МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ М.П. ДРАГОМАНОВА

Факультет інформатики

Кафедра програмної інженерії

**ТРЕМБІЦЬКИЙ Нікіта Володимирович**

**РОЗРОБКА САЙТУ ПОШУКУ КІНОФІЛЬМІВ**

Курсова робота

Спеціальність: 6.050103 – програмна інженерія, 0501 – інформатика та обчислювальна техніка

**Науковий керівник:**

ТЕРЗЯН Тигран Корюнович

кандидат технічних наук

**Допущено до захисту:**

завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

підпис

М.М. Мучник

Київ - 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ М.П. ДРАГОМАНОВА

Кафедра програмної інженерії

Дисципліна Об'єктно-орієнтоване програмування

Спеціальність 6.050103 – програмна інженерія,

Курс 3 Група 31ІПЗ Семестр 5

ЗАВДАННЯ

**на курсову роботу студента**

Трембіцький Нікіта Володимирович

1. Тема роботи " Розробка сайту пошуку кінофільмів "

2. Строк здачі студентом закінченого роботи 30.11.2018

3. Вихідні дані до роботи:

– матеріали що до предметної області “Сайт пошуку кінофільмів”;

– вимоги, сформульовані у Технічному завданні на розробку програми “Movie 43”

4. Зміст пояснювальної записки (перелік питань, які підлягають розробці):

– Об'єктний аналіз предметної області.

– Постановка задачі.

– Розробка проекту засобами UML.

– Реалізація проекту засобами ООП.

– Контрольний приклад виконання програми.

– Технічне завдання.

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

– UML-діаграми варіантів використання, класів, взаємодії.

6. Дата видачі завдання 27.09.18

# КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів розрахунково-графічної роботи | **Термін**  **виконання**  **етапів роботи** | **Підписи**  **керівника,**  **студента** |
|  | Отримання теми курсової роботи та узгодження постановки задачі з керівником. Складання плану роботи. | 27.09.17 |  |
|  | Пошук та вивчення літератури з питань курсової роботи. Виконання об'єктно орієнтованого аналізу предметної області задачі. Опис основних понять. | 09.10.17 |  |
|  | Складання технічного завдання на розробку програми та його узгодження з керівником. Розробка діаграм варіантів використання. | 11.10.17 |  |
|  | Проектування програми: розробка UML- діаграм класів, взаємодії, станів та діяльності. | 18.10.17 |  |
|  | Розробка та узгодження з керівником інтерфейсу користувача | 23.10.17 |  |
|  | Розробка інформаційного забезпечення: проектування структур файлів та схем бази даних. Створення та початкове наповнення інформаційного забезпечення. | 08.11.17 |  |
|  | Розробка програмного забезпечення: реалізація класів та автономна перевірка їх методів. Опис та обґрунтування обраних програмних рішень. | 15.11.17 |  |
|  | Налагодження та перевірка працездатності програми в цілому. Її демонстрація керівнику роботи. | 20.11.17 |  |
|  | Опис роботи програми. Підготовка пояснювальної записки в цілому. Її передача керівнику. | 27.11.17 |  |
|  | Оформлення додатків до пояснювальної записки: технічне завдання, інструкція оператора, текст програми, ілюстрації до її роботи тощо. | 29.11.17 |  |
|  | Передача роздрукованої курсової роботи на кафедру. | 30.11.17 |  |
|  | Захист курсової роботи. | За розкладом |  |

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Трембіцький Н.В.

(підпис)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Терзян Т.К.

(підпис) (прізвище, ім'я, по батькові)

" " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018 р.

ЗМІСТ

ВСТУП…………………………………………………………………………………5

ОБ’ЄКТНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ «САЙТ ПОШУКУ КІНОФІЛЬМІВ»………………………………………………..………………...........6 Аналіз предметної області ……………………………….................................6

Постановка задачі ……………………………………………………………...9

ПРОЕКТУВАННЯ САЙТУ ПОШУКУ КІНОФІЛЬМІВ……………………..........11

Структури даних та діаграми класів……………………………………….....11

Схеми взаємодії об’єктів та алгоритми виконання їх методів.......................13

Опис класів……………………………………………………………………..14

РЕАЛІЗАЦІЯ САЙТУ ПОШУКУ КІНОФІЛЬМІВ …………………………….…..20

Вибір середовища розробки …………………..………………………………20

Реалізація класів ……………………………...………………………………..21

Використання запозичених класів ……………………………………………23

Контрольний приклад …………………………………………………………26

ВИСНОВКИ………………………………………………………………..………….28

ЛІТЕРАТУРА………………………………………………………………………….29

ДОДАТКИ…………………………...………………………………………………...30

**ВСТУП**

Кіно стало невід’ємною частиною нашого життя. Перегляд художньго фільму – це один із найпоширеніших видів відпочинку. Одні фільми можуть розважити, інші – переносити в минуле чи відправляти у космос, збагачуючи глядача корисною інформацією. Хтось взагалі полюбляє фільми жахів. Коротко кажучи, сьогодні кінематограф пропонує фільми на будь-який смак і з будь-якою метою. Кіноіндустрія постійно розвивається і дивує чимось новим.

Українці люблять дивитися кіно і, завдяки високошвидкісному інтернету та "м’якому" законодавству, часто роблять це вдома. Наша країна входить у топ держав із найшвидшим – і найдешевшим у світі – доступом у всесвітню мережу. І навіть після закриття сайтів, які закордонні правовласники звинувачували в порушенні авторських прав – наприклад, Ex.ua – у нас усе ще є багато можливостей дивитися кіно в інтернеті, та ще й безкоштовно. [1]

На теренах всесвітньої мережі побутує жарт, мовляв, процес пошуку цікавого фільму тривав довше, ніж його перегляд. Причому не факт, що навіть за таких обставин і витрачених зусиль очікування будуть виправдані. Тому я вирішив створити веб-сайт, який має полегшити підбір кінострічки для перегляду. Щоправда, відразу ж варто зазначити, що мій сайт лише може "порадити", що дивитися, але не надасть доступу для перегляду фільмів. [2]

Курсова робота складається зі вступу, 3 розділів, висновків, списку

використаних джерел із 9 найменувань, 2 додатків, містить 7 ілюстрацій. Загальний обсяг 68 сторінок

**Розділ 1. Об’єктний аналіз предметної області «Сайт пошуку кінофільмів»**

**1.1. Аналіз предметної області**

Веб-сайт — це сукупність логічно зв'язаної гіпертекстової інформації, оформленої у вигляді окремих сторінок і доступної в мережі Інтернет.

Подібне визначення веб-сайту було правильним на початку існування Інтернету, коли мережа і веб-сайти використовувалися в основному як розважальна система. До кінця 90-х років веб-сайти дійсно були в основному статичними сторінками. Для створення веб-сайту було потрібне лише знання мови гіпертекстової розмітки — HTML. Якщо ж сторінка надавала якісь програмні засоби — це були виключно засоби, що міг надати сервер, на якому розташований веб-сайт. Про зручність і красу тогочасних веб-сайтів взагалі особливо не доводилося говорити. Час спливає, розвиваються мови програмування, розширюються канали передачі інформації... Зараз Інтернет вже є самодостатньою галуззю економіки, а веб-сайти стали повноправними представництвами фірм в Інтернеті.

Аналізуючи вітчизняне законодавство, можна дійти до висновку, що дефініція поняття «веб-сайт» міститься у Законі України “Про авторське право і суміжні права”[4], де визначено, що веб-сайт - сукупність даних, електронної (цифрової) інформації, інших об’єктів авторського права і (або) суміжних прав тощо, пов’язаних між собою і структурованих у межах адреси веб-сайту і (або) облікового запису власника цього веб-сайту, доступ до яких здійснюється через адресу мережі Інтернет, що може складатися з доменного імені, записів про каталоги або виклики і (або) числової адреси за Інтернет-протоколом;

*Типи веб-сайтів.*

*Рекламні веб-сайти*

Веб-сайти можуть створюватися виключно в рекламно-промоутерських цілях. Такі сайти безпосередньо не займаються продажем, їх завдання полягає в донесенні до цільової аудиторії рекламної інформації, і створюються вони з розрахунку на певне коло товарів або послуг. Зазвичай такі сайти виконуються з використанням великої кількості графіки, flash-анімації. Для залучення клієнтів на сайт використовують ігрові й розважальні методи.

*Веб-сайти-продавці*

Для таких сайтів характерна наявність описового рекламного матеріалу для товарів або послуг, каталог даних товарів або послуг, інформації про фірму-продавця, а також контактна інформація. Додаткові сервіси, такі, як корисна інформація, зручність замовлення через сайт у поєднанні із грамотною розкруткою, можуть зробити веб-сайт привабливим для сторонніх рекламодавців.

*Веб-сайти-"альтруїсти"*

Інформаційні веб-сайти, або сайти, які надають деякі безкоштовні сервіси, теж потрібно обслуговувати, розвивати, а отже, вкладати в них кошти. Але проекти, які не приносять прибуток, довго не живуть, тому для таких веб-сайтів характерне заробляння грошей або на рекламі, або на зборі статистичних даних. На таких сайтах дуже часто пропонують зареєструватися, аби отримати маленький додатковий сервіс.

*Веб-сайти для підтримки*

Останній тип веб-сайтів — це підтримка клієнтів. Зазвичай на таких сайтах розміщують оновлення для програмних продуктів, новини; якщо йдеться про сайт банку, це може бути система управління засобами клієнта. Ці інтернет-ресурси є рекламою фірми, товару та ін. [3]

Сайт, розроблювальний в курсовій роботі, належить до типу веб-сайти-"альтруїсти" з функціями пошукової системи. Тематика сайту – кінематограф. Виходячи з цього відомими україномовними аналогами є сайти: КіноБаза, kino.net.ua, Кіноріум, kino-teatr.ua, Moviestape, Джарвіс.

КіноБаза — єдиний сервіс українською з можливістю створювати довільні списки фільмів та серіалів, акторів з можливістю вибору приватності. Сервіс також має найгнучкіший фільтр фільмів, серіалів та акторів серед подібних сайтів українською. Крім цього на сайті існують такі можливості як:

* можливість додавати фільм у список улюблених та список перегляну пізніше;
* можливість додавати персону в список улюблених;
* досить непоганий фільтр фільмів;
* можливість імпортувати та експортувати оцінки;
* можливість прив'язувати та відв'язувати облікові записи соціальних мереж;
* налаштування приватності та інші деталі облікового запису;
* сніппет рейтингу кожного фільму, що є на сайті;
* касові збори в Україні.
* створення користувацьких списків акторів та фільмів, серіалів [5]

Але найбільш відомим кінопорталом є сайт Imdb, проте він англомовний.

База даних значною мірою заповнюється добровольцями, це чимось нагадує концепцію вікі. У базі зараз зібрана інформація про більш ніж 4,7 мільйони фільмів і телесеріалів, є інформація майже про 6,9 мільйонів акторів, режисерів та інших професіоналів кіно зі всього світу.

Досить популярним в Україні був сервіс Кінопошук, але з 16 травня 2017 року доступ до проекту з території України обмежено строком на три роки, за рішенням РНБО (як один з проектів Яндексу).

**1.2. Постановка задачі**

Згідно з проведеним аналізом предметної області потрібно розробити веб-сайт і спроектувати його взаємодію з відкритою інтернет базою даних фільмів TheMovieDB.org

Реалізувати на сайті функції:

* Пошук фільмів, серіалів, акторів і режисерів
* Можливість додавати фільм у список перегляну пізніше
* Фільтр фільмів
* Детальна інформація про фільми і серіали
* Біографія і фільмографія акторів і режисерів
* Рекомендація популярних і найкращих фільмів
* Підбір фільмів подібних обраному

Так як база даних TheMovieDB.org надає повнофункціональне API для роботи з базою, то потрібно реалізувати тільки клієнтську частину сайту. Нині найкраща мова програмування для написання додатків на клієнтській стороні – це JavaScript.

