ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка десктопного застосунку для керування каталогом фільмів на основі мови C++ і Qt»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо–професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Михайло ЗИКОВ
(підпис)

Виконав: здобувач вищої освіти групи ТЦР–44
Михайло ЗИКОВ

Керівник: Артур ГАВОР
ст. викладач

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО–КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально–науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо–професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ

Завідувач кафедри
Технологій цифрового розвитку

_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Зикову Михайлу Віталійовичу

1. Тема кваліфікаційної роботи: «Розробка десктопного застосунку для керування каталогом фільмів на основі мови C++ і Qt»
керівник кваліфікаційної роботи ст. викладач. Артур ГАВОР
затверджені наказом Державного університету інформаційно–комунікаційних технологій від «20» лютого 2026 року №51.
2. Строк подання кваліфікаційної роботи «___» травня 2026 року.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку десктопних застосунків, принципи об’єктно–орієнтованого програмування, документація до фреймворку Qt, засоби створення графічного інтерфейсу користувача, технічна документація мови програмування C++.
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих програмних рішень для керування каталогом фільмів.
2. Проектування структури десктопного застосунку для керування каталогом фільмів.
3. Програмна реалізація та опис функціонування застосунку на основі C++ і Qt.
4. Тестування розробленого застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.

3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Алгоритм роботи застосунку.
6. Діаграма класів.
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання «20» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково–технічної літератури	20.02.2026 – 27.02.2026	
2	Аналіз та дослідження існуючих аналогів	28.02.2026 – 05.03.2026	
3	Вибір та обґрунтування технологічного стеку	06.03.2026 – 10.03.2026	
4	Проєктування архітектури застосунку	11.03.2026 – 25.03.2026	
5	Програмна реалізація застосунку	26.03.2026 – 15.04.2026	
6	Тестування застосунку	16.04.2026 – 20.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	21.03.2026 – 29.04.2026	
8	Розробка демонстраційних матеріалів	30.04.2026 – 14.05.2026	
9	Підготовка презентації та попередній захист	15.05.2026 – 25.05.2026	

Здобувач вищої освіти _____ Михайло ЗИКОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Артур ГАВОР
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 6 табл., 36 рис., 22 джерел.

Мета роботи – розробка десктопного застосунку CinemaHub для ведення особистої кінотеки, що забезпечує зручніший і швидший доступ до власної фільмотеки порівняно з наявними онлайн–сервісами – завдяки повній офлайн–роботі без залежності від мережі, миттєвому пошуку з автодоповненням та сучасному інтерфейсу з перемиканням теми в реальному часі, реалізованому засобами мови C++ і фреймворку Qt 6.

Об'єкт дослідження – процес проєктування та розробки десктопного програмного забезпечення для керування мультимедійним каталогом.

Предмет дослідження – методи та засоби проєктування і реалізації десктопного застосунку для керування каталогом фільмів на основі мови C++ і фреймворку Qt 6.

Короткий зміст роботи: Проаналізовано наявні програмні аналоги та обґрунтовано вибір технологій. Спроєктовано архітектуру з тринадцяти класів, підключено SQLite та розроблено UML–діаграми. Реалізовано: анімована заставка, авторизація, каталог із пошуком, відкриття трейлерів, управління обраним, новини, тренди, профіль та перемикання теми. Застосунок працює повністю офлайн без залежності від мережі. Проведено тестування – усі тести пройдено успішно.

Сферою використання є організація та ведення особистої кінотеки в офлайн–режимі без залежності від зовнішніх сервісів.

КЛЮЧОВІ СЛОВА: ДЕСКТОПНИЙ ЗАСТОСУНОК, C++, QT, ГРАФІЧНИЙ ІНТЕРФЕЙС, КАТАЛОГ ФІЛЬМІВ, АВТОРИЗАЦІЯ, ПОШУК, МУЛЬТИМЕДІА

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна кіноіндустрія щороку поповнюється тисячами нових стрічок, що суттєво ускладнює процес пошуку, систематизації та відстеження переглянутого контенту. Більшість популярних інструментів для роботи з каталогами фільмів – IMDb, Letterboxd, Кінопошук – є онлайн–сервісами, що означає повну залежність від стабільного інтернет–з'єднання, обмежені можливості персоналізації та відсутність контролю над власними даними. Натомість попит на автономне програмне забезпечення, яке не вимагає підключення до мережі, зберігає дані локально та надає зручний і сучасний інтерфейс, залишається незадоволеним.

Розробка якісного десктопного застосунку для керування каталогом фільмів є актуальним інженерним завданням, яке потребує поєднання ефективних технологій і продуманого проєктування. Мова програмування C++ забезпечує високу швидкодію та ефективне управління ресурсами системи, а міжплатформний фреймворк Qt надає потужні засоби для створення сучасного графічного інтерфейсу з підтримкою мультимедіа. Поєднання цих інструментів дозволяє реалізувати застосунок, що за рівнем зручності та візуальною якістю не поступається провідним комерційним стрімінговим сервісам.

Для досягнення поставленої мети в бакалаврській роботі вирішено такі завдання.

Проведено аналіз існуючих програмних рішень для керування каталогом фільмів та визначено їх переваги і недоліки.

Досліджено можливості мови C++ та фреймворку Qt 6 для побудови крос–платформних десктопних застосунків із мультимедійними функціями. Спроєктовано архітектуру застосунку з використанням об'єктно–орієнтованого підходу та UML–нотації.

Реалізовано анімовану заставку при запуску, систему авторизації та реєстрації користувача.

Розроблено основний функціонал застосунку: перегляд каталогу у вигляді карток з постерами, горизонтальне прокручування по рядках добірок, пошук з

автодоповненням, фільтрацію за жанром, відображення детальної інформації про фільм, відтворення відеоматеріалів, управління списком обраного, перегляд новин, трендів і сповіщень.

Реалізовано перемикання між темною та світлою темами без перезапуску застосунку та персоналізацію профілю користувача.

Проведено тестування розробленого застосунку та виконано аналіз результатів.

Об'єкт дослідження – процес проєктування та розробки десктопного програмного забезпечення для керування мультимедійним каталогом.

Предмет дослідження – методи та засоби проєктування і реалізації десктопного застосунку на основі мови C++ і фреймворку Qt для управління каталогом фільмів.

Мета роботи – розробка десктопного застосунку CinemaHub для керування каталогом фільмів із сучасним графічним інтерфейсом, функціями пошуку, фільтрації, перегляду деталей, відтворення відео та керування профілем користувача на основі мови програмування C++ і фреймворку Qt.

Методи дослідження. У процесі виконання бакалаврської кваліфікаційної роботи застосовувались: методи об'єктно–орієнтованого аналізу та проєктування програмних систем, порівняльний аналіз існуючих програмних рішень, метод компонентного проєктування інтерфейсу, UML–моделювання, а також методи функціонального та модульного тестування програмного забезпечення.

Наукова новизна одержаних результатів. У ході роботи запропоновано архітектурне рішення десктопного застосунку, що реалізує концепцію стрімінгового сервісу в офлайн–середовищі на основі компонентної структури Qt Widgets. Основна новизна полягає в оригінальному підході до перемикання теми інтерфейсу в режимі реального часу без перезапуску застосунку шляхом динамічної зміни QSS–таблиць стилів на рівні кореневого вікна – підхід, що рідко зустрічається в десктопних бакалаврських роботах і забезпечує миттєве оновлення всіх дочірніх компонентів через механізм успадкування стилів Qt.

Практична значущість одержаних результатів. Очікується, що застосунок CinemaHub зможе застосовуватись як самостійний інструмент для ведення особистої кінотеки в офлайн–режимі. Результати роботи демонструють повний цикл розробки десктопного програмного забезпечення – від аналізу аналогів і проєктування архітектури до реалізації та тестування – із використанням сучасних можливостей Qt 6.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Зиков М. В., Гавор А. С. Застосування мови C++ та фреймворку Qt для розробки десктопного застосунку з керування каталогом фільмів. // VII Міжнародна науково–технічна конференція «Сучасний стан та перспективи розвитку IoT».

2. Зиков М. В. Програмна реалізація десктопного застосунку CinemaHub для керування каталогом фільмів засобами C++ та Qt 6. // Всеукраїнська науково–технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація».

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Опис предметної галузі

Кіно для багатьох людей давно стало не просто способом відпочити, а повноцінним захопленням. Ті, хто регулярно переглядає фільми, рано чи пізно стикаються з однаковою проблемою: важко тримати в голові, що вже дивився, що планував подивитись, які фільми сподобались, а які ні. Зі зростанням особистої колекції ця проблема лише загострюється – окремі нотатки, таблиці або просто пам'ять перестають справлятись із таким обсягом інформації.

Задача цього проєкту – закрити саме цю потребу. Потрібен застосунок, який дозволяє швидко знайти потрібний фільм, переглянути про нього інформацію, позначити як обраний і навіть подивитись трейлер – і все це без інтернету, локально, на власному комп'ютері. Веб–сервіси типу IMDb або Letterboxd для цього не підходять: вони прив'язані до мережі, не дають повного контролю над даними і не мають десктопного клієнта.

Цільова аудиторія застосунку – люди, які активно стежать за кіно, формують власні списки перегляду і хочуть мати під рукою зручний інструмент без зайвих складнощів. Для них важливо: швидкий пошук, зрозумілий інтерфейс і можливість налаштувати застосунок під себе.

