__title" onClick={clickedPlay}>{props.title}</div>
           <div className={`f-media__playing ${props.type === 'single' ? 'hide-comp': ''}`}>NOW PLAYING</div>
           <div className={`f-media__location ${props.type === 'single' ? 'hide-comp': ''}`}>{props.location}</div>
           <div className="f-media__buttons">
             <div className="f-media__play-button" onClick={clickedPlay}>
               <i className="fas fa-play"/>
             </div>
             <div className={`f-media__add-button ${props.type !== 'single' ? 'hide-comp': ''}`} onClick={() => clickedAdd
               <i className="fas fa-plus"/>
             </div>
             <div className={`f-media__info-button ${props.type === 'single' ? 'hide-comp': ''}`} onClick={clickedPlay}>MO
           </div>
         </div>
       </div>
     </div>
   )
}

export default FeaturedMedia;
```

Додаток Г. component Search

```
import { useStateContext } from "../../WOprovider";
import { useEffect, useState } from "react";
import axios from "axios";
import Link from "next/link";
import { useRouter } from "next/router";

const Search = (props) => {
    const globalState = useStateContext();
    const [popData, setPopData] = useState([]);
    const [searchData, setSearchData] = useState([]);
    const [showResults, setShowResults] = useState(false);
    const [text, setText] = useState("");
    const router = useRouter();

    useEffect( () => {
        const fetchData = async () => {
        try {
            let popData = await axios.get(
                `https://api.themoviedb.org/3/discover/movie?primary_release_year=2021&api_
            );
            setPopData(popData.data.results.filter((item, i) => i < 14));

            setShowResults(false);
            console.log("popdata", popData.data.results);
        } catch (error) {
            console.log(error);
        }
    }
    }, []);

    useEffect(() => {
        if (globalState.searchOpen) {
            document.body.style.overflowY = "hidden";
        } else {
            document.body.style.overflowY = "auto";
        }
    }, [globalState.searchOpen]);
```

```
const handleInput = async (e) => {
    try {
        setText(e.target.value);
        let searchData = await axios.get(
            `https://api.themoviedb.org/3/search/multi?query=${e.target.value}&api_key=
        );
        setSearchData(
            searchData.data.results.filter(
                (item, i) => item.media_type === "tv" || item.media_type === "movie",
            ),
        );
        setShowResults(true);
    } catch (error) {
        console.log(error);
    }
};

const clickedposter = (type, id, media_type) => {
    if (type === "popular") {
        router.push(`/movie/${id}`);
        globalState.setSearchOpenAction(!globalState.searchOpen);
    }
    if (type === "search") {
        router.push(`/${media_type}/${id}`);
        globalState.setSearchOpenAction(!globalState.searchOpen);
    }
};
```

```
return (
    <div
        className={`search-modal ${
            globalState.searchOpen ? "search-modal--active" : ""
        }`}>
        <div className="search-modal__input-group ">
            <input
                className="search-modal__input"
                type="text"
                placeholder="search for a title"
                onChange={handleInput}
                value={text}
            />
            <div
                className="search-modal__close-button"
                onClick={() =>
                    globalState.setSearchOpenAction(!globalState.searchOpen)
                }>
                <i className="fas fa-times" />
            </div>
        </div>

        <h3 className="search-modal__title">
            {showResults && searchData.length >= 1
                ? `Search Result for ${text}`
                : "Popular Searches"}
        </h3>

        <div className="search-modal__posters">
            {showResults && searchData.length >= 1 ? (
                <SearchResults
                    searchData={searchData}
                    clickedposter={clickedposter}
                />
            ) : (
                <PopularResults
                    popData={popData}
                    clickedposter={clickedposter}
                />
            )}
        </div>
    </div>
);
```

```
const PopularResults = (props) => {
    return props.popData.map((item, index) => {
        return (
            <div
                key={index}
                className="search-modal__poster"
                onClick={() => props.clickedposter("popular", item.id)}>
                <img src={`https://image.tmdb.org/t/p/w185${item.poster_path}`} />
                <div className="search-modal__top-layer">
                    <i className="fas fa-play" />
                </div>
            </div>
        );
    });
};

const SearchResults = (props) => {
    return props.searchData.map((item, index) => {
        return (
            <div
                key={index}
                className="search-modal__poster"
                onClick={() =>
                    props.clickedposter("popular", item.id, item.media_type)
                }>
                <img src={`https://image.tmdb.org/t/p/w185${item.poster_path}`} />
                <div className="search-modal__top-layer">
                    <i className="fas fa-play" />
                </div>
            </div>
        );
    });
};

export default Search;
```

Додаток Ґ. component Media Row

```
import { useState, useEffect } from "react";
import axios from "axios";
import { shuffleArray } from "../../utilities";
import Link from "next/link";

const MediaRow = (props) => {
    const [loadingData, setLoadingData] = useState(true);
    const [movies, setMoviesData] = useState([]);

    useEffect(() => {
        axios
            .get(
                `https://api.themoviedb.org/3/${props.endpoint}&api_key=64e0f967a0e15cb2abcefb00f295a079&language=
            )
            .then(function (response) {
                setMoviesData(shuffleArray(response.data.results));
                setLoadingData(false);

            })
            .catch(function (error) {

            });
    }, [props.updateData]);

    const loopComp = (comp, digit) => {
        let posters = [<Skeleton key={'a'}/>, <Skeleton key={'b'}/>, <Skeleton key={'c'}/>, <Skeleton key={'d'}/>,


        return posters;
    };
    const showPosters = (type) => {
        return loadingData
            ? loopComp(<Skeleton />, 10)
            : movies.map((movie) => {
                  return <Poster key={movie.id} movieData={movie} type={type} mediaType={props.mediaType} />;
              });
    };

    return (
        <div className={`media-row ${props.type}`}>
            <h3 className="media-row__title">{props.title}</h3>
            <div className="media-row__posters">
                {showPosters(props.type)}

            </div>
        </div>
```

```
const Poster = (props) => {
    const posterSize = (type) => {
        if (props.type === "large-vertical") {
            return "400";
        }
        if (type === "small-vertical") {
            return "185";
        }
        if (type === "large-horizontal") {
            return "500";
        }
        if (type === "small-horizontal") {
            return "342";
        }
    };
    return (
        <Link href={`/${props.mediaType === 'movie' ? 'movie' : 'tv'}/${props.movieData.id}`} legac
            <a>
                <div className="media-row__poster">
                    <img
                        src={`https://image.tmdb.org/t/p/w${posterSize(props.type)}${
                            props.movieData.poster_path
                        }`}
                    />
                    <div className="media-row__top-layer">
                        <i className="fas fa-play" />
                    </div>
                </div>
            </a>
        </Link>
    );
};

const Skeleton = () => {
    return (
        <div className="media-row__poster-skeleton">
            <div className="media-row__poster-skeleton-img"></div>
        </div>
    );
};

MediaRow.defaultProps = {
    mediaType: 'movie'
}
export default MediaRow;
```