JavaScript (JS) — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. [6]

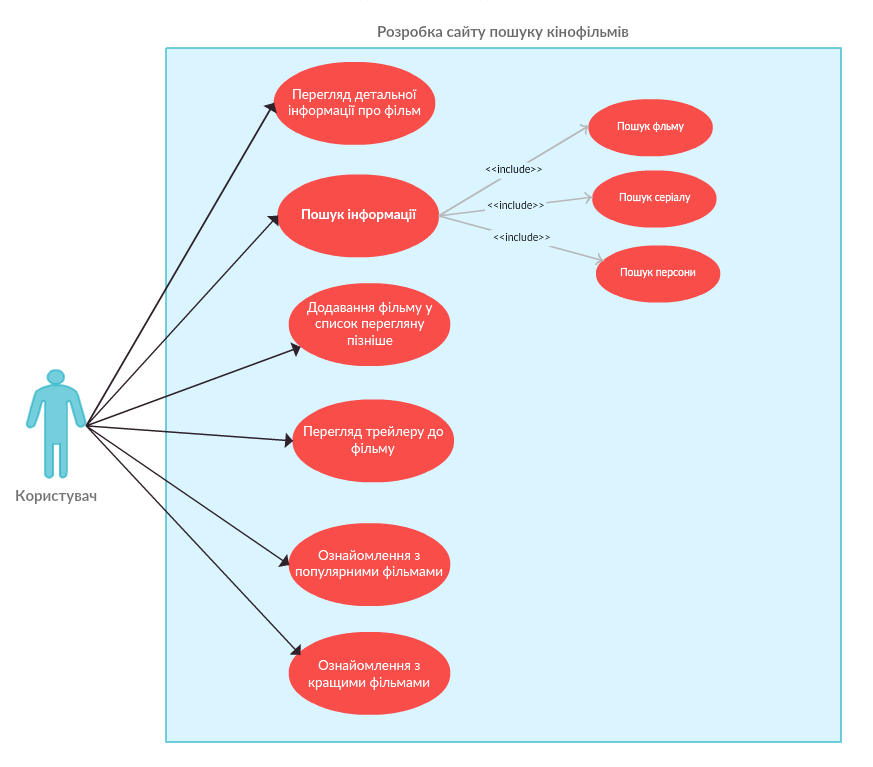


Рис 1.1 Діаграмма використання

**Розділ 2. Проектування сайту пошуку кінофільмів**

**2.1. Структури даних та діаграми класів**

Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об’єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об’єкти й кооперації, а також їхні відносини. [8]

На цій діаграмі показано класи, об’єкти й кооперації, а також їхні відносини:[9]

****

Рис. 2.1 Діаграма класів

**2.2 Схеми взаємодії об’єктів та алгоритми виконання їх методів**

Сайт був спроектований по архітектурному шаблону MVC (Моде́ль–вигляд–контро́лер). Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель. [7]

**2.3 Опис класів**

**Структура класу Item**

Базовий клас для опрацювання детальної інформації. Є основою для майбутніх класів.

Конструктор класу – присвоює значення полю id.

*Клас містить такі поля:*

Id – поле, яке містить унікальний ідентифікатор елементу.

Cast – поле, яке містить масив акторів кінофільму.

Crew – поле , яке містить масив знімальної групи.

Recommendations – поле, яке містить масив подібних до обраного фільмів.

*Клас містить такі методи:*

insertIntoDOM() – метод вставляє розмітку в DOM-дерево сторінки.

getPictureURL() – метод повертає відформатований шлях до зображення.

renderGenre() – метод обробляє і повертає жанр кінофільму.

renderRuntime() – метод обробляє і повертає тривалість кінофільму.

renderDate() – метод обробляє і повертає дату в залежності від локалізації.

renderRate() – метод обробляє і повертає оцінку кінофільму в зірках і балах.

renderButtons() – асинхронний метод, який викликає метод getVideo і в залежності від його результату відображає соціальні кнопки.

getVideo() – асинхронний метод, який надсилає запит на сервер, обробляє відповідь і повертає посилання на трейлер кінофільму.

getCredits() - асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю cast масив акторів кінофільму і полю crew масив учасників знімального процесу.

getRecommend() – асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю recommendations масив подібних до обраного фільмів.

renderDirector() – метод обробляє дані в полі crew і повертає ім’я режисера.

renderBudget() - метод повертає відформатований бюджет фільму.

renderCountries() – метод повертає країни-виробники кінофільму.

renderCompanies() – метод повертає компанії-виробники кінофільму. renderCasts() – метод викликає метод renderCast для кожного елементу масиву cast і повертає готову розмітку.

renderCast() – метод обробляє актора і повертає для нього розмітку.

renderRecommends() – метод викликає метод renderRecommend для кожного елементу масиву recommendations і повертає готову розмітку.

renderRecommend() – метод обробляє рекомендований фільм і повертає для нього розмітку.

**Структура класу Movie**

Клас є дочірнім від класу Item і уособлює в собі абстракцію об’єкта фільм.

Конструктор класу – викликає конструктор батьківського класу і присвоює полю id значення і полю type значення “movie/”.

*Клас містить такі поля:*

Type – поле, яке містить значення типу для створюваних об’єктів.

Res – поле, яке містить об’єкт з інформацією про фільм.

Title – поле, яке містить назву фільму.

Date – поле, яке містить дату першого показу фільму.

Img – поле, яке містить посилання на зображення постеру до фільму.

*Клас містить такі методи:*

getItem() – асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю res інформацію про фільм,title – назву, date – дату, img – посилання на зображення, а також викликає методи батьківського класу getCredits і getRecommend.

renderResults() - асинхронний метод, який обробляє дані про фільм, генерує розмітку і вставляє її в DOM-дерево проекту.

getGenres() – статичний метод, який надсилає запит на сервер і повертає масив жанрів фільмів.

**Структура класу TV**

Клас є дочірнім від класу Item і уособлює в собі абстракцію об’єкта серіал.

Конструктор класу – викликає конструктор батьківського класу і присвоює полю id значення і полю type значення “tv/”.

*Клас містить такі поля:*

Type – поле, яке містить значення типу для створюваних об’єктів.

Res – поле, яке містить об’єкт з інформацією про серіал.

Title – поле, яке містить назву серіалу.

Date – поле, яке містить дату першого показу серіалу.

Img – поле, яке містить посилання на зображення постеру до серіалу.

*Клас містить такі методи:*

getItem() – асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю res інформацію про фільм,title – назву, date – дату, img – посилання на зображення, а також викликає методи батьківського класу getCredits і getRecommend.

renderResults() - асинхронний метод, який обробляє дані про серіал, генерує розмітку і вставляє її в DOM-дерево проекту.

getGenres() – статичний метод, який надсилає запит на сервер і повертає масив жанрів серіалів.

renderDirector() - метод повертає відформатоване ім’я режисера.

renderSeasons() - асинхронний метод, який викликає метод renderSeason для кожного сезону серіалу і вставляє готову розмітку в документ.

renderSeason() - асинхронний метод, який генерує розмітку для сезону і викликає метод renderEpisodes.

renderEpisodes() - асинхронний метод, який надсилає запит на сервер, обробляє відповідь і для кожного епізоду викликає метод renderEpisode.

renderEpisode() – метод, який генерує розмітку для епізоду.

**Структура класу Person**

Клас є дочірнім від класу Item і уособлює в собі абстракцію об’єкта людина (актор або режисер).

Конструктор класу – викликає конструктор батьківського класу і присвоює полю id значення і полю type значення “p/”.

*Клас містить такі поля:*

Type – поле, яке містить значення типу для створюваних об’єктів.

Res – поле, яке містить об’єкт з інформацією про людину.

Credits – поле, яке містить масив фільмів, в якому знімалася чи знімала людина.

*Клас містить такі методи:*

getItem() – асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю res інформацію про людину, а також викликає метод getCredits.

renderResults() - асинхронний метод, який обробляє дані про людину, генерує розмітку і вставляє її в DOM-дерево проекту.

getCredits() – асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю credits масив фільмів, в якому знімалася чи знімала людина.

renderAge() - метод повертає відформатований вік людини.

renderDeath() - метод повертає відформатовану дату смерті людини, якщо така є.

renderWorks() – метод, який для кожного елементу масиву credits викликає метод renderWork і повертає готову розмітку.

renderWork() - метод, який генерує розмітку для елементу фільмографії людини.

renderCharacter() – метод генерує розмітку для ролі людини.

**Структура класу TopRated**

Клас є дочірнім від класу Movie і уособлює в собі абстракцію об’єкта найращих фільмів.

Конструктор класу – викликає конструктор батьківського класу.

*Клас містить такі поля:*

Items – поле, яке містить масив елементів списку.

*Клас містить такі методи:*

getTopRated() - асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю items масив найращих фільмів.

**Структура класу Popular**

Клас є дочірнім від класу Movie і уособлює в собі абстракцію об’єкта популярних фільмів.

Конструктор класу – викликає конструктор батьківського класу.

*Клас містить такі поля:*

Items – поле, яке містить масив елементів списку.

*Клас містить такі методи:*

getPopular () - асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю items масив популярних фільмів.

**Структура класу MovieList**

Клас уособлює в собі абстракцію об’єкта списку з фільмами.

Конструктор класу – присвоює полям data і page значення.

*Клас містить такі поля:*

Data –поле, яке містить масив фільмів.

Page – поле, яке містить номер поточної сторінки.

*Клас містить такі методи:*

Sort() – метод, який в залежності від значення, яке передається, викликає потрібний метод сортування.

sortByMaxRaiting() – сортування за найбільшим рейтингом.

sortByMinRaiting() – сортування за найменшим рейтингом.

sortByNewDate() – сортування по новизні випуску(новіші).

sortByOldDate() - сортування по новизні випуску(старіші).

**Структура класу Likes**

Клас, який уособлює в собі абстракцію об’єкта списку з лайками.

Конструктор класу – ініціалізує поле likes як пустий масив.

*Клас містить такі поля:*

Likes – поле, яке містить масив лайків.

*Клас містить такі методи:*

addLike() – метод добавляє лайк в список лайків і викликає метод persistData.

deleteLike() – метод видяляє лайк зі списку лайків і викликає метод persistData.

isLiked() – метод перевіряє чи обраний елемент знаходиться в списку лайків.

getNumLikes() – метод повертає кількість лайків в списку.

persistData() – метод записує значення поля likes в локальне сховище браузера.

readStorage() - метод зчитує значення масиву likes з локального сховища браузера в поле likes.

**Структура класу Search**

Клас усособлює в собі абстракцію об’єкта пошук.

Конструктор класу – присвоює полю query значення.

*Клас містить такі поля:*

Query – поле, яке містить текст пошукового запиту.

Result – поле, яке містить масив результатів пошукового запиту.

TotalResults() – поле, яке містить кількість знайдених результатів.

*Клас містить такі методи:*

getResults() - асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю result масив результатів пошукового запиту і полю totalResults - кількість знайдених результатів.

goToPage() - асинхронний метод, який надсилає запит на сервер, обробляє відповідь і присвоює полю result масив результатів пошукового запиту, змінюючи сторінку пошукових результатів.