Предметна галузь має кілька особливостей, які вплинули на підхід до розробки. По–перше, кіноконтент оновлюється постійно – щороку виходять тисячі нових фільмів, тому каталог має бути структурованим і зручним для навігації навіть при великій кількості позицій. По–друге, частина аудиторії використовує пристрої без постійного доступу до мережі – в дорозі, у регіонах зі слабким інтернетом або просто за особистими вподобаннями. Для них важлива повна офлайн–функціональність. По–третє, люди, які серйозно ведуть облік переглянутого, цінують естетику інтерфейсу: застосунок, який виглядає застаріло, просто не буде використовуватись регулярно – навіть якщо технічно він працює правильно.

На основі цього аналізу при проєктуванні CinemaHub визначено три основні вимоги: зручна навігація у каталозі, повна автономність і сучасне оформлення.

Для розуміння загальної концепції і зовнішнього вигляду розробленого рішення на рисунку 1.1 наведено головне вікно застосунку CinemaHub – так він виглядає після авторизації користувача.

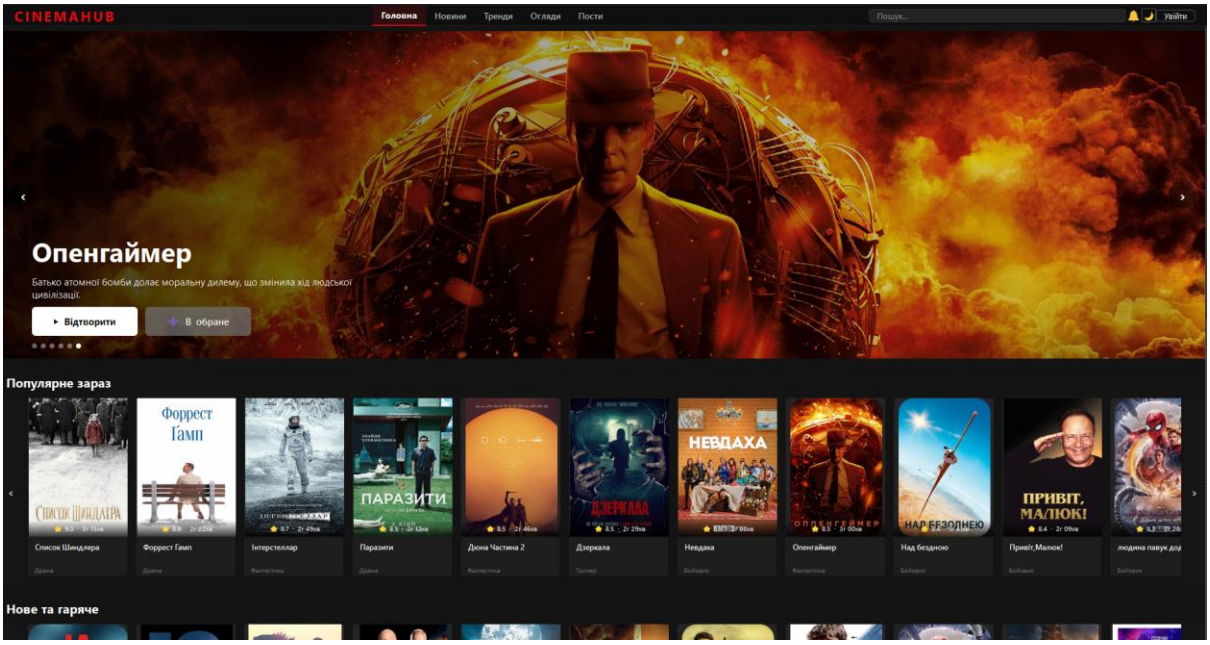


Рис. 1.1 Головне вікно застосунку CinemaHub

Головна сторінка спроєктована з урахуванням принципів сучасних стримінгових платформ. Верхню частину займає банер на повну ширину вікна – великий постер поточного фільму з назвою, коротким описом і двома кнопками: «Відтворити» і «В обране». Нижня частина сторінки містить горизонтальні ряди карток, кожен з яких сформований за окремим критерієм добірки. Інтерфейс головної сторінки наведено на рис. 1.1.

1.2 Аналіз існуючих програмних рішень

Перш ніж розпочати розробку, варто розібратись, що вже існує на ринку і чому це не вирішує задачу повністю.

IMDb – найбільша у світі база даних про кіно. Тут є все: акторський склад, рейтинги, рецензії, трейлери. Але це суто онлайн–платформа. Без інтернету вона

не працює взагалі, а десктопного застосунку як такого не існує – лише мобільний додаток і сайт. Для формування локальної кінотеки цей варіант не підходить.

Якщо розглянути IMDb детальніше з точки зору потреб, які вирішує CinemaHub, стає очевидним ключове обмеження: платформа проєктувалась як довідник, а не як інструмент для особистого відстеження. Там немає офлайн–режиму, немає можливості зберегти власні нотатки про фільм, а список «Watchlist» синхронізується через акаунт і стає недоступним без інтернету. Для людини, яка хоче мати локальний контроль над своєю кінотекою, це принципова проблема.

Letterboxd – цікава платформа для тих, хто хоче ділитися враженнями від фільмів. Але основна її функція – соціальна мережа, а не персональний каталог. Якщо людині не потрібна соціальна складова, більша частина функціоналу Letterboxd просто не буде використовуватись. Крім того, відсутність десктопного клієнта і залежність від мережі роблять його непридатним для офлайн–роботи.

Kodi – медіацентр із відкритим кодом, який підтримує локальні бібліотеки і відтворення відео. Технічно він може замінити повноцінний медіасервер, але налаштувати його під звичайного користувача – окреме випробування. Інтерфейс перевантажений, логіка роботи неочевидна, а зовнішній вигляд далекий від сучасних стандартів.

Kodi вирішує задачу відтворення контенту – але це медіаплеєр, а не менеджер каталогу. Щоб налаштувати Kodi для ведення кінотеки, потрібно встановити кілька плагінів, налаштувати сканування папок, розібратись з форматами метаданих. Для звичайного користувача, який хоче просто бачити список фільмів і шукати по ньому, це надлишкова складність.

Підтримка відео в MediaMonkey обмежена базовим відтворенням – немає ні пошуку за жанром, ні рейтингів, ні інформації про акторський склад. Фільтрація і пошук налаштовані під музику: альбоми, виконавці, роки виходу. Для відеоконтенту ці категорії не підходять, і користувачу доводиться шукати обхідні шляхи. Саме тому MediaMonkey не розглядається як повноцінне рішення для ведення кінотеки.

Варто окремо відзначити загальну тенденцію: більшість сучасних сервісів для роботи з кіноконтентом розробляються виключно як веб–платформи або мобільні додатки. Це пов'язано з тим, що хмарна модель дозволяє монетизувати продукт через підписку і збирати аналітику про поведінку користувача. Однак такий підхід означає, що без активного акаунту і стабільного інтернету користувач втрачає доступ до власних даних – списків переглянутого, оцінок, нотаток. Десктопний застосунок із локальним зберіганням позбавлений цієї залежності і залишається повністю функціональним у будь–яких умовах.

Ще одна спільна проблема розглянутих рішень – відсутність гнучкої персоналізації інтерфейсу. Жоден із аналізованих сервісів не підтримує перемикання теми оформлення. Для користувача, який проводить за застосунком тривалий час, це суттєво – особливо в умовах слабкого освітлення. CinemaHub вирішує цю задачу через вбудоване перемикання між темним і світлим режимами без перезапуску.

Таблиця 1.1

Порівняльна характеристика існуючих програмних рішень

Критерій	IMDb	Letterboxd	Kodi	MediaMonkey	CinemaHub
Робота офлайн	–	–	+	+	+
Десктопний застосунок	–	–	+	+	+
Пошук з автодоповненням	+	+	–	–	+
Відтворення відео	–	–	+	–	+
Сучасний інтерфейс	+	+	–	–	+
Персоналізація профілю	+	+	–	–	+
Фільтрація за жанром	+	+	+	–	+
Зміна теми	–	–	–	–	+

У таблиці 1.1 наведено узагальнене порівняння розглянутих рішень за основними критеріями.

З таблиці видно: рішення, що працює офлайн і при цьому має сучасний інтерфейс із пошуком, відео та персоналізацією – просто не існує. Саме цей пробіл і заповнює CinemaHub.

Варто також зупинитись на питанні збереження даних користувача. Всі розглянуті онлайн–платформи – IMDb, Letterboxd – зберігають дані облікових записів, списки переглянутого і оцінки на власних серверах. Якщо сервіс закриється або змінить умови використання, користувач ризикує втратити все, що збирав роками. Локальний десктопний застосунок позбавлений цього ризику: дані зберігаються на машині самого користувача і нікуди не передаються.

Окремо варто відзначити питання монетизації. Всі розглянуті платформи або мають платну підписку, або показують рекламу. CinemaHub – це повністю безкоштовний інструмент без прихованих обмежень і рекламних елементів в інтерфейсі.

1.3 Аналіз технологій та інструментів розробки

C++ обраний як основна мова розробки. Причина проста – він дає повний контроль над тим, що відбувається в програмі, забезпечує високу швидкодію і добре поєднується з Qt. Стандарт C++17 дозволяє писати чистий, структурований код із зручними сучасними конструкціями мови.