**Розділ 3. Реалізація сайту пошуку кінофільмів**

**3.1. Вибір середовища розробки**

Сайт був розроблений на мові розмітки HTML5 і CSS3, з використанням front-end бібліотеки Bootstrap 4. Ця бібліотека була розроблена командою розробників з компанії Twitter для спрощення написання сайтів і зручнішого впровадження адаптивності. Вона містить в собі готові елементи розробки сайтів, такі як блоки, форми, кнопки і тд.

Вся логіка сайта написана на мові програмування JavaScript за стандартом ECMA Script 2016.

*JavaScript (JS)* — динамічна, об'єктно-орієнтована прототипна мова програмування. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки. [6]

Через те, що під час проектування я зрозумів, що проект буде досить об’ємним, було вирішено використовувати модульний підхід в розробці.Модулі дозволяють розбити проект на частини і таким чином краще структурувати код.

Для збірки модулів в один виконуваний файл потрібен бандлер модулей. Під час розробки сайту був використаний Webpack.

Webpack - це те, що відомо нам як "бандлер модулей". Він приймає модулі JavaScript, аналізує їх залежності один від одного, а потім з'єднує їх найефективнішим способом, випускаючи у кінці лише один JavaScript файл. З Webpack, модулі не обмежені тільки файлами JavaScript. Завдяки Лоадерам, Webpack розуміє, що модуль JavaScript може потребувати CSS файл, а цей CSS файл може потребувати зображення. Результат роботи Webpack буде містити тільки те, що потрібно.

**3.2 Реалізація класів**

Кожен клас реалізовано на мові програмування JavaScript.

**Реалізація класу Likes**

*addLike()* – метод добавляє лайк в список лайків і викликає метод persistData.

**addLike**(id, title, date, img) {

const like = {

id,

title,

date,

img

};

this.likes.**push**(like);

this.**persistData**();

return like;

}

*deleteLike()* – метод видяляє лайк зі списку лайків і викликає метод persistData.

**deleteLike**(id) {

const index = this.likes.**findIndex**(el => el.id === id);

this.likes.**splice**(index, 1);

this.**persistData**();

}

*isLiked()* – метод перевіряє чи обраний елемент знаходиться в списку лайків.

**isLiked**(id) {

return this.likes.**findIndex**(el => el.id === id) !== -1;

}

getNumLikes() – метод повертає кількість лайків в списку.

**getNumLikes**() {

return this.likes.length;

}

persistData() – метод записує значення поля likes в локальне сховище браузера.

**persistData**() {

localStorage.**setItem**('likes', JSON.**stringify**(this.likes));

}

readStorage() - метод зчитує значення масиву likes з локального сховища браузера в поле likes.

**readStorage**() {

const storage = JSON.**parse**(localStorage.**getItem**('likes'));

if (storage) this.likes = storage;

}

Реалізацію решти класів уже було описано у пункті 2.3, а код їхньої реалізації буде приведений у Додатку 2.

**3.3. Використання запозичених класів**

Була використана jQuery бібліотека OwlCarousel для створення адаптивного слайдера з підтримкою мобільних пристроїв.

*Реалізація класу*

const owl = **$**(".owl-carousel");

owl.**owlCarousel**({

loop: true,

margin: 20,

responsive: {

0: {

items: 1

},

768: {

items: 3

},

1000: {

items: 4

}

}

});

**$**(".owl-carousel").**on**("wheel", ".owl-stage", function(e) {

const delta =

e.originalEvent.deltaY ||

e.originalEvent.detail ||

e.originalEvent.wheelDelta;

if (delta > 0) {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("next.owl");

} else {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("prev.owl");

}

e.**preventDefault**();

});

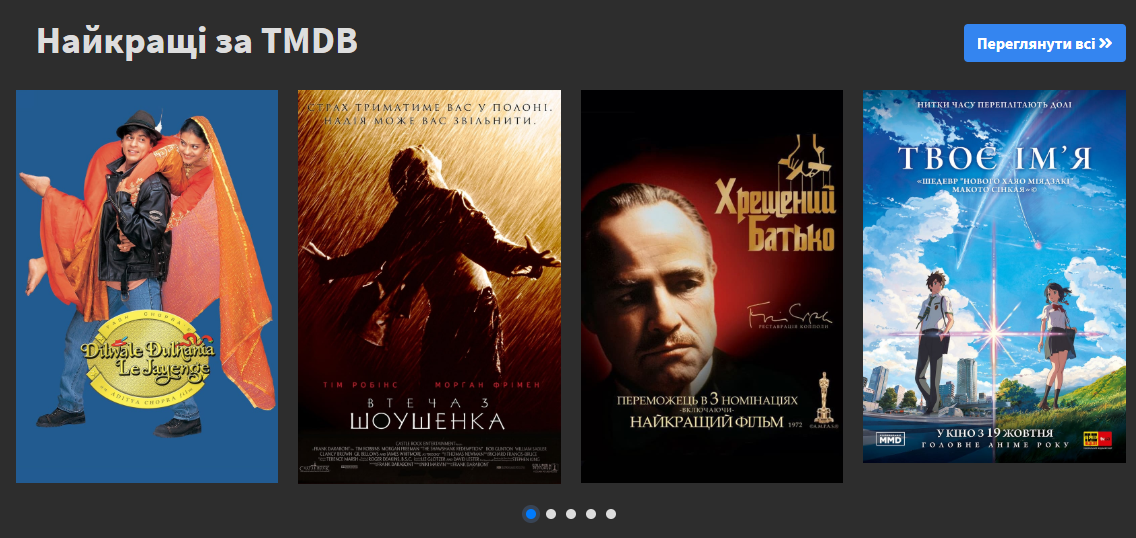
Також була застосована бібліотека MagnificPopup, за допомогою якої можна створювати модальні окна, галереї зображень і тд. Я використав для створення модального відеопрогравача з сайту youtube.com.

Рис. 3.1 Використання бібліотеки OwlCarousel

*Реалізація класу*

**$**(".content").**magnificPopup**({

delegate: ".btn-video",

type: "iframe"

});

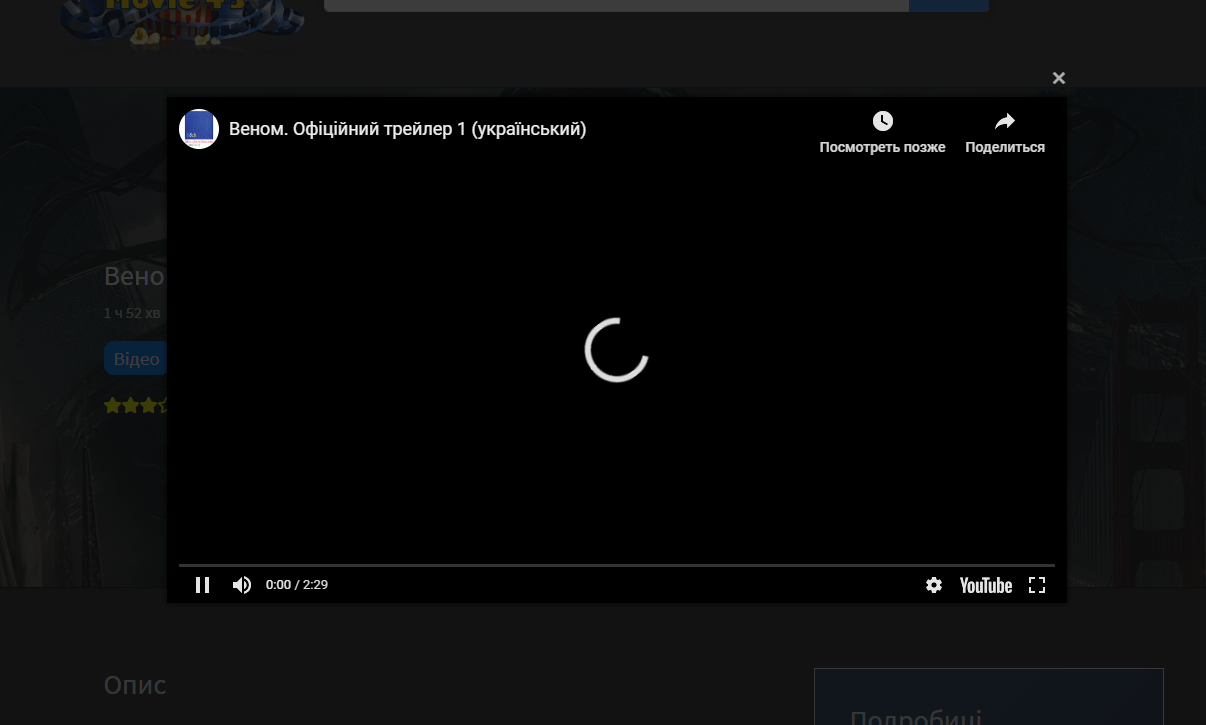


Рис. 3.2 Використання бібліотеки MagnificPopup

**3.4. Контрольний приклад**

Сайт був повністю завершений і доступний за посиланням <http://searchfilm.zzz.com.ua>.

На Рис 3.3 зображено головна сторінка сайту. На ній користувач може переглянути популярні на даний момент фільми і найкращі фільми за всі часи. Верхня частина сайту (хедер) незмінна і однакова на всіх сторінках сайту. Вона містить в собі пошукове вікно, логотип і вікно фільмів, які користувач хоче подивитись пізніше.

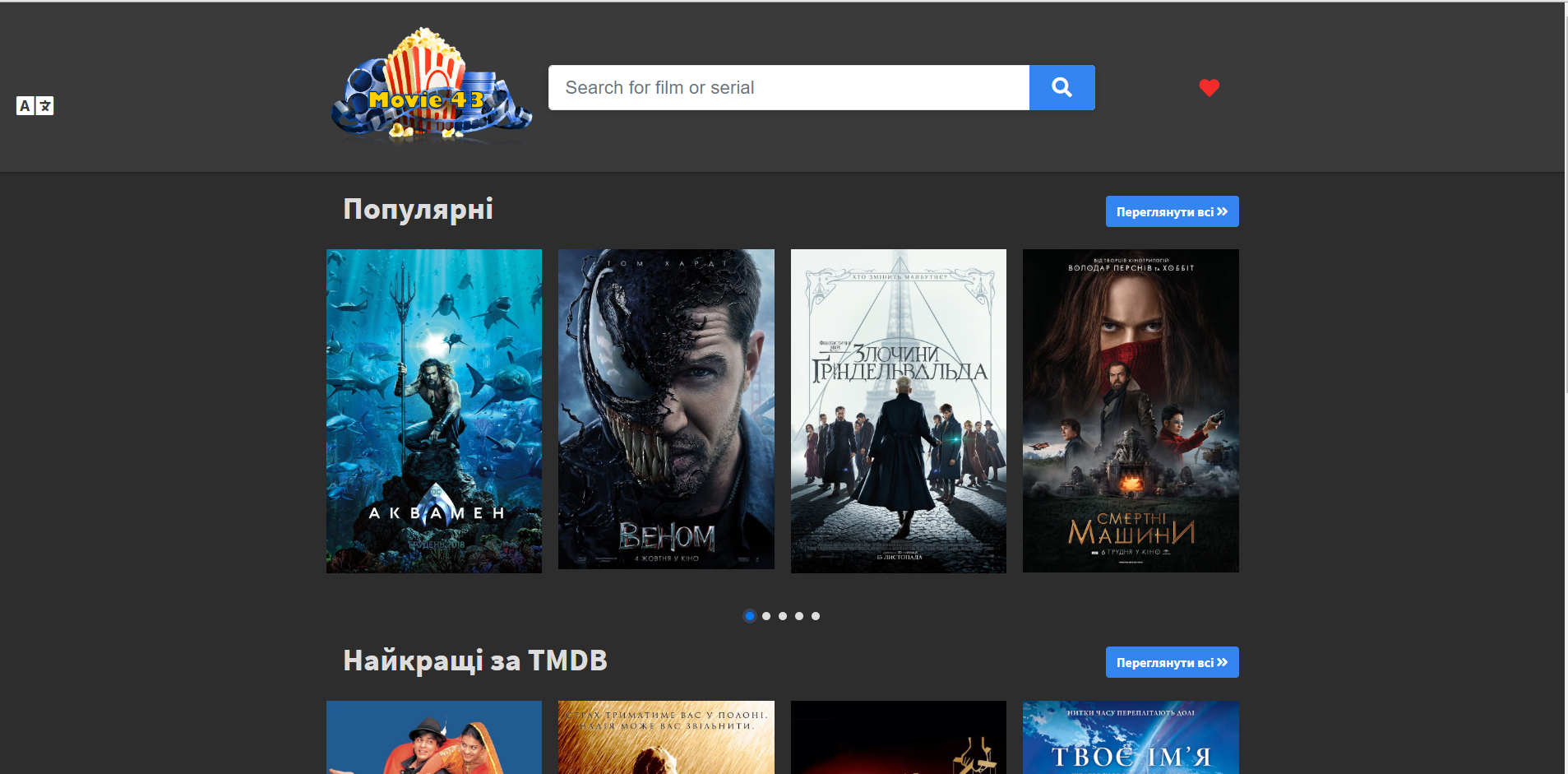


Рис 3.3 Головна сторінка

На Рис 3.4 зображено сторінка пошуку фільмів. Сторінка відображає коротку інформацію про фільми знайдені за запитом. Користувач може відсортувати результати запиту за критеріями і перейти на наступну або попередню сторінку результатів.

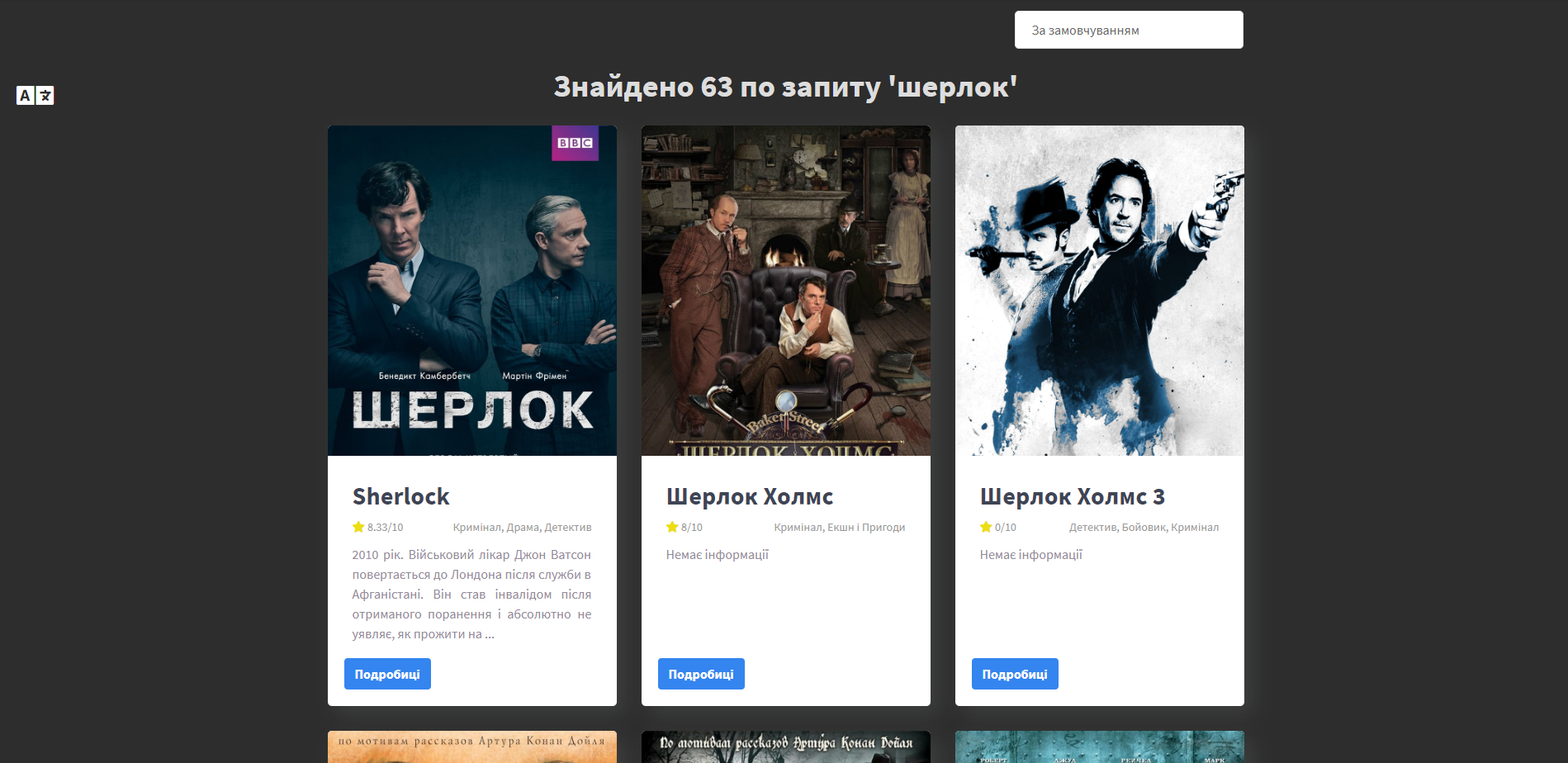


Рис 3.4 Сторінка пошуку

На Рис 3.5 зображена сторінка детальної інформації. На ній користувач може переглянути детальну інформацію про обраний фільм – почитати опис, подивитись трейлер, ознайомитись з акторським і режисерським складом, додати фільм в улюблені.

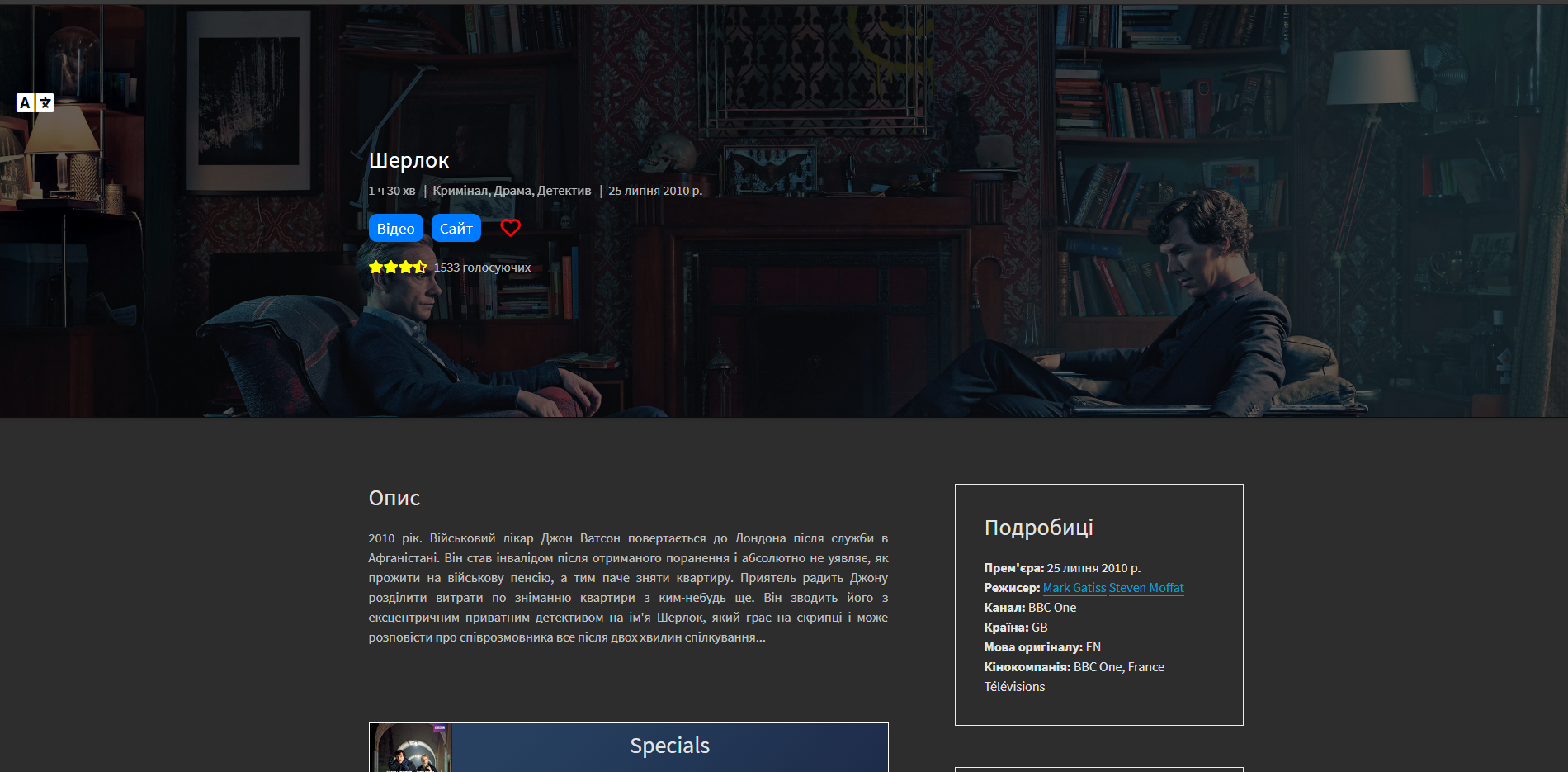


Рис 3.5 Сторінка детальної інформації

**Висновки**

В даній курсовій роботі вдалося розробити сайт на мові програмування JavaScript за стандартом ES6.

Виконання курсової роботи було розділено на два етапи:

* теоретичний аналіз поставленої задачі, що включає постановку задач, виділення основних класів, виділення основних дій та алгоритмів їх реалізації;
* програмна реалізація задач засобами JavaScript, що включає в себе реалізацію основних класів, реалізацію основних дій, проектування взаємодії з сервером.

Результатом курсової роботи можна вважати закріплення знань в області створення алгоритмів, навичок програмування в JavaScript та створення веб-додатків, отриманих під час курсу «об’єктно-орієнтоване програмування», та підготовка проекту для майбутньої дипломної роботи.

**Список використаних джерел**

1) Сила кінематографу, або чому соціальне кіно здатне змінити реальність? [Електроний ресурс]. Доступ за посиланням: <http://ildc.org.ua/statti/syla-kinematografu-abo-chomu-sotsialne-kino-zdatne-zminyty-realnist/>. [Дата звернення 14.12.2018 р.]

2) Як українці створили кінопошук "для себе", а потім вирішили на цьому заробити [Електроний ресурс]. Доступ за посиланням: <https://techno.nv.ua/ukr/made-in-ukraine/jak-ukrajintsi-stvorili-kinoposhuk-dlja-sebe-a-potim-virishili-na-tsomu-zarobiti-2445402.html>. [Дата звернення 14.12.2018 р.]