Qt 6 – це не просто інструментарій для розробки графічного інтерфейсу користувача (GUI). Це повноцінна екосистема: компоненти інтерфейсу, система сигналів і слотів для зв'язку між об'єктами, підтримка мультимедіа, стилізація через QSS–таблиці, інструменти для роботи з даними. Qt дозволяє будувати застосунки, що виглядають сучасно і працюють стабільно – саме те, що потрібно для CinemaHub.

Qt Multimedia – окремий модуль у складі Qt, завдяки якому в застосунку реалізовано вбудований відеоплеєр. Підключати сторонні бібліотеки для цього не знадобилось.

CMake використовується для збірки проєкту. Починаючи з Qt 6 це стандартний і рекомендований підхід – він добре інтегрується з Qt Creator і дозволяє гнучко налаштовувати структуру проєкту.

Qt Creator – середовище розробки, яке йде у комплекті з Qt. Зручний редактор форм, вбудований відладчик, підтримка CMake з коробки – все це суттєво прискорює розробку.

Під час вибору технологічного стеку також розглядались альтернативи. Electron на базі JavaScript і Node.js дозволяє будувати десктопні застосунки з веб–технологіями, але має суттєвий недолік: кожен застосунок містить вбудований браузерний рушій Chromium, що значно збільшує розмір дистрибутиву і споживання пам'яті. Для застосунку з великою кількістю зображень і медіаконтенту це неприйнятно. JavaFX на Java є непоганою альтернативою з точки зору кросплатформності, але поступається C++ у продуктивності і потребує встановлення JVM на цільовій машині. wxWidgets – зріла C++ бібліотека, але вона надає лише базові елементи інтерфейсу без власної системи стилізації, що ускладнює створення сучасного візуального оформлення.

Qt 6 відрізняється від попередньої версії Qt 5 у кількох важливих аспектах. По–перше, CMake став основною і рекомендованою системою збірки замість qmake. По–друге, відбулась реструктуризація модулів – частина застарілого функціоналу була прибрана, а нові модулі, зокрема Qt Multimedia, отримали суттєво оновлену архітектуру. По–третє, Qt 6 повністю підтримує стандарт C++17 і частково C++20, що дозволяє використовувати сучасні можливості мови без обмежень.

У проєкті CinemaHub можливості C++17 використовуються в кількох місцях. Лямбда–функції застосовуються при підключенні сигналів і слотів через *connect()* для коротких обробників подій. Ключове слово auto використовується для виведення типів при роботі з ітераторами і контейнерами Qt. Структуровані прив'язки спрощують роботу з парами значень. Ці можливості мови дозволяють писати більш компактний і читабельний код без втрати продуктивності.

Вибір між технологіями завжди є компромісом між продуктивністю, зручністю розробки і кінцевим розміром застосунку. Для застосунку, що працює з великою кількістю зображень, постерів і медіаданих, критично важливою є саме швидкість обробки і мінімальне споживання пам'яті. Electron на базі JavaScript не відповідає цій вимозі через вбудований браузерний рушій, JavaFX потребує окремого встановлення середовища виконання, а wxWidgets не має власної системи стилізації для створення сучасного інтерфейсу. Qt 6 у парі з C++17 задовольняє всі ці вимоги одночасно: висока продуктивність, вбудована підтримка відео через Qt Multimedia і повна стилізація через QSS без жодних зовнішніх залежностей.

У таблиці 1.2 порівняно Qt з альтернативними технологіями для реалізації десктопних застосунків.

Таблиця 1.2

Порівняння технологій для розробки десктопних застосунків

Критерій	Qt 6 / C++	wxWidgets / C++	Electron / JS	JavaFX / Java
Продуктивність	Висока	Висока	Низька	Середня
Вбудоване відео	+	–	+	+
Стилізація UI	Повна (QSS)	Обмежена	Повна (CSS)	Середня
Крос–платформність	+	+	+	+
Зрілість екосистеми	Висока	Середня	Висока	Висока
Розмір дистрибутиву	Середній	Малий	Великий	Середній

Qt 6 у парі з C++17 виявився найбільш збалансованим варіантом – він дає і продуктивність, і повноцінну підтримку мультимедіа, і гнучкі засоби стилізації інтерфейсу.

Важливий практичний критерій при виборі технологій – простота розповсюдження готового застосунку. Electron–застосунки містять вбудований браузерний рушій, і кінцевий дистрибутив важить від 150 до 300 МБ. JavaFX потребує встановленого JRE на цільовій машині, що є окремим кроком для користувача. Qt6 з C++ дозволяє зібрати застосунок із мінімальним набором залежностей – достатньо скопіювати папку зі зібраним проєктом, і він одразу запускається без додаткових встановлень.

Середовище Qt Creator суттєво прискорило розробку завдяки вбудованому візуальному редактору форм, інтегрованому відладчику і підтримці CMake з автоматичним виявленням залежностей. Виведення сигналів і слотів у режимі відлагодження дозволяє відстежувати з'єднання між об'єктами в режимі реального часу, що спрощує пошук логічних помилок у взаємодії компонентів.

Модуль Qt Multimedia заслуговує окремої уваги: він дозволяє відтворювати відеофайли всередині застосунку через класи *QMediaPlayer* і *QVideoWidget* без підключення сторонніх відеодекодерів. Це і використано в CinemaHub для реалізації резервного відеоплеєра – якщо для фільму не знайдено посилання на YouTube, відео відтворюється безпосередньо у застосунку.

Таким чином, обраний технологічний стек – C++17, Qt 6, CMake та SQLite – повністю відповідає вимогам до застосунку: висока продуктивність, офлайн–робота, сучасний інтерфейс і зручне розповсюдження без додаткових залежностей. Поєднання цих інструментів дозволило реалізувати застосунок, що за рівнем функціональності і візуальною якістю відповідає сучасним вимогам до десктопного програмного забезпечення.

Окремо варто відзначити, що всі обрані технології є відкритими або безкоштовними для некомерційного використання. Qt 6 поширюється за ліцензією LGPL, C++ є стандартизованою мовою без ліцензійних обмежень, CMake – відкрите програмне забезпечення, SQLite – повністю у публічному домені. Це означає, що розроблений застосунок може вільно розповсюджуватись і використовуватись без жодних правових обмежень. Саме такий підхід до вибору

інструментів є характерним для сучасних відкритих десктопних проєктів і забезпечує довгострокову підтримуваність розробки.

Таке поєднання технологій – C++17, Qt 6, CMake і SQLite – не тільки вирішує поставлені технічні задачі, але й закладає основу для можливого розширення застосунку в майбутньому. Модульна архітектура Qt і стандартизований інтерфейс SQLite дозволяють додавати нові функції без переписування вже існуючого коду.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Проєктування архітектури застосунку

До початку написання коду потрібно було вирішити, як застосунок влаштований зсередини. Якщо цього не зробити заздалегідь, потім дуже важко щось змінити або додати без того, щоб щось зламати.

В основу CinemaHub покладено компонентний підхід: кожен екран – це окремий клас із конкретною відповідальністю. Класи між собою напряму не пов'язані – вони спілкуються через механізм сигналів і слотів Qt. Це дає можливість змінювати один компонент, не чіпаючи решту.

При проєктуванні архітектури застосовано метод об'єктно–орієнтованого аналізу: систему розбито на класи з чіткими межами відповідальності. Взаємодія між класами описана за допомогою UML–нотації – структурна схема компонентів і діаграма класів наведені у підрозділах 2.1 та 3.1. Компонентний підхід до проєктування інтерфейсу забезпечив незалежність модулів і спростив подальше тестування.

Застосунок реалізовано у вигляді дев'яти програмних класів, кожен з яких інкапсулює окремий функціональний модуль системи.

Клас IntroSplash відповідає за відображення анімованої заставки при запуску застосунку. Після завершення анімації керування передається модулю авторизації.

Клас LoginWindow реалізує форми входу та реєстрації користувача. Перевірка облікових даних виконується через модуль управління користувачами, після успішної авторизації відкривається головне вікно.

Клас MainWindow є центральним класом застосунку і забезпечує навігацію між усіма розділами: головна сторінка, новини, тренди, огляди, пости, пошук, сповіщення.

Клас MovieCard реалізує візуальне представлення окремого фільму у вигляді картки з постером, назвою та жанром. Компонент використовується у всіх розділах, де відображається перелік фільмів.

Клас ScrollRow реалізує горизонтальний контейнер карток з підтримкою плавного прокручування. Застосовується для формування тематичних підбірок на головній сторінці та в розділі «Тренди».

Клас MovieDetailsWindow відповідає за відображення повної інформації про фільм: постер, опис, акторський склад, відгуки, схожі стрічки та запуск трейлера.

Клас VideoPlayer є резервним засобом відтворення відеоматеріалів. Активується у випадку відсутності посилання на YouTube для конкретного фільму.

Клас ProfileWindow реалізує екран персонального профілю користувача з можливістю зміни нікнейму, кольору аватара та завантаження фотографії.

Клас MovieData є модулем даних без графічного інтерфейсу. Він зберігає каталог фільмів і надає методи для виконання пошуку, фільтрації та отримання запису за ідентифікатором. Всі інші компоненти звертаються до даних виключно через цей модуль.

Клас UserManager є singleton–компонентом без графічного інтерфейсу. Він відповідає за зберігання поточної сесії користувача: перевірку логіна і пароля при вході, реєстрацію нових акаунтів, зберігання шляху до аватара. Дані акаунтів зберігаються локально через QSettings. Клас реалізовано за патерном одиначки – у системі існує лише один екземпляр.