3) Веб-сайт: визначення й застосування [Електроний ресурс]. Доступ за посиланням: [http://www.webtec.com.ua/uk/articles/index/view/2011-05-05/web-site.](http://www.webtec.com.ua/uk/articles/index/view/2011-05-05/web-site.%20) [Дата звернення 14.12.2018 р.]

4) Про авторське право і суміжні права [Електроний ресурс]. Доступ за посиланням: <http://zakon.rada.gov.ua/laws/main/3792-12>. [Дата звернення 14.12.2018 р.]

5) КіноБаза Wikipedia [Електроний ресурс]. Доступ за посиланням: [https://uk.wikipedia.org/wiki/%D0%9A%D1%96%D0%BD%D0%BE%D0%91%D0%B0%D0%B7%D0%B0](https://uk.wikipedia.org/wiki/%D0%9A%D1%96%D0%BD%D0%BE%D0%91%D0%B0%D0%B7%D0%B0%20) [Дата звернення 14.12.2018 р.]

6) JavaScript Wikipedia [Електроний ресурс]. Доступ за посиланням: [https://uk.wikipedia.org/wiki/JavaScript](https://uk.wikipedia.org/wiki/JavaScript%20) [Дата звернення 14.12.2018 р.]

7) Модель-вид-контролер Wikipedia [Електроний ресурс]. Доступ за посиланням: <https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80> [Дата звернення 14.12.2018 р.]

8) Діаграма класів Wikipedia [Електроний ресурс]. Доступ за посиланням: [https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0\_%D0%BA%D0%BB%D0%B0%D1%81%D1%96%D0%B2](https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%96%D0%B2%20) [Дата звернення 14.12.2018 р.]

9) Creatly конструювання uml і інших діаграм [Електроний ресурс]. Доступ за посиланням [https://creately.com/.](https://creately.com/.%20) [Дата звернення 29.05.2018 р.]

**Додатки**

**Додаток 1**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ

ІМЕНІ М.П. ДРАГОМАНОВА

Факультет інформатики

Кафедра програмної інженерії

РОЗРОБКА САЙТУ ПОШУКУ КІНОФІЛЬМІВ

Технічне завдання на курсову роботу  
 з дисципліни "Об’єктно-орієнтоване програмування"

Розробник: Трембіцький Н.В.

Керівник: Терзян Т.К.

Київ 2018

ЗМІСТ

[РОЗДІЛ 1. НАЙМЕНУВАННЯ Й ОБЛАСТЬ ЗАСТОСУВАННЯ](#_Toc495610113) 32

[1.1. Найменування програмного засобу](#_Toc495610116) 32

[1.2. Призначення розробки та область застосування](#_Toc495610116) 32

[1.3. Мета](#_Toc495610116) 32

[1.4. Найменування розробника та замовника](#_Toc495610116) 32

[РОЗДІЛ 2. ПІДСТАВА ДЛЯ РОЗРОБКИ](#_Toc495610114) 33

[2.1. Документ на підставі якого ведеться розробка](#_Toc495610116) 33

[РОЗДІЛ 3. ВИМОГИ ДО ПРОГРАМИ](#_Toc495610115) 33

[3.1. Вимоги до функціональних характеристик](#_Toc495610116) 33

[3.1.1. Загальні вимоги](#_Toc495610116) 33

[3.1.2. Склад виконуваних функцій](#_Toc495610116) 33

[3.1.3. Організація вхідних і вихідних даних](#_Toc495610116) 33

[3.1.4. Часові характеристики і розмір пам'яті програми](#_Toc495610116) 33

[3.2. Вимоги до надійності](#_Toc495610117) 34

[3.2.1. Вимоги до надійного функціонування](#_Toc495610116) 34

[3.2.1. Контроль вхідної і вихідної даних](#_Toc495610116) 34

[3.2.1. Вимоги до надійного функціонування](#_Toc495610116) 34

[3.2.1. Час відновлення після відмови](#_Toc495610116) 34

[3.3. Умови експлуатації і збереження](#_Toc495610118) 35

[3.4. Вимоги до інформаційної і програмної сумісності](#_Toc495610119) 35

[3.4.1. Вимоги до інформаційних структур на вході і виході](#_Toc495610116) 35

[3.4.2. Вимоги до системи програмних засобів](#_Toc495610116) 35

[3.5. Вимоги до складу і параметрів технічних засобів](#_Toc495610120) 35

**1. Найменування й область застосування**

**1.1. Найменування програмного засобу**

Повне найменування програмної системи: "Сайт пошуку кінофільмів Movie43" (надалі "програма").

Коротка назва програмної системи - "Movie43"

**1.2. Призначення розробки та область застосування**

Сайт пошуку кінофільмів Movie43 повинен замінити користувачу Кінопошук.

Тобто бути довідковою системою (енциклопедією) у сфері кінематографа.

Система може бути використана як самостійний незалежний веб-додаток.

**1.3. Мета**

Програма має полегшити підбір кінострічки для перегляду і надати детальну інформацію про рекомендований фільм. Щоправда, відразу ж варто зазначити, що вона лише може "порадити", що дивитися, але не надасть доступу для перегляду фільмів.

**1.4. Найменування розробника та замовника.**

Розробник даного продукту - студент групи 31ІПЗ Трембіцький Нікіта.

Замовник програмного продукту - кафедра програмної інженерії Національного педагогічного університету імені М.П. Драгоманова в межах виконання курсової з дисципліни «Об’єктно-орієнтоване програмування» (надалі замовник).

**2. Підстава для розробки**

**2.1. Документ на підставі якого ведеться розробка**

Робота ведеться на підставі навчального плану за напрямом 121 «Інженерія програмного забезпечення» наказу про закріплення тем курсових робіт за студентами.

**3. Вимоги до програми**

**3.1. Вимоги до функціональних характеристик**

**3.1.1. Загальні вимоги**

Програмна система має забезпечувати:

* інтерфейс користувача, що не залежить від операційної системи;
* постійний доступ користувачів до БД;
* аутентифікацію користувачів та захист даних від несанкціонованого доступу;

**3.1.2. Склад виконуваних функцій**

Розробити веб-сайт і спроектувати його взаємодію з відкритою інтернет базою даних фільмів TheMovieDB.org

Реалізувати на сайті функції:

* Пошук фільмів, серіалів, акторів і режисерів
* Можливість додавати фільм у список перегляну пізніше
* Фільтр фільмів
* Детальна інформація про фільми і серіали
* Біографія і фільмографія акторів і режисерів
* Рекомендація популярних і найкращих фільмів
* Підбір фільмів подібних обраному

**3.1.3. Організація вхідних і вихідних даних**

Вхідними даними є пошукові запити користувача.

Організація вхідних і вихідних даних повинна відповідати інформаційній структурі виконуваних операцій, вхідним та вихідним паперовим документами.

**3.1.4. Часові характеристики і розмір пам’яті, необхідної для роботи програми.**

Час реакції програми на дії користувача (маніпуляції з пристроями введення даних) не повинен перевищувати 0,25 с.

Час виконання команд меню не більше 1 с.

Відображення масивів даних за запитами не більше 3 хвилин. Доступність БД – 90% цілодобово.

**3.2. Вимоги до надійності**

**3.2.1. Вимоги до надійного функціонування**

Програма повинна нормально функціонувати при безперебійній роботі ПК. Доступність БД залежить від сервісу, де ви плануєте орендувати сервер.

При апаратних збоях, відновлення нормальної роботи програми повинне виконуватися після:

а) апаратні збої сервера - перезавантаження ОС сервера, запуск сервера БД

б)апаратні збої робочої станції – перезавантаження ОС ПК, запуск виконуваного файлу програми.

в) БД повинна повертатись в найближчий несуперечний стан – передбачити точки відновлення.

При збоях програмного забезпечення:

а) система повинна забезпечувати можливість відновлення даних та фіксацію і «відкат» транзакцій.

б) в системі має бути реалізована коректна обробка виняткових ситуацій.

**3.2.2. Контроль вхідної і вихідної даних**

Для контролю коректності вхідних даних та захисту від помилок оператора потрібно:

* Використання механізму авто заповнення та вибору за переліком для зв’язаних даних;
* Захист від помилок оператора (залипань, випадкових символів тощо).
* Визначені некоректні дії повинні супроводжуватись повідомленнями про помилку і блокуванням операцій оновлення даних. В системі має бути передбачений захист від загального блокування.

Для вихідної даних передбачити:

* відсутність блокування даних через багато користувальницький доступ;
* постійне оновлення даних у відображених на екрані звітних формах.

**3.2.3. Час відновлення після відмови**

Час відновлення після відмови, не пов'язаною з роботою програми, повинен складатися із: часу перезапуску операційної системи; часу запуску сервера БД (підключення до сервера ), запуску виконуваного файлу, часу повторного введення або зчитування даних.

**3.3. Умови експлуатації і збереження**

Програма використовується у багато користувальницькому середовище. Регламенті операції проводити за графіком:

* створення резервних копій даних: 1 раз на тиждень;
* обслуговуючі функції (прибирання “сміття”, де фрагментація тощо) - щоденне;
* оновлення системи ідентифікації та аутентифікації користувачів – 1 раз на місяць та за потребою.

**3.4. Вимоги до інформаційної і програмної сумісності**

**3.4.1. Вимоги до інформаційних структур на вході і виході**

Формат відображення даних має дозволяти імпорт даних в додатки MS Office для редагування та перегляду.

**3.4.2. Вимоги до методів рішення і мов програмування**

Вибір методів рішення здійснюється розробникам без узгодження з замовником..

**3.4.3. Вимоги до системи програмних засобів**

Вимоги до програмного забезпечення сервера:

ОС – серверна версія ОС Windows не нижче Windows 2000 Server, IIS (або інший сервер), СУБД визначається встановлюється та налаштовується розробником (ліцензування СУБД виконується замовником).

Вимоги до програмного забезпечення робочої станції: браузер, який має підтримку HTML5 і CSS3 .

**3.5. Вимоги до складу і параметрів технічних засобів**

Вимоги до складу технічних засобів:

Сервер:

* сервер у базовій конфігурації із підтримкою RAID;
* з’ємний запам’ятовуючий пристрій для архівування даних;
* принтер для друку;
* засоби для під’єднання до локальної мережі.
* ПК на базі процесорів Intel, AMD у стандартній комплектації;
* периферійні пристрої друку;
* засоби для під’єднання до локальної мережі. засоби для під’єднання до локальної мережі.
* Pentium III з тактовою частотою 750 МГц;
* оперативна пам’ять - 128Мбайт;
* об’єм дискової пам’яті - 20Гбайт.