Клас NotificationsWindow відображає персоналізовані повідомлення для користувача. Вікно будується динамічно у методі buildUi() і підтримує різні категорії сповіщень з кольоровими мітками.

Клас SearchWindow реалізує окремий екран розширеного пошуку з фільтрацією за жанром і сортуванням результатів. Використовується як альтернатива до вбудованого пошуку у навігаційній панелі.

Структурну схему компонентів застосунку наведено на рис. 2.1.

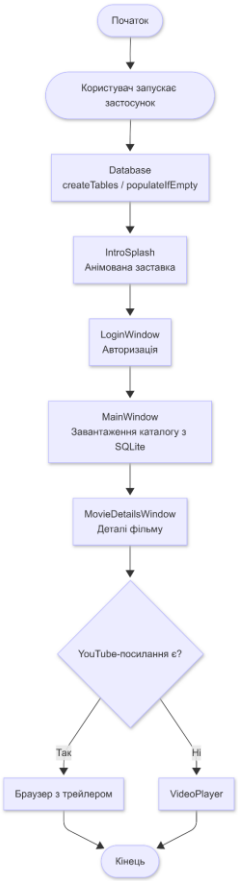


Рис. 2.1 Структурна схема компонентів застосунку CinemaHub

Важливо відзначити, що такий поділ на компоненти відповідає принципу єдиної відповідальності – кожен клас відповідає рівно за одну частину системи. *MainWindow* не знає як влаштована картка фільму всередині, а *MovieCard* не знає нічого про навігацію між розділами. Зв'язок між ними існує тільки через сигнал *movieClicked*, який *MovieCard* випускає при кліку, і *MainWindow* обробляє. Це типовий приклад слабкої зв'язаності між компонентами, що є ознакою якісної архітектури.

Окремого пояснення потребує клас *MovieData*. Він не є візуальним компонентом і не успадковується від жодного Qt–класу інтерфейсу. Фактично це модуль даних: він зберігає весь каталог фільмів у вигляді *QVector<Movie>* і надає функції для роботи з ним. Будь–який інший компонент, якому потрібні дані про фільми, просто викликає функції *MovieData* – і не знає, звідки ці дані беруться. Якщо в майбутньому захочеться замінити зберігання з коду на базу даних – достатньо змінити тільки *MovieData*, не торкаючись решти.

Порядок ініціалізації компонентів при запуску є окремою частиною архітектурного рішення. При відкритті застосунку виконується чітко визначена послідовність: спочатку запускається базовий об'єкт Qt–застосунку, без якого жоден компонент не може функціонувати. Після цього відображається анімована заставка. Лише після завершення анімації відкривається вікно авторизації. І тільки після успішного входу користувача ініціалізується і відкривається головне вікно застосунку.

Такий підхід до архітектури обрано з кількох причин. По–перше, кожен клас можна розробляти і тестувати незалежно, не чіпаючи решту. По–друге, якщо в майбутньому потрібно замінити, наприклад, вікно профілю або додати новий розділ – достатньо змінити один компонент, а не переписувати логіку в кількох місцях. По–третє, механізм сигналів і слотів Qt ідеально підходить для такої моделі: компонент не знає, хто його слухає, він просто надсилає сигнал – і *MainWindow* реагує. Це зменшує зв'язаність між класами і робить код більш передбачуваним.

Такий підхід до організації застосунку дає кілька практичних переваг. По–перше, якщо потрібно щось змінити в одному екрані, наприклад, додати нове поле до профілю – не треба торкатись решти коду. По–друге, кожен клас можна тестувати окремо. По–третє, додати новий розділ до застосунку – це просто створити новий клас і підключити його до MainWindow через стандартний *connect()*. Саме такою гнучкістю і виправдовується компонентна архітектура для цього проєкту.

Структуру всіх файлів проєкту зручно переглянути у Qt Creator. Дерево файлів відображає кожен клас окремо – заголовочний файл і файл реалізації.

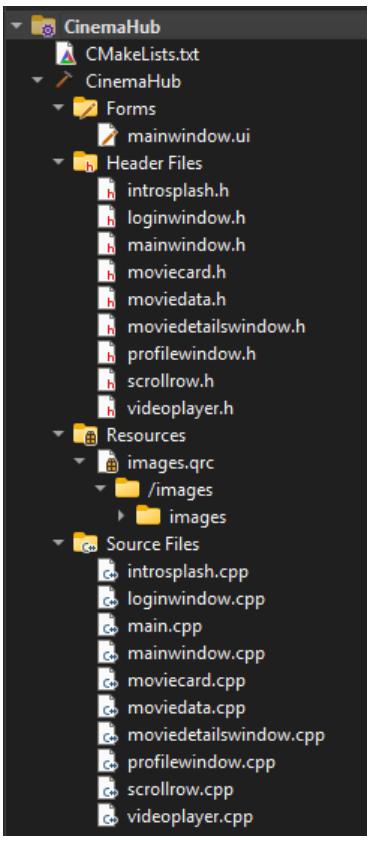


Рис. 2.2 Структура файлів проєкту CinemaHub у Qt Creator

2.2 Моделювання вимог

Перед тим як проєктувати інтерфейс і писати логіку, потрібно було чітко визначити – що взагалі повинен робити застосунок. Для цього побудовано діаграму варіантів використання, яка описує всі дії, доступні користувачу.

В системі один актор – Користувач. Йому доступні такі варіанти використання: авторизація; реєстрація; перегляд каталогу; перегляд трендів; перегляд новин; пошук фільмів; фільтрація за жанром; перегляд інформації про фільм; відтворення трейлера; додавання до обраного; редагування профілю; перемикання теми інтерфейсу.

Діаграма варіантів використання наведена на рисунку 2.3.

Рис. 2.3 Діаграма варіантів використання CinemaHub

На основі варіантів використання сформульовано функціональні вимоги, наведені в таблиці 2.1.

Таблиця 2.1

Функціональні вимоги до застосунку

№	Вимога	Пріоритет
ФВ–01	Відображати анімовану заставку при кожному запуску	Високий
ФВ–02	Підтримувати реєстрацію та авторизацію користувача	Високий
ФВ–03	Відображати банерний карусель на головному екрані	Високий
ФВ–04	Відображати каталог у вигляді карток із постерами	Високий

Продовження таблиці 2.1

№	Вимога	Пріоритет
ФВ–05	Забезпечити пошук з автодоповненням у реальному час	Високий
ФВ–06	Фільтрувати фільми за жанром	Середній
ФВ–07	Відтворювати трейлери фільмів	Високий
ФВ–08	Додавати фільми до списку обраного	Середній
ФВ–09	Редагувати профіль користувача	Середній
ФВ–10	Перемикати темну та світлу тему без перезапуску	Середній

Окрім функціональних, визначено нефункціональні вимоги до застосунку, які описують якісні характеристики його роботи. Застосунок повинен функціонувати повністю в офлайн–режимі – жоден із основних сценаріїв роботи не може залежати від мережевого з'єднання. Виняток становить лише відкриття трейлера на YouTube, яке потребує браузера і інтернету, але не є обов'язковою функцією для роботи з каталогом. Час відгуку інтерфейсу при навігації між розділами і виконанні пошуку не повинен перевищувати 200 мс – це забезпечується тим, що всі дані зберігаються локально і не потребують мережевих запитів. Перемикання теми інтерфейсу має відбуватись миттєво, без перезапуску або затримки. Застосунок повинен коректно відображатись на екранах з роздільною здатністю від 1280×720 пікселів і вище.

Для реалізації наведених вимог обрано конкретні технічні підходи. Офлайн–режим забезпечується зберіганням усіх даних каталогу у локальній базі даних SQLite – жодного мережевого запиту застосунок не виконує. Єдиний виняток – відкриття трейлера через системний браузер, але це не є обов'язковою функцією

для роботи з каталогом. Час відгуку менше 200 мс досягається тим, що пошук і фільтрація виконуються в оперативній пам'яті без звернення до файлової системи. Миттєве перемикання теми реалізовано через єдиний виклик *setStyleSheet()* на рівні *MainWindow* – стиль автоматично успадковується всіма дочірніми елементами.

Вимога до мінімальної роздільної здатності 1280×720 пікселів враховувалась при визначенні розмірів усіх ключових елементів інтерфейсу: ширина карток у рядах, розмір шрифту, відступи між елементами. На цій роздільній здатності інтерфейс залишається повністю читабельним і функціональним.

2.3 Проєктування структури даних

Для зберігання каталогу фільмів у застосунку CinemaHub застосовано вбудовану реляційну СУБД SQLite через модуль Qt6::Sql. SQLite зберігає всі дані в єдиному файлі cinema.db без потреби в окремому серверному процесі, що повністю відповідає вимозі офлайн–роботи: база даних доступна без мережевого підключення.

При першому запуску клас Database автоматично створює файл cinema.db у системній директорії користувача (AppData\Roaming\CinemaHub\), формує таблиці movies і favorites та заповнює таблицю movies 103 записами. Усі операції вставки виконуються в межах однієї SQL–транзакції (BEGIN TRANSACTION… COMMIT), що забезпечує цілісність даних і швидкість першого запуску.

Читання даних реалізовано у функції loadFromDB() модуля moviedata.cpp через QSqlQuery. Функція виконується лише один раз за сесію завдяки прапорцю g_loaded. Ключова перевага такого підходу – можливість редагувати каталог фільмів безпосередньо в базі даних через DB Browser for SQLite без перекомпіляції застосунку.

Структура Movie оголошена у файлі *moviedata.h*.

```
struct Movie {
    int         id;
    QString     title;
    QString     genre;
    int         year;
    QString     description;
    QString     imagePath;
    QStringList cast;
    float       rating   = 0.0f;
    int         duration = 0;
};
```

Рис. 2.4 Оголошення структури Movie у файлі moviedata.h

Поля структури описано в таблиці 2.2.

Таблиця 2.2

Нефункціональні вимоги до застосунку

Поле	Тип	Опис
id	int	Унікальний числовий ідентифікатор
title	*QString*	Назва фільму
genre	*QString*	Жанр
year	int	Рік виходу
description	*QString*	Короткий опис сюжету
imagePath	*QString*	Шлях до файлу постера
cast	*QStringList*	Список акторів
rating	float	Оцінка від 0.0 до 10.0
duration	int	Тривалість у хвилинах

Модуль зберігання даних надає функції для роботи з каталогом: отримання повного списку, пошук за ідентифікатором, вибірка за жанром та повнотекстовий пошук за назвою .

Для зберігання списку обраних фільмів використовується структура даних типу множина «QSet», що гарантує унікальність збережених числових ідентифікаторів та забезпечує константний час пошуку. Додавання і видалення з обраного реалізовано через спеціалізовані функції перемикання і перевірки стану.

Ресурси застосунку – постери фільмів і відеофайли трейлерів – організовано через систему *Qt Resource System*. Постери підключено через файл *images.qrc* і компілюються безпосередньо у виконуваний файл, що виключає проблеми з відносними шляхами при запуску застосунку в різних середовищах. Відеофайли розміщено в окремій папці videos і розгортаються поруч із виконуваним файлом при збірці через команду *add_custom_command* у *CMakeLists.txt*. Такий підхід спрощує розповсюдження застосунку – достатньо скопіювати папку із зібраним проєктом, і всі ресурси вже на місці.

Окремого пояснення потребує вибір структури даних типу множина для зберігання списку обраних фільмів. Контейнер без дублікатів гарантує: один і той самий фільм не може бути доданий до обраного двічі. Операція перевірки наявності елемента виконується за константний час, що важливо для плавної роботи при наведенні курсору на картки. Функція перемикання стану або додає ідентифікатор до множини, або видаляє його – залежно від поточного стану.

Більш складні структури – наприклад, зв'язаний список або словник – не дали б жодних переваг при поточному обсязі каталогу, але додали б складності коду без практичної користі.

2.4 Проєктування інтерфейсу користувача

При проєктуванні інтерфейсу CinemaHub орієнтиром слугував стиль сучасних стрімінгових платформ – темний фон, червоні акцентні кольори і картковий підхід до відображення контенту. Цей вибір обумовлений тим, що

більшість цільової аудиторії знайомі з подібними сервісами і одразу розуміють принципи навігації без додаткового навчання.

Сценарій запуску застосунку побудовано у три кроки: анімована заставка – вікно авторизації – головний екран. Такий порядок забезпечує ідентифікацію користувача до відкриття основного інтерфейсу і є стандартним підходом для застосунків із персоналізацією. Заставка виконує функцію первинного враження і задає загальний візуальний тон.

Навігація між розділами організована через горизонтальне меню у верхній частині вікна. Такий підхід обрано тому, що кількість розділів фіксована і не перевищує семи пунктів – вони вільно розміщуються у верхній панелі без прокручування. Праворуч у панелі розміщено функціональні елементи: поле пошуку, кнопка сповіщень, перемикач теми і кнопка профілю. Це забезпечує доступ до ключових функцій з будь–якого екрана застосунку.

Фільми відображаються у вигляді карток фіксованого розміру. На головній сторінці і в розділі «Тренди» картки згруповано в горизонтальні ряди за жанрами з кнопками–стрілками для прокручування. У розділі «Пости» весь каталог відображається у вигляді рівномірної сітки. Картка містить постер, назву, жанр, а також інформаційну смугу з рейтингом, роком і тривалістю фільму.

Застосунок підтримує дві теми – темну і світлу. Темна тема є основною і відповідає стилістиці кінематографічної тематики. Світла тема передбачена для використання в умовах яскравого освітлення. Перемикання між темами відбувається через одну кнопку у навігаційній панелі і застосовується до всього інтерфейсу одночасно через механізм QSS–таблиць стилів.

При проєктуванні кольорової схеми визначено єдиний акцентний колір – насичений червоний відтінок, який добре читається як на темному, так і на світлому фоні. У темній темі основний фон виконаний у майже чорному кольорі, картки і панелі – у темно–сірому відтінку. У світлій темі фон змінюється на нейтральний світло–сірий, текст – на майже чорний. Акцентний червоний колір залишається незмінним в обох темах, що забезпечує візуальну єдність інтерфейсу. Зовнішній вигляд темної теми наведено на рис. 3.27, світлої – на рис. 3.21.

Навігаційна панель реалізована у вигляді горизонтального контейнера компонентів з фіксованою висотою. Ліворуч – логотип і кнопки розділів, праворуч – поле пошуку, кнопка теми і кнопка–аватар профілю. Такий поділ між навігацією за контентом і налаштуваннями є стандартним підходом для більшості сучасних сервісів і одразу зрозумілий без пояснень.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура проєкту та діаграма класів

Проєкт CinemaHub складається з тринадцяти класів, кожен з яких реалізовано окремою парою файлів – заголовочним *(.h)* і файлом реалізації *(.cpp)*. Ресурси застосунку підключено через *Qt Resource System*.

Структуру файлів проєкту наведено на рис. 2.2.

Діаграма класів відображає склад кожного класу та зв'язки між ними. Центральне місце займає *MainWindow*, від якого залежать усі інші компоненти. *MovieData* виступає як постачальник даних без графічного представлення.

Зв'язки між класами реалізовано через механізм сигналів і слотів: наприклад, Картка фільму надсилає сигнал вибору, а головне вікно або вікно деталей фільму його обробляють. Такий підхід дає змогу змінювати один клас, не торкаючись решти – кожен компонент знає тільки про свої власні дії і нічого про те, хто на них реагує.

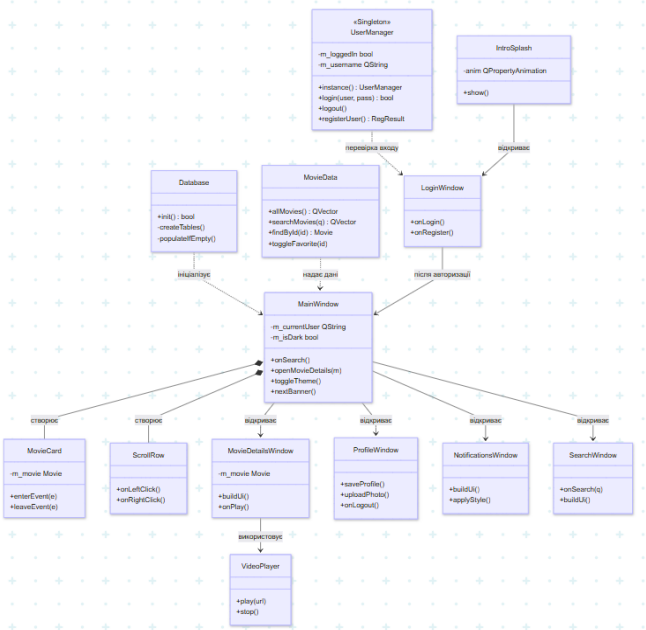


Рис. 3.1 Діаграма класів застосунку CinemaHub

Діаграма відображає всі тринадцять компонентів: Database, UserManager, IntroSplash, LoginWindow, MainWindow, MovieCard, ScrollRow,

MovieDetailsWindow, ProfileWindow, VideoPlayer, NotificationsWindow, SearchWindow і MovieData.

3.2 Реалізація анімованої заставки

Клас *IntroSplash* реалізує перший екран після запуску. Вікно створюється без рамки та заголовку через прапорець *Qt::FramelessWindowHint*. Анімація реалізована через *QPropertyAnimation* – прозорість вікна змінюється від нуля до максимуму і назад. Після завершення *QTimer* закриває заставку і відкриває *LoginWindow*. Екранну форму заставки наведено на рис. 3.3.

```
m_safetyTimer = new QTimer(this);
m_safetyTimer->setSingleShot(true);
m_safetyTimer->setInterval(6000);
connect(m_safetyTimer, &QTimer::timeout, this, &IntroSplash::doFadeOut);
m_safetyTimer->start();

m_effect = new QGraphicsOpacityEffect(this);
m_effect->setOpacity(1.0);
setGraphicsEffect(m_effect);
```

```
auto* anim = new QPropertyAnimation(m_effect, "opacity", this);
anim->setDuration(600);
anim->setStartValue(1.0);
anim->setEndValue(0.0);
anim->setEasingCurve(QEasingCurve::InOutQuad);
connect(anim, &QPropertyAnimation::finished, this, [this]() {
    emit finished();
});
anim->start(QAbstractAnimation::DeleteWhenStopped);
```

Рис. 3.2 Реалізація анімації у introsplash.cpp

Заставка займає весь екран і не має стандартної рамки вікна – це досягається встановленням прапорця *Qt::FramelessWindowHint* і розгортанням вікна на весь екран. Логотип CinemaHub відображається по центру на чорному фоні. Анімація складається з двох фаз: логотип поступово з'являється протягом першої секунди, потім так само поступово зникає. Після завершення анімації вікно закривається автоматично і одразу відкривається вікно авторизації. Користувач не може пропустити заставку вручну – це зроблено навмисно, щоб задати настрій перед входом в застосунок.