**Додаток 2. Код програми**

**Контролер**

*// Импорт стилей*

import "../css/style.scss";

*// Импорт слайдер библиотеки*

import "owl.carousel/dist/assets/owl.carousel.scss";

import "owl.carousel";

*//// Импорт компонентов*

*// Поиск*

import Search from "./models/Search";

import \* as searchView from "./views/searchView";

*// Популярные фильмы*

import Popular from "./models/Popular";

import \* as popularView from "./views/popularView";

*// Лучшие фильмы*

import TopRated from "./models/TopRated";

import \* as ratedView from "./views/topRatedView";

*// Импорт базовых элементов страницы*

import { elements, renderLoader, clearLoader } from "./views/base";

*// Импорт списка*

import MovieList from "./models/MovieList";

import \* as movieListView from "./views/movieListView";

import Likes from "./models/Likes";

import \* as likesView from "./views/likesView";

import Movie from "./models/Movie";

import TV from "./models/TV";

import Person from "./models/Person";

import config from "./config";

*// Обьект текущего состояния*

const state = {};

*/\*\**

*\* КОНТРОЛЛЕР ПОИСКА*

*\*/*

const **controlSearch** = async url => {

*// 1) Получаю запрос с представления*

const query = searchView.**getInput**() || url;

if (query && query != "") {

*// 2) Новый обьект поиска и добавляю в состоянии*

state.search = new **Search**(query);

*// 3) Очистка поискового поля*

searchView.**clearInput**();

*// 4) Очистка результата*

searchView.**clearResults**();

*// 5) Отрисовка лоадера*

**renderLoader**(elements.searchResList);

*// 6) Скрываю блок постеров и показываю результаты*

elements.sortBy.selectedIndex = 0;

searchView.**showResults**();

try {

*// 7) Поиск фильмов*

await state.search.**getResults**();

*// 8) Скрываю лоадер*

**clearLoader**();

elements.searchTitle.innerHTML = `Знайдено ${

state.search.totalResults

} по запиту '${**decodeURIComponent**(state.search.query)}'`;

state.mList = new **MovieList**(state.search.result);

movieListView.**renderResults**(

state.search.result,

state.search.totalResults

);

} catch (error) {

**clearLoader**();

**alert**("Something wrong with the search...");

**console**.**log**(error);

}

}

};

*/\*\**

*\* КОНТРОЛЕР ДЕТАЛЬНОЙ ИНФОРМАЦИИ*

*\*/*

const **controlDetail** = async (type, id) => {

switch (type) {

case "movie":

state.detail = new **Movie**(id);

break;

case "tv":

state.detail = new **TV**(id);

break;

case "person":

state.detail = new **Person**(id);

break;

default:

throw new **Error**("Page not found");

break;

}

searchView.**showDetail**();

**renderLoader**(elements.info);

await state.detail.**getItem**();

**clearLoader**();

state.detail.**renderResults**(state.likes.**isLiked**(state.detail.type + id));

};

*/\*\**

*\* КОНТРОЛЛЕР ЛАЙКОВ*

*\*/*

const **controlLikes** = () => {

if (!state.likes) state.likes = new **Likes**();

const currentID = state.detail.type + state.detail.id;

*// User has NOT yet liked current recipe*

if (!state.likes.**isLiked**(currentID)) {

*// Add like to the state*

const newLike = state.likes.**addLike**(

currentID,

state.detail.title,

state.detail.date,

state.detail.img

);

*// Toggle the like button*

likesView.**toggleLikeBtn**(true);

*// Add like to UI list*

likesView.**renderLike**(newLike);

*// User HAS liked current recipe*

} else {

*// Remove like from the state*

state.likes.**deleteLike**(currentID);

*// Toggle the like button*

likesView.**toggleLikeBtn**(false);

*// Remove like from UI list*

likesView.**deleteLike**(currentID);

}

likesView.**toggleLikeMenu**(state.likes.**getNumLikes**());

};

const **mainMenu** = async () => {

searchView.**showAdvertise**();

if (!state.topRated) {

*// 1) Создаю новый обьект лучших фильмов и добавляю в состоянии*

state.topRated = new **TopRated**();

*// 3) Получаю асинхронно лучшие и популярные фильмы*

await state.topRated.**getTopRated**();

*// 4) Отрисовываю постеры в дом-дерво*

ratedView.**renderResults**(state.topRated.items);

}

if (!state.popular) {

*// 2) Создаю новый обьект популярных фильмов и добавляю в состоянии*

state.popular = new **Popular**();

*// 3) Получаю асинхронно лучшие и популярные фильмы*

await state.popular.**getPopular**();

*// 4) Отрисовываю постеры в дом-дерво*

popularView.**renderResults**(state.popular.items);

}

*// 5) Инициализирую слайдер для постеров*

const owl = **$**(".owl-carousel");

owl.**owlCarousel**({

loop: true,

margin: 20,

responsive: {

0: {

items: 1

},

768: {

items: 3

},

1000: {

items: 4

}

}

});

};

const **popularMenu** = async () => {

searchView.**clearResults**();

searchView.**showResults**();

elements.searchTitle.innerHTML = "Популярні фільми";

if (!state.popular) {

state.popular = new **Popular**();

await state.popular.**getPopular**();

}

state.mList = new **MovieList**(state.popular.items);

movieListView.**renderResults**(state.popular.items);

};

const **top\_ratedMenu** = async () => {

searchView.**clearResults**();

searchView.**showResults**();

elements.searchTitle.innerHTML = "Найкращі фільми";

if (!state.topRated) {

state.topRated = new **TopRated**();

await state.topRated.**getTopRated**();

}

state.mList = new **MovieList**(state.topRated.items);

movieListView.**renderResults**(state.topRated.items);

};

const **controlURL** = () => {

let url = window.location.hash;

if (url) {

url = url.**replace**("#", "");

switch (url) {

case "popular":

**popularMenu**();

break;

case "top-rated":

**top\_ratedMenu**();

break;

default:

const urlInfo = url.**split**("/");

const [type, id] = urlInfo;

if (type === "search") **controlSearch**(id);

else **controlDetail**(type, id);

break;

}

} else {

**mainMenu**();

}

};

window.**addEventListener**("hashchange", controlURL);

*// Событие поиска в инпуте*

elements.searchForm.**addEventListener**("submit", e => {

e.**preventDefault**();

if (searchView.**getInput**() !== "")

window.location.hash = `search/${searchView.**getInput**()}`;

});

document.**addEventListener**("DOMContentLoaded", () => {

**$**(".content").**magnificPopup**({

delegate: ".btn-video",

type: "iframe"

});

**$**(".info").**on**("click", ".detail-main\_\_season-main", function(e) {

e.**preventDefault**();

var $this = **$**(this),

container = $this.**closest**(".detail-main\_\_season"),

currentContent = container.**find**(".detail-main\_\_episodes"),

duration = 200;

if (!container.**hasClass**("active")) {

container

.**addClass**("active")

.**siblings**()

.**removeClass**("active")

.**find**(".detail-main\_\_episodes")

.**stop**(true)

.**slideUp**(duration);

currentContent.**stop**(true).**slideDown**(duration);

} else {

container.**removeClass**("active");

currentContent.**stop**(true).**slideUp**(duration);

}

});

**$**(".owl-carousel").**on**("wheel", ".owl-stage", function(e) {

const delta =

e.originalEvent.deltaY ||

e.originalEvent.detail ||

e.originalEvent.wheelDelta;

if (delta > 0) {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("next.owl");

} else {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("prev.owl");

}

e.**preventDefault**();

});

**controlURL**();

state.likes = new **Likes**();

*// Restore likes*

state.likes.**readStorage**();

*// Toggle like menu button*

likesView.**toggleLikeMenu**(state.likes.**getNumLikes**());

*// Render the existing likes*

state.likes.likes.**forEach**(like => likesView.**renderLike**(like));

});

elements.searchResPages.**addEventListener**("click", e => {

const btn = e.target.**closest**(".btn-inline");

if (btn) {

const page = **parseInt**(btn.dataset.goto, 10);

**goToPage**(page);

}

});

const **goToPage** = async page => {

searchView.**clearResults**();

**renderLoader**(elements.searchResList);

try {

*// 7) Поиск фильмов*

await state.search.**goToPage**(page);

**clearLoader**();

state.mList = new **MovieList**(state.search.result, page);

movieListView.**renderResults**(

state.search.result,

state.search.totalResults,

page

);

} catch (error) {

**clearLoader**();

**alert**("Something wrong with the search...");

**console**.**log**(error);

}

};

elements.sortBy.**addEventListener**("change", e => {

searchView.**clearResults**();

const filter = e.target.options[e.target.selectedIndex].dataset.sort;

movieListView.**renderResults**(

state.mList.**sort**(filter),

state.search ? state.search.totalResults : undefined,

state.mList.page

);

});

elements.changeLanguage.**addEventListener**("click", () => {

config.**setLanguage**();

window.location.**reload**();

});

elements.info.**addEventListener**("click", e => {

if (e.target.**matches**(".detail-intro\_\_like, .detail-intro\_\_like \*")) {

e.**preventDefault**();

**controlLikes**();

}

});

**Конфиг**

const API\_KEY = "30c94ae83f75740e876f0072914f1491";

let language = localStorage.**getItem**('lang') || 'uk-UA';

const **changeLanguage** = () => {

if (language === "uk-UA") {

localStorage.**setItem**('lang', 'ru-RU');

} else {

localStorage.**setItem**('lang', 'uk-UA');

}

}

const **getLanguage** = () => {

return language;

}

export default {

searchUrl: "https://api.themoviedb.org/3/search/multi",

bgSrc: "https://image.tmdb.org/t/p/original",

imageSrc: "https://image.tmdb.org/t/p/w780",

noImageSrc: "https://via.placeholder.com/350x400/cccccc/000?text=NO+IMAGE+AVAILABLE",

baseMovieUrl: "https://api.themoviedb.org/3/",

queryTVById: "https://api.themoviedb.org/3/tv/",

queryMovieById: "https://api.themoviedb.org/3/movie/",

queryPersonById: "https://api.themoviedb.org/3/person/",

apiKey: API\_KEY,

language: **getLanguage**(),

**setLanguage**: () => {

**changeLanguage**();

},

popularMovie: "https://api.themoviedb.org/3/movie/popular",

topRatedMovie: "https://api.themoviedb.org/3/movie/top\_rated",

movieGenre: "https://api.themoviedb.org/3/genre/movie/list",

tvGenre: "https://api.themoviedb.org/3/genre/tv/list"

};

**Моделі**

**Item**

import {

elements

} from "../views/base";

import config from "../config";

export default class **Item** {

constructor(id) {

this.id = id;

}

**insertIntoDOM**(markup) {

elements.info.**insertAdjacentHTML**("afterbegin", markup);

}

**getPictureUrl**(url) {

if (url) {

return config.bgSrc + url;

} else {

return config.noImageSrc;

}

}

**renderGenre**(genres) {

if (!genres) return "Невідомо";

const resGenres = new **Set**();

genres.**forEach**(el => {

resGenres.**add**(el["name"]);

});

return `${**Array**.**from**(resGenres).**join**(", ")}`;

}

**renderRuntime**(time) {

if (!time) return `Невідомо`;

if (time / 60 > 1) {

return `${Math.**floor**(time / 60)} ч ${time % 60} хв`;

}

return `${time} хв`;

}

**renderDate**(date) {

if (date) {

const formatter = new **Intl**.**DateTimeFormat**("uk", {

year: "numeric",

month: "long",

day: "numeric"

});

return formatter.**format**(**Date**.**parse**(date));

} else {

return "-";

}

}

**renderRate**(rate) {

const countOfStars = 4;

let calcRate = (countOfStars \* rate) / 10;

let result = '';

for (let i = 0; i < countOfStars; i++) {

if (i < Math.**floor**(calcRate)) {

result += '<i class="fa fa-star"></i>';

} else if ((calcRate % 1) \* 10 > 5) {

calcRate = 0;

result += '<i class="fa fa-star"></i>';

} else if ((calcRate % 1) \* 10 < 5 && calcRate !== 0) {

calcRate = 0;

result += '<i class="fas fa-star-half-alt"></i>'

} else {

result += '<i class="far fa-star"></i>';

}

}

result += `<div class="detail-intro\_\_rating--rate">${rate}/10</div>`

return result;

}

async **renderButtons**(data, type) {

let result = "";

const urlVideo = await this.**getVideo**(type);

if (urlVideo) {

result += `<a href="http://www.youtube.com/watch?v=${urlVideo}" class="btn-inline btn-video" >Відео</a>`;

}

if (data.homepage) {

result += `<a href="${data.homepage}" target="\_blank" class="btn-inline">Сайт</a>`;