Рис. 3.3 Анімована заставка при запуску застосунку

На рисунку 3.3 видно вигляд заставки під час анімації – чорний фон і логотип по центру. Вікно не має стандартної рамки і заголовку, що дозволяє зосередити увагу виключно на логотипі. Тривалість анімації підібрана так, щоб заставка не тривала занадто довго – користувач бачить її кілька секунд і одразу потрапляє до вікна авторизації.

3.3 Реалізація системи авторизації

Авторизацію реалізовано у вигляді діалогового вікна. В межах одного вікна реалізовано дві форми – вхід до акаунту та реєстрація нового користувача. Перемикання між формами відбувається без закриття вікна шляхом показу та приховування відповідних елементів інтерфейсу.

При натисканні кнопки входу виконується перевірка введених облікових даних через клас *UserManager*. У разі успішної аутентифікації вікно закривається викликом методу *accept()*, що слугує сигналом для *main.cpp* відкрити головне вікно застосунку. При реєстрації перевіряється унікальність логіну та збіг введених паролів; у разі невідповідності користувачу виводиться повідомлення про помилку.

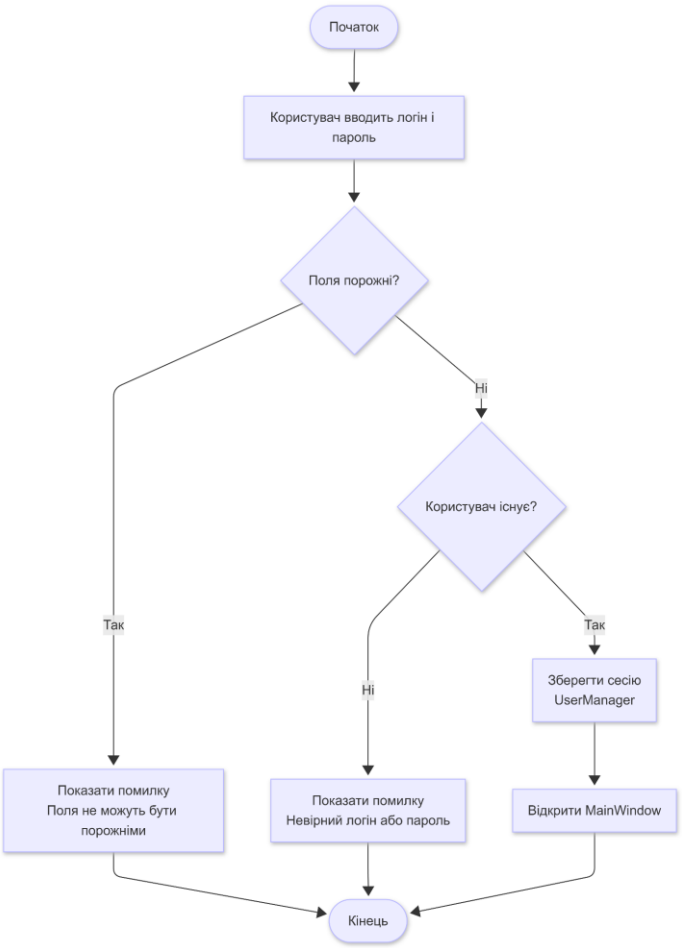


Рис. 3.4 Блок-схема алгоритму авторизації у LoginWindow

Вікно авторизації витримане в темному стилі з червоними акцентами. Зверху відображається логотип CinemaHub, нижче – заголовок «Вхід в акаунт». Форма містить два поля: «Логін» і «Пароль». Поле пароля приховує введені символи. При некоректних даних під полями з'являється червоне повідомлення про помилку, а поле пароля очищується і отримує фокус. Кнопка «Увійти» виконана з білим фоном і чорним текстом – це головна дія на екрані, і вона виділяється на темному фоні.

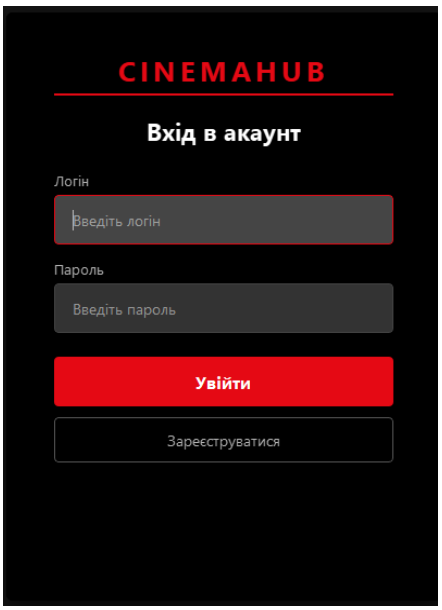


Рис. 3.5 Екранна форма входу в акаунт

На рисунку 3.5 наведено екранну форму входу. Вікно виконано в темному стилі з червоними акцентами. Поле пароля приховує введені символи. При некоректних даних між полями з'являється повідомлення про помилку червоним кольором, а поле пароля автоматично очищується. Кнопка «Увійти» є головною дією і виділяється білим кольором на темному фоні.

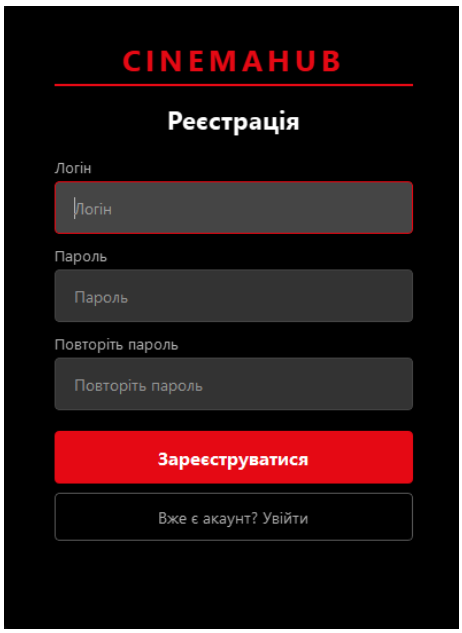


Рис. 3.6 Екранна форма реєстрації нового користувача

На рисунку 3.6 наведено форму реєстрації. Вона містить три поля: логін, пароль і повторення пароля. Перевірка збігу паролів і унікальності логіну відбувається на стороні застосунку до збереження даних. Форма знаходиться в тому самому вікні що й форма входу – перемикання між ними відбувається без закриття вікна.

3.4 Реалізація головного вікна та навігації

Головний клас застосунку є центральним вузлом навігації. Перемикання між сторінками реалізовано через стековий механізм – при натисканні кнопки меню активується відповідна сторінка без перебудови інтерфейсу. Підключення кнопок виконано у конструкторі через механізм сигналів і слотів. Реалізацію підключення навігаційних кнопок наведено на рисунку 3.7.

```
connect(ui->btnHome,   &QPushButton::clicked, this, [this]{ showPage(ui->pageHome);   });
connect(ui->btnNews,   &QPushButton::clicked, this, [this]{ showPage(ui->pageNews);   });
connect(ui->btnTrends, &QPushButton::clicked, this, [this]{ showPage(ui->pageTrends); });
connect(ui->btnNotif,    &QPushButton::clicked, this, [this]{ showPage(ui->pageNotifications); });
connect(ui->btnReviews,  &QPushButton::clicked, this, [this]{ showPage(ui->pageReviews);   });
connect(ui->btnPosters,  &QPushButton::clicked, this, [this]{ showPage(ui->pagePosters);   });
connect(ui->btnLogin,  &QPushButton::clicked, this, &MainWindow::onLoginClicked);
connect(ui->btnAvatar, &QPushButton::clicked, this, &MainWindow::onAvatarClicked);
connect(ui->searchEdit, &QLineEdit::textChanged, this, &MainWindow::onSearch);
connect(ui->btnTheme,   &QPushButton::clicked,     this, &MainWindow::toggleTheme);
```

Рис. 3.7 Підключення навігаційних кнопок у mainwindow.cpp

Банерний карусель реалізовано через *QTimer*. З кожним спрацюванням таймера викликається *nextBanner()*. Метод перевіряє, чи не порожній масив банерів, після чого зчитує поточний елемент і оновлює заголовок та опис та завантажує відповідне зображення банера через ресурсну систему Qt. Індекс інкрементується в іншій частині коду і скидається до нуля при виході за межі масиву. Реалізацію методу наведено на рисунку 3.8. Екранну форму головного вікна наведено на рис. 1.1.