}

return result;

}

async **getVideo**(type) {

try {

let resJson = await **fetch**(`https://api.themoviedb.org/3/${type}/${this.id}/videos?api\_key=${config.apiKey}&language=${config.language}`);

let json = await resJson.**json**();

let res = json.results;

if (res.length == 0) {

resJson = await **fetch**(`https://api.themoviedb.org/3/${type}/${this.id}/videos?api\_key=${config.apiKey}&language=en-US`);

json = await resJson.**json**();

res = json.results;

}

let url = "";

for (let i = 0; i < res.length; i++) {

if (res[i].site == "YouTube" && res[i].type == "Trailer") {

url = res[i].key;

break;

}

}

return url;

} catch (error) {

**alert**(error);

}

}

async **getCredits**(type) {

try {

const resJson = await **fetch**(`https://api.themoviedb.org/3/${type}/${this.id}/credits?api\_key=${config.apiKey}&language=${config.language}`);

const json = await resJson.**json**();

this.cast = json.cast;

this.crew = json.crew;

} catch (error) {

**alert**(error);

}

}

async **getRecommend**(type) {

try {

const resJson = await **fetch**(`https://api.themoviedb.org/3/${type}/${this.id}/recommendations?api\_key=${config.apiKey}&language=${config.language}`);

const json = await resJson.**json**();

this.recommendations = json.results;

} catch (error) {

**alert**(error);

}

}

**renderDirector**() {

const crew = this.crew;

let result = '';

crew.**forEach**((el) => {

if (el.job == "Director") result += `<a href="#person/${el.id}">${el.name}</a> `;

});

return result;

}

**renderBudget**(budget) {

let result = '';

if (budget / (1000 \* 1000) > 1) {

if (budget / (1000 \* 1000 \* 1000) > 1) {

result += `${Math.**round**(budget/(1000 \* 1000 \* 1000))} billion`;

} else {

result += `${Math.**round**(budget/(1000 \* 1000))} million`;

}

} else {

result += budget;

}

return result;

}

**renderCountries**(countries) {

let result = [];

countries.**forEach**((el) => {

result.**push**(el.name);

})

return result.**join**(', ');

}

**renderCompanies**(company) {

let result = [];

company.**forEach**((el) => {

result.**push**(el.name);

})

return result.**join**(', ');

}

**renderCasts**() {

let result = "";

this.cast.**slice**(0, 5).**forEach**((el) => {

result += this.**renderCast**(el);

});

return result;

}

**renderCast**(cast) {

return `

<li class="detail-main\_\_cast-item">

<a href="#person/${cast.id}">

<div class="detail-main\_\_img-wrapper">

<img src="https://image.tmdb.org/t/p/w780/${cast.profile\_path}"

alt="${cast.name}" class="detail-main\_\_img">

</div>

<h6 class="detail-main\_\_cast-name">${cast.name}</h6>

</a>

</li>

`;

}

**renderRecommends**(type) {

let result = "";

this.recommendations.**forEach**((el) => {

result += this.**renderRecommend**(el, type);

});

return result;

}

**renderRecommend**(item, type) {

return `

<div class="movie-popular\_\_item">

<img src="${config.imageSrc}${item.poster\_path}" class="movie-popular\_\_poster"

alt="">

<div class="movie-popular\_\_buttons">

<a href="#${type}/${item.id}" class="btn btn-info">Детальніше</a>

</div>

<div class="movie-popular\_\_stars">

<div class="movie-popular\_\_rating">

${this.**renderRate**(item.vote\_average)}

</div>

</div>

<div class="movie-popular\_\_detail">

<h4 class="movie-popular\_\_title">${item.title || item.name}</h4>

<span class="movie-popular\_\_release">${this.**renderDate**(item.release\_date || item.first\_air\_date)}</span>

</div>

</div>

`;

}

}

**Likes**

export default class **Likes** {

constructor() {

this.likes = [];

}

**addLike**(id, title, date, img) {

const like = {

id,

title,

date,

img

};

this.likes.**push**(like);

*// Perist data in localStorage*

this.**persistData**();

return like;

}

**deleteLike**(id) {

const index = this.likes.**findIndex**(el => el.id === id);

this.likes.**splice**(index, 1);

*// Perist data in localStorage*

this.**persistData**();

}

**isLiked**(id) {

return this.likes.**findIndex**(el => el.id === id) !== -1;

}

**getNumLikes**() {

return this.likes.length;

}

**persistData**() {

localStorage.**setItem**('likes', JSON.**stringify**(this.likes));

}

**readStorage**() {

const storage = JSON.**parse**(localStorage.**getItem**('likes'));

*// Restoring likes from the localStorage*

if (storage) this.likes = storage;

}

}

**Movie**

import config from '../config';

import Item from './Item';

export default class **Movie** extends **Item** {

constructor(id) {

super(id);

this.type = 'movie/';

}

async **getItem**() {

try {

const resJson = await **fetch**(`${config.queryMovieById}${this.id}?api\_key=${config.apiKey}&language=${config.language}`);

this.res = await resJson.**json**();

this.title = this.res.title;

this.date = this.**renderDate**(this.res.release\_date);

this.img = this.**getPictureUrl**(this.res.poster\_path);

await this.**getCredits**('movie');

await this.**getRecommend**('movie');

} catch (error) {

**alert**(error);

}

}

async **renderResults**(isLiked) {

const data = this.res;

const markup = `

<article class="detail">

<section class="detail-intro" style="background: url(${this.**getPictureUrl**(data.backdrop\_path)}) no-repeat ;background-size:cover;">

<div class="container">

<div class="detail-intro\_\_info">

<h3 class="detail-intro\_\_title">${data.title}</h3>

<ul class="detail-intro\_\_subtext-list">

<li class="detail-intro\_\_subtext-item">${this.**renderRuntime**(data.runtime)}</li>

<li class="detail-intro\_\_subtext-item">${this.**renderGenre**(data.genres)}</li>

<li class="detail-intro\_\_subtext-item">${this.**renderDate**(data.release\_date)}</li>

</ul>

${await this.**renderButtons**(data,'movie')}

<a href="" class="detail-intro\_\_like"><i class="${isLiked?"fa fa-heart":"far fa-heart"}"></i></a>

<div class="detail-intro\_\_rating">

${this.**renderRate**(data.vote\_average)}

<span>${data.vote\_count} голосуючих</span>

</div>

</div>

</div>

</section>

<section class="detail-main">

<div class="container">

<div class="row">

<div class="col-lg-8 col-sm-12">

<div class="detail-main\_\_content">

<div class="detail-main\_\_storyline">

<h3 class="detail-main\_\_title">Опис</h3>

<p class="detail-main\_\_description">

${data.overview}

</p>

</div>

</div>

</div>

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

<aside>

<div class="detail-main\_\_widget widget">

<h3 class="detail-main\_\_title">Подробиці</h3>

<ul class="detail-main\_\_list list">

<li class="detail-main\_\_item"><strong>Прем'єра: </strong>${this.**renderDate**(data.release\_date)}</li>

<li class="detail-main\_\_item"><strong>Режисер: </strong>${this.**renderDirector**()}</li>

<li class="detail-main\_\_item"><strong>Бюджет: </strong>${this.**renderBudget**(data.budget)} USD</li>

<li class="detail-main\_\_item"><strong>Збори: </strong>${this.**renderBudget**(data.revenue)} USD</li>

<li class="detail-main\_\_item"><strong>Країна: </strong>${this.**renderCountries**(data.production\_countries)}</li>

<li class="detail-main\_\_item"><strong>Мова оригіналу: </strong>${data.original\_language.**toUpperCase**()}</li>

<li class="detail-main\_\_item"><strong>Кінокомпанія: </strong>${this.**renderCompanies**(data.production\_companies)}</li>

</ul>

</div>

<div class="detail-main\_\_widget widget">

<h3 class="detail-main\_\_title">Акторський склад</h3>

<ul class="detail-main\_\_list list">

${this.**renderCasts**()}

</ul>

</div>

</aside>

</div>

</div>

</div>

</section>

</article>

<article class="recommend">

<section class="movie-popular">

<div class="container">

<div class="row align-items-center">

<div class="col-md-8 text-center text-md-left">

<h2 class="movie-popular\_\_header">Вам також можуть сподобатись...</h2>

</div>

<div class="owl-carousel owl-theme">

${this.**renderRecommends**('movie')}

</div>

</div>

</div>

</section>

</article>

`;

this.**insertIntoDOM**(markup);

const owl = **$**(".info .owl-carousel");

owl.**owlCarousel**({

loop: true,

margin: 20,

responsive: {

0: {

items: 1

},

768: {

items: 3

},

1000: {

items: 4

}

}

});

**$**(".info .owl-carousel").**on**("mousewheel", ".owl-stage", function (e) {

if (e.originalEvent.wheelDelta > 0) {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("next.owl");

} else {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("prev.owl");

}

e.**preventDefault**();

});

}

static async **getGenres**() {

try {

const resJson = await **fetch**(`${config.movieGenre}?api\_key=${config.apiKey}&language=${config.language}`);

const res = await resJson.**json**();

return res.genres;

} catch (error) {

**alert**(error);

}

}

}

**MovieList**

export default class **MovieList** {

constructor(data, page) {

this.data = data;

this.page = page;

}

**sort**(filter) {

if (!this.data) {

return;

}

const data = [...this.data];

if (filter === "raiting-max") {

this.**sortByMaxRaiting**(data);

}

if (filter === "raiting-min") {

this.**sortByMinRaiting**(data);

}

if (filter === "date-new") {

this.**sortByNewDate**(data);

}

if (filter === "date-old") {

this.**sortByOldDate**(data);

}

return data;

}

**sortByMaxRaiting**(data) {

data.**sort**((a, b) => {

a.vote\_average = a.vote\_average || 0;

b.vote\_average = b.vote\_average || 0;

return a.vote\_average < b.vote\_average ? 1 : -1;

});

}

**sortByMinRaiting**(data) {

data.**sort**((a, b) => {

a.vote\_average = a.vote\_average || 0;

b.vote\_average = b.vote\_average || 0;

return a.vote\_average > b.vote\_average ? 1 : -1;

});

}

**sortByNewDate**(data) {

data.**sort**((a, b) => {

const a\_date = new **Date**(a.release\_date || a.first\_air\_date || "1200-01-01");

const b\_date = new **Date**(b.release\_date || b.first\_air\_date || "1200-01-01");

return a\_date < b\_date ? 1 : -1;

});

}

**sortByOldDate**(data) {

data.**sort**((a, b) => {

const a\_date = new **Date**(a.release\_date || a.first\_air\_date);

const b\_date = new **Date**(b.release\_date || b.first\_air\_date);

return a\_date > b\_date ? 1 : -1;

});

}

}

**Person**

import config from "../config";

import Item from "./Item";

const professions = {

Acting: "Актор",

Writing: "Сценарист",

Directing: "Режисер",

Art: "Візуальні ефекти"

};

export default class **Person** extends **Item** {

constructor(id) {

super(id);

this.type = "p";

}

async **getItem**() {

try {

const resJson = await **fetch**(

`${config.queryPersonById}${this.id}?api\_key=${

config.apiKey

}&language=${config.language}`

);

this.res = await resJson.**json**();

await this.**getCredits**();

} catch (error) {

**alert**(error);

}

}

**renderResults**() {

const data = this.res;

const markup = `

<article class="detail-person">

<div class="container">

<div class="row">

<div class="col-lg-7 col-sm-12">

<div class="detail-person\_\_info widget">

<h2 class="detail-person\_\_title">${

data.name

}</h2>

<p class="detail-person\_\_biography">${

data.biography

}</p>

</div>

<div class="detail-person\_\_films widget">

<h3 class="detail-person\_\_works-title detail-person\_\_title">Відомі роботи</h3>