Окрім банерного каруселя, головна сторінка містить чотири тематичних ряди карток, які формуються автоматично на основі даних каталогу. Перший ряд – «Популярне зараз» – відображає двадцять фільмів із найвищим рейтингом, відсортованих за спаданням. Другий ряд – «Нове та гаряче» – показує двадцять

останніх доданих позицій за зворотним порядком ідентифікаторів. Третій ряд – «Топ за рейтингом» – включає всі фільми з оцінкою 8.0 і вище. Четвертий ряд – «Рекомендовано для тебе» – формується як вибірка через кожен третій фільм каталогу, що забезпечує різноманітність жанрів без повторення з попередніми рядами. Кожен ряд реалізовано через клас *ScrollRow* і підтримує горизонтальне прокручування зі стрілками.

```
void MainWindow::nextBanner()
{
    if (m_banners.isEmpty() || !m_bannerImage) return;

    const BannerData& bd = m_banners[m_bannerIdx];
    m_bannerTitle->setText(bd.title);
    m_bannerDesc->setText(bd.desc);
```

Рис. 3.8 Реалізація методу nextBanner()

3.5 Реалізація карток фільмів та горизонтального прокручування

Картка фільму реалізована як самостійний візуальний компонент. При натисканні лівою кнопкою миші генерується сигнал вибору з ідентифікатором фільму. Обробники наведення і виходу курсору застосовують або скасовують підсвічування картки. Реалізацію обробників подій наведено на рисунку 3.9. Екранну форму ряду карток із ефектом наведення наведено на рис. 3.10.

```
void MovieCard::mousePressEvent(QMouseEvent* e)
{
    if (e->button() == Qt::LeftButton) emit movieClicked(m_movie.id);
    QFrame::mousePressEvent(e);
}
void MovieCard::enterEvent(QEnterEvent* e) { applyHover(true);  QFrame::enterEvent(e); }
void MovieCard::leaveEvent(QEvent* e)      { applyHover(false); QFrame::leaveEvent(e); }
```

Рис. 3.9 Реалізація обробників подій у moviecard.cpp

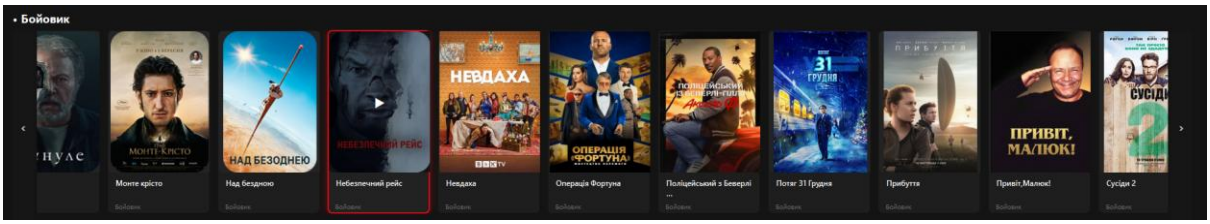


Рис. 3.10 Екранна форма горизонтального ряду карток фільмів

На рисунку 3.10 показано горизонтальний ряд карток з ефектом наведення. При наведенні курсору картка підсвічується червоною рамкою з тінню і по центру постера з'являється іконка відтворення. Також у правому верхньому куті постера з'являється кнопка серця для додавання до обраного. Ця кнопка залишається видимою постійно, якщо фільм вже додано до обраного.

Горизонтальне прокручування рядів реалізовано через окремий компонент. Методи прокручування перевіряють стан анімації – якщо вона вже виконується, зупиняють її. Після цього встановлюються початкове і кінцеве значення анімації з урахуванням граничних значень смуги прокручування. Реалізацію методів наведено на рисунку 3.11.

```
void ScrollRow::scrollLeft()
{
    if (m_anim->state() == QAbstractAnimation::Running) m_anim->st
    int cur = m_scroll->horizontalScrollBar()->value();
    m_anim->setStartValue(cur);
    m_anim->setEndValue(qMax(0, cur - STEP));
    m_anim->start();
}

void ScrollRow::scrollRight()
{
    if (m_anim->state() == QAbstractAnimation::Running) m_anim->st
    int cur = m_scroll->horizontalScrollBar()->value();
    int mx  = m_scroll->horizontalScrollBar()->maximum();
    m_anim->setStartValue(cur);
    m_anim->setEndValue(qMin(mx, cur + STEP));
    m_anim->start();
```

Рис. 3.11 Реалізація методів прокручування у scrollrow.cpp

3.6 Реалізація пошуку з автодоповненням

Ініціалізація системи автодоповнення відбувається під час завантаження головного модуля. Для цього алгоритм формує масив доступних назв шляхом повного обходу локального каталогу фільмів. Список передається до *QCompleter* із налаштуваннями: нечутливість до регістру, режим пошуку за входженням (*MatchContains*), спливаючий список підказок та максимум вісім видимих елементів. Реалізацію методу наведено на рисунку 3.12.

```
void MainWindow::setupAutocomplete()
{
    QStringList titles;
    for (const Movie& mv : allMovies())
        titles.append(mv.title);

    m_completer = new QCompleter(titles, this);
    m_completer->setCaseSensitivity(Qt::CaseInsensitive);
    m_completer->setFilterMode(Qt::MatchContains);
    m_completer->setCompletionMode(QCompleter::PopupCompletion);
    m_completer->setMaxVisibleItems(8);
```

Рис. 3.12 Реалізація методу setupAutocomplete() у mainwindow.cpp

Логіку пошуку реалізовано у методі *onSearch()*. При порожньому запиті відображається головна сторінка. В іншому разі перемикається сторінка пошуку, очищуються попередні результати, викликається *searchMovies()*. Якщо результатів немає – відображається відповідне повідомлення. Якщо є – для кожного фільму створюється *MovieCard* і додається до макету. Реалізацію методу наведено на рисунку 3.13. Екранну форму пошуку наведено на рис. 3.14.

Реалізований механізм пошуку не залежить від мережевого підключення – уся фільтрація відбувається локально на основі даних, завантажених з SQLite при старті. Це означає, що навіть без інтернету пошук працює миттєво і повертає актуальні результати. Порівняно з онлайн–сервісами, де пошук потребує мережевого запиту, такий підхід забезпечує нижчу затримку і стабільнішу роботу незалежно від якості з'єднання.

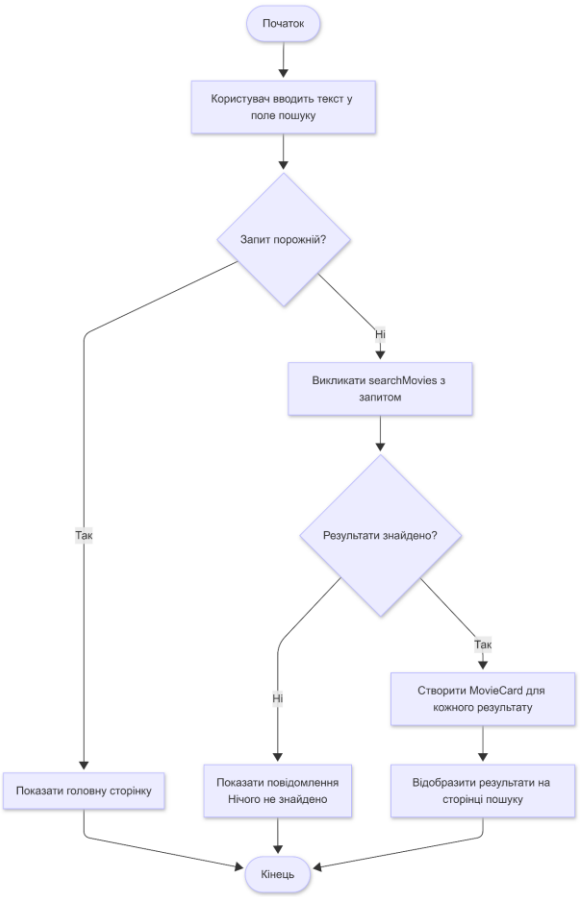


Рис. 3.13 Блок-схема алгоритму пошуку onSearch()

При введенні тексту в поле пошуку список підказок з'являється миттєво – без затримки і без натискання Enter. Це досягається тим, що *QCompleter* налаштований у режимі *PopupCompletion* і підключений до сигналу *textChanged*. Підказки фільтруються за входженням рядка – тобто «бу» знайде і «Буча», і «Довбуш», і будь–яке інше слово що містить ці букви. При виборі підказки кліком текст підставляється у поле і пошук запускається автоматично.

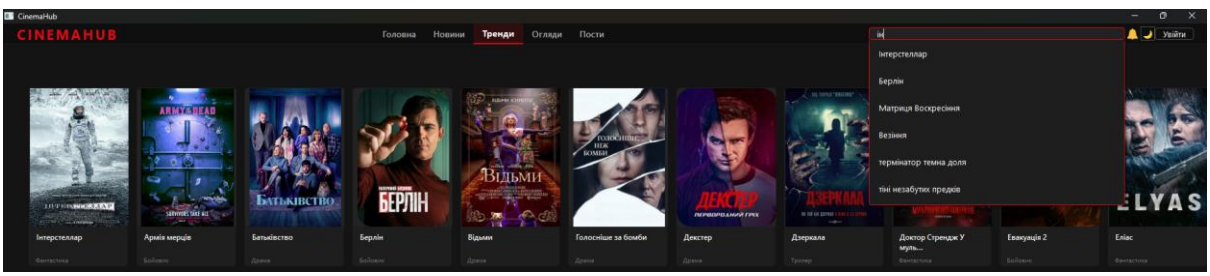


Рис. 3.14 Екранна форма пошуку з автодоповненням

На рисунку 3.14 показано роботу пошуку з автодоповненням. У верхній частині видно поле пошуку з введеними символами і список підказок під ним. Нижче відображаються результати – картки знайдених фільмів і заголовок із кількістю знайдених результатів. Підказки з'являються миттєво при кожному введеному символі без необхідності натискати Enter.