<ul class="detail-person\_\_works">

${this.**renderWorks**()}

</ul>

</div>

</div>

<div class="col-lg-5 col-sm-12">

<div class="detail-person\_\_avatar">

<img class="detail-person\_\_img" src="${this.**getPictureUrl**(

data.profile\_path

)}">

</div>

<div class="detail-person\_\_facts widget">

<h3 class="detail-person\_\_title">Подробиці</h3>

<ul class="detail-person\_\_list list">

<li class="detail-person\_\_item"><strong>Дата народження:</strong> ${this.**renderDate**(

data.birthday

)} ${this.**renderAge**(

data.birthday,

data.deathday

)}</li>

${this.**renderDeath**(data.deathday)}

<li class="detail-person\_\_item"><strong>Місце народження: </strong>${data.place\_of\_birth ||

"-"}</li>

<li class="detail-person\_\_item"><strong>Професія: </strong>${

professions[data.known\_for\_department]

}</li>

</div>

</div>

</div>

</div>

</article>

`;

this.**insertIntoDOM**(markup);

}

async **getCredits**() {

try {

const resJson = await **fetch**(

`${config.queryPersonById}${this.id}/combined\_credits?api\_key=${

config.apiKey

}&language=${config.language}`

);

const res = await resJson.**json**();

if (this.res.known\_for\_department == "Acting") {

this.credits = res.cast;

} else {

this.credits = res.crew;

this.credits = this.credits.**filter**(el => {

return el.job == "Director";

});

}

this.credits.**sort**((a, b) => {

return a.popularity < b.popularity ? 1 : -1;

});

this.credits = this.credits.**slice**(0, 10);

} catch (error) {

**alert**(error);

}

}

**renderAge**(data, death) {

if (!data) return "";

if (death) return "";

return `(${new **Date**().**getFullYear**() -

new **Date**(**Date**.**parse**(data)).**getFullYear**()} років)`;

}

**renderDeath**(death) {

if (!death) return "";

return `

<li class="detail-person\_\_item"><strong>Дата смерті:</strong> ${this.**renderDate**(

death

)}</li>

`;

}

**renderWorks**() {

let result = "";

this.credits.**forEach**(el => {

result += this.**renderWork**(el);

});

return result;

}

**renderWork**(data) {

const markup = `

<li class="detail-person\_\_work">

<a class="detail-person\_\_work-link" href="#${data.media\_type}/${

data.id

}">

<div class="detail-person\_\_work-poster">

<img class="detail-person\_\_work-img" src="${this.**getPictureUrl**(

data.poster\_path

)}">

</div>

<div class="detail-person\_\_work-info">

<h3 class="detail-person\_\_work-title">${data.title ||

data.name}</h3>

<p class="detail-person\_\_work-character">${this.**renderCharacter**(

data.character

)}</p>

<p class="detail-person\_\_work-date">${this.**renderDate**(

data.first\_air\_date || data.release\_date

)}</p>

</div>

</a>

</li>

`;

return markup;

}

**renderCharacter**(data) {

if (!data) {

return "";

}

return `<strong>В ролі:</strong> ${data}`;

}

}

**Popular**

import config from '../config';

import Movie from "./Movie";

export default class **Popular** extends **Movie** {

constructor() {

super();

}

async **getPopular**() {

try {

const resJson = await **fetch**(`${config.popularMovie}?api\_key=${config.apiKey}&language=${config.language}`);

const res = await resJson.**json**();

this.items = res.results;

} catch (error) {

**alert**(error);

}

}

}

**Top-Rated**

import config from '../config';

import Movie from "./Movie";

export default class **TopRated** extends **Movie** {

constructor() {

super();

}

async **getTopRated**() {

try {

const resJson = await **fetch**(`${config.topRatedMovie}?api\_key=${config.apiKey}&language=${config.language}`);

const res = await resJson.**json**();

this.items = res.results;

} catch (error) {

**alert**(error);

}

}

}

**Search**

import config from '../config';

export default class **Search** {

constructor(query) {

this.query = query;

}

async **getResults**() {

try {

const resJson = await **fetch**(`${config.searchUrl}?api\_key=${config.apiKey}&language=${config.language}&query=${this.query}`);

const res = await resJson.**json**();

this.result = res.results;

this.totalResults = res.total\_results;

} catch (error) {

**alert**(error);

}

}

async **goToPage**(page) {

try {

const resJson = await **fetch**(`${config.searchUrl}?api\_key=${config.apiKey}&language=${config.language}&query=${this.query}&page=${page}`);

const res = await resJson.**json**();

this.result = res.results;

} catch (error) {

**alert**(error);

}

}

}

**Tv**

import config from "../config";

import Item from "./Item";

export default class **TV** extends **Item** {

constructor(id) {

super(id);

this.type = 'tv/';

}

async **getItem**() {

try {

const resJson = await **fetch**(

`${config.queryTVById}${this.id}?api\_key=${config.apiKey}&language=${config.language}`

);

this.res = await resJson.**json**();

this.title = this.res.name;

this.date = this.**renderDate**(this.res.first\_air\_date);

this.img = this.**getPictureUrl**(this.res.poster\_path);

await this.**getCredits**("tv");

await this.**getRecommend**("tv");

} catch (error) {

**alert**(error);

}

}

async **renderResults**(isLiked) {

const data = this.res;

const markup = `

<article class="detail">

<section class="detail-intro" style="background: url(${this.**getPictureUrl**(

data.backdrop\_path

)}) no-repeat ;background-size:cover;">

<div class="container">

<div class="detail-intro\_\_info">

<h3 class="detail-intro\_\_title">${data.name}</h3>

<ul class="detail-intro\_\_subtext-list">

<li class="detail-intro\_\_subtext-item">${this.**renderRuntime**(

data.episode\_run\_time[0]

)}</li>

<li class="detail-intro\_\_subtext-item">${this.**renderGenre**(

data.genres

)}</li>

<li class="detail-intro\_\_subtext-item">${this.**renderDate**(

data.first\_air\_date

)}</li>

</ul>

${await this.**renderButtons**(data, "tv")}

<a href="" class="detail-intro\_\_like"><i class="${isLiked?"fa fa-heart":"far fa-heart"}"></i></a>

<div class="detail-intro\_\_rating">

${this.**renderRate**(data.vote\_average)}

<span>${data.vote\_count} голосуючих</span>

</div>

</div>

</div>

</section>

<section class="detail-main">

<div class="container">

<div class="row">

<div class="col-lg-8 col-sm-12">

<div class="detail-main\_\_content">

<div class="detail-main\_\_storyline">

<h3 class="detail-main\_\_title">Опис</h3>

<p class="detail-main\_\_description">

${data.overview}

</p>

</div>

</div>

<div class="detail-main\_\_seasons">

</div>

</div>

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

<aside>

<div class="detail-main\_\_widget">

<h3 class="detail-main\_\_title">Подробиці</h3>

<ul class="detail-main\_\_list list">

<li class="detail-main\_\_item"><strong>Прем'єра: </strong>${this.**renderDate**(

data.first\_air\_date

)}</li>

<li class="detail-main\_\_item"><strong>Режисер: </strong>${this.**renderDirector**(

data.created\_by

)}</li>

<li class="detail-main\_\_item"><strong>Канал: </strong>${

data.networks[0].name

}</li>

<li class="detail-main\_\_item"><strong>Країна: </strong>${

data.origin\_country[0]

}</li>

<li class="detail-main\_\_item"><strong>Мова оригіналу: </strong>${data.original\_language.**toUpperCase**()}</li>

<li class="detail-main\_\_item"><strong>Кінокомпанія: </strong>${this.**renderCompanies**(

data.production\_companies

)}</li>

</ul>

</div>

<div class="detail-main\_\_widget">

<h3 class="detail-main\_\_title">Акторський склад</h3>

<ul class="detail-main\_\_list list">

${this.**renderCasts**()}

</ul>

</div>

</aside>

</div>

</div>

</div>

</section>

</article>

<article class="recommend">

<section class="movie-popular">

<div class="container">

<div class="row align-items-center">

<div class="col-md-8 text-center text-md-left">

<h2 class="movie-popular\_\_header">Вам також можуть сподобатися...</h2>

</div>

<div class="owl-carousel owl-theme">

${this.**renderRecommends**("tv")}

</div>

</div>

</div>

</section>

</article>

`;

this.**insertIntoDOM**(markup);

this.**renderSeasons**(data.seasons);

const owl = **$**(".owl-carousel");

owl.**owlCarousel**({

loop: true,

margin: 20,

responsive: {

0: {

items: 1

},

768: {

items: 3

},

1000: {

items: 4

}

}

});

**$**(".owl-carousel").**on**("mousewheel", ".owl-stage", function (e) {

if (e.originalEvent.wheelDelta > 0) {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("next.owl");

} else {

**$**(e.target)

.**closest**(".owl-carousel")

.**trigger**("prev.owl");

}

e.**preventDefault**();

});

}

static async **getGenres**() {

try {

const resJson = await **fetch**(

`${config.tvGenre}?api\_key=${config.apiKey}&language=${config.language}`

);

const res = await resJson.**json**();

return res.genres;

} catch (error) {

**alert**(error);

}

}

**renderDirector**(data) {

let result = "";

data.**forEach**(el => {

result += `<a href="#person/${el.id}">${el.name}</a> `;

});

return result;

}

async **renderSeasons**(data) {

let result = "";

for (let i = 0; i < data.length; i++) {

result += await this.**renderSeason**(data[i]);

}

document.**querySelector**('.detail-main\_\_seasons').**insertAdjacentHTML**("afterbegin", result);

}

async **renderSeason**(el) {

return `

<div class="detail-main\_\_season">

<div class="detail-main\_\_season-main">

<div class="detail-main\_\_season-poster">

<img class="detail-main\_\_season-img" src="${this.**getPictureUrl**(el.poster\_path)}">

</div>

<div class="detail-main\_\_season-info">

<h3 class="detail-main\_\_season-title">${el.name}</h3>

<p class="detail-main\_\_season-date"><i class="far fa-calendar-alt"></i> ${this.**renderDate**(el.air\_date)}</p>

<p class="detail-main\_\_season-ep\_count"><i class="far fa-bookmark"></i> Кількість серій: ${el.episode\_count}</p>

</div>

</div>

<ul class="detail-main\_\_episodes">

${await this.**renderEpisodes**(el.season\_number)}

</ul>

</div>

`;

}

async **renderEpisodes**(num) {

try {

const resJson = await **fetch**(

`${config.queryTVById}${this.id}/season/${num}?api\_key=${config.apiKey}&language=${config.language}`

);

const res = await resJson.**json**();

let markup = '';

res.episodes.**forEach**((el) => {

markup += this.**renderEpisode**(el);

})

return markup;

} catch (error) {

**alert**(error);

}

}

**renderEpisode**(el) {

return `

<li class="detail-main\_\_episode">

<div class="detail-main\_\_episode-poster">

<img class="detail-main\_\_episode-img" src="${this.**getPictureUrl**(el.still\_path)}">

</div>

<div class="detail-main\_\_episode-info">

<h3 class="detail-main\_\_episode-title">Серія ${el.episode\_number}</h3>

<h3 class="detail-main\_\_episode-title">${el.name}</h3>

<p class="detail-main\_\_episode-overview"> ${el.overview}</p>

</div>

</li>

`

}

}