3.7 Реалізація сторінки фільму та відкриття трейлера

Вікно MovieDetailsWindow будується динамічно у методі buildUi(). Вміст розміщено у QScrollArea, що дозволяє прокручувати сторінку. Послідовно відображаються блок відеозаглушки, інформація про фільм, горизонтальний ряд схожих фільмів того самого жанру та блок відгуків глядачів.

Відтворення трейлера реалізовано через окремий обробник події натискання кнопки. Алгоритм шукає посилання на YouTube за назвою фільму у словнику із майже ста записів. Якщо посилання знайдено – воно передається системному браузеру для відкриття. Якщо ні – запускається резервний відеоплеєр. Реалізацію методу наведено на рисунку 3.16. Екранну форму сторінки фільму наведено на рис. 3.17, трейлер у браузері – на рис. 3.18.

Всі елементи сторінки фільму будуються динамічно у методі buildUi() кожного разу при відкритті вікна. Це означає, що дані завжди актуальні – якщо каталог оновлюється через базу даних, наступне відкриття сторінки покаже вже нові дані без перезапуску застосунку. Вікно підтримує прокручування через QScrollArea, що дозволяє розмістити необмежену кількість схожих фільмів і відгуків без обмежень висоти.

```
static QString trailerUrl(const QString& title)
{
    static const struct { const char* key; const char* url; } T[] = {
        {"Інтерстеллар",                    "https://www.youtube.com/watch?v=zSWdZVtXT7E"},
        {"Ірландець",                        "https://www.youtube.com/watch?v=WHXxVmeGQUc"},
        {"Аватар Шлях води",                "https://www.youtube.com/watch?v=d9MyW72ELq0"},
        {"Армія мерців",                    "https://www.youtube.com/watch?v=K4RhR2zmCUc"},
        {"Батьківство",                      "https://www.youtube.com/watch?v=2OaRmYmKYjE"},
        {"Берлін",                          "https://www.youtube.com/watch?v=N3viMjGI5Bc"},
        {"Бетмен",                          "https://www.youtube.com/watch?v=mqqft2x_Aa4"},
        {"Богемна рапсодія",                "https://www.youtube.com/watch?v=mP0VHJYFOAU"},
        {"Буча",                            "https://www.youtube.com/watch?v=9DPfJFSNQXM"},
        {"Відьми",                          "https://www.youtube.com/watch?v=8gEABCJDjQA"},
        {"Голосніше за бомби",              "https://www.youtube.com/watch?v=N4dEbPjUqaY"},
        {"Гравітація",                      "https://www.youtube.com/watch?v=OiTiKOy59o4"},
        {"Гран туризмо",                    "https://www.youtube.com/watch?v=jC9aWdSFLvs"},
        {"Гра в кальмара",                  "https://www.youtube.com/watch?v=oqxAJKy0ii4"},
        {"Декстер",                         "https://www.youtube.com/watch?v=2-k0gBCeaD0"},
        {"Джокер",                          "https://www.youtube.com/watch?v=xCOkXLQYmCk"},
        {"Дзеркала",                        "https://www.youtube.com/watch?v=6c5V-iDWkxw"},
        {"Довбуш",                          "https://www.youtube.com/watch?v=BkOUz9k_MuU"},
        {"Доктор Стрендж",                  "https://www.youtube.com/watch?v=aWzlQ2N6qqg"},
        {"Думками навиворіт",               "https://www.youtube.com/watch?v=LEjhY15eCx0"},
        {"Дюна Частина 2",                  "https://www.youtube.com/watch?v=Way9Dexny3w"},
        {"Дюна",                            "https://www.youtube.com/watch?v=8g18jFHCLXk"},
        {"Евакуація 2",                     "https://www.youtube.com/watch?v=Y274jZs5s7s"},
        {"Еліас",                           "https://www.youtube.com/watch?v=FkPbDuHpVdI"},
        {"Зелена книга",                    "https://www.youtube.com/watch?v=Rta9T9JMoMw"},
        {"Знаки",                           "https://www.youtube.com/watch?v=If3sJsTfXCQ"},
        {"Знахар",                          "https://www.youtube.com/watch?v=tFKnOimSNFY"},
        {"Зникла безвісти",                 "https://www.youtube.com/watch?v=5jFN4r10d6g"},
        {"Золота рибка",                    "https://www.youtube.com/watch?v=F8xSMPr8iRo"},
        {"Зоополіс",                        "https://www.youtube.com/watch?v=jWM0ct-OLsM"},
        {"Зоряні війни",                    "https://www.youtube.com/watch?v=8Qn_spdM5Zg"},
        {"Король талси",                    "https://www.youtube.com/watch?v=mNI0Xse0JrE"},
        {"Крижане серце",                   "https://www.youtube.com/watch?v=bwzLiQZDw2I"},
        {"Кімната вбивств",                 "https://www.youtube.com/watch?v=uFE1yXi3DRg"},
        {"Людина зі сталі",                 "https://www.youtube.com/watch?v=T6DJcgm3wNY"},
        {"Лють",                            "https://www.youtube.com/watch?v=PBtAOKl3d9M"},
        {"Марсіанин",                       "https://www.youtube.com/watch?v=ej3ioOneTy8"},
        {"Матриця",                         "https://www.youtube.com/watch?v=9ix7TUGVYIo"},
        {"Месники",                         "https://www.youtube.com/watch?v=tmeOjFno6Do"},
        {"Мовчання",                        "https://www.youtube.com/watch?v=IClg-OIASNE"},
        {"Монте крісто",                    "https://www.youtube.com/watch?v=qFNkrHHYVck"},
        {"Гарфілд",                         "https://www.youtube.com/watch?v=sGSFWJ9BKRo"},
```

Рис. 3.15 Фрагмент словника YouTube–посилань на трейлери у файлі moviedetailswindow.cpp

Рис. 3.16 Блок-схема алгоритму відтворення трейлера onPlay()

Вікно деталей фільму відкривається поверх головного при кліку на будь–яку картку. У верхній частині розміщено чорний прямокутник із великою іконкою відтворення по центру і підписом «Натисніть «Відтворити» нижче» – це візуальна заглушка замість самого відеоплеєра. Нижче відображається назва фільму великим шрифтом, жанр і рік у підзаголовку. Далі – кнопки дій: «Відтворити», лайк, дизлайк, додати до списку і поділитися посиланням. Нижче – короткий опис сюжету, список акторів і горизонтальний ряд схожих фільмів того самого жанру. В самому низу – блок відгуків глядачів із оцінками у вигляді зірочок.

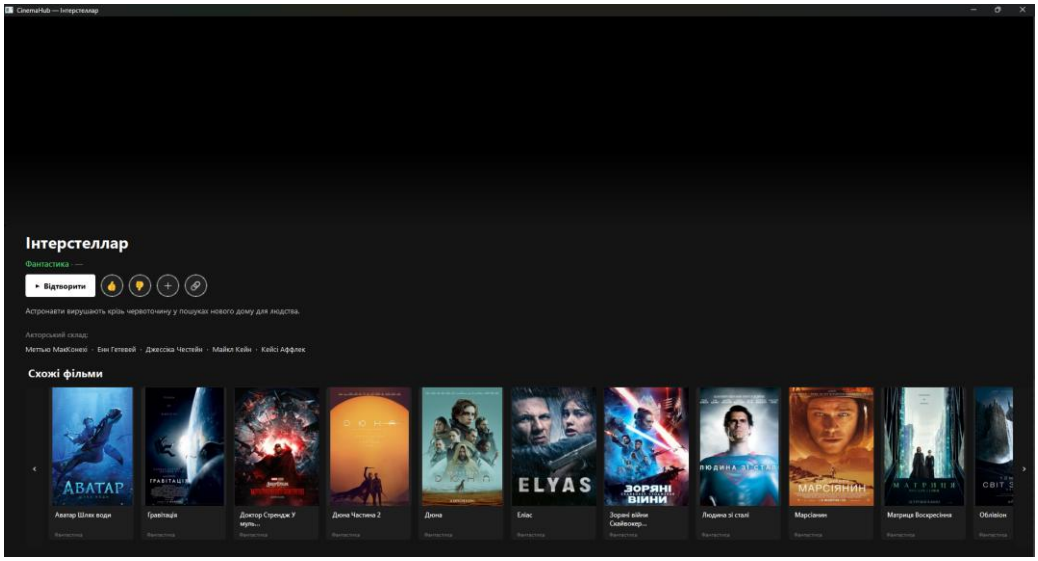


Рис. 3.17 Екранна форма сторінки фільму

На рисунку 3.17 показано вікно детальної інформації про фільм «Інтерстеллар». Зверху – темний блок із кнопкою «Відтворити» і підказкою. Нижче – назва, жанр, рік, кнопки дій і короткий опис. В нижній частині – горизонтальний ряд схожих фільмів і блок відгуків з оцінками.

Такий підхід до відтворення трейлерів – через системний браузер, а не вбудований плеєр – має практичну перевагу: браузер підтримує всі формати відео і субтитри YouTube без додаткових налаштувань. При цьому застосунок залишається відкритим і повністю функціональним, поки користувач дивиться трейлер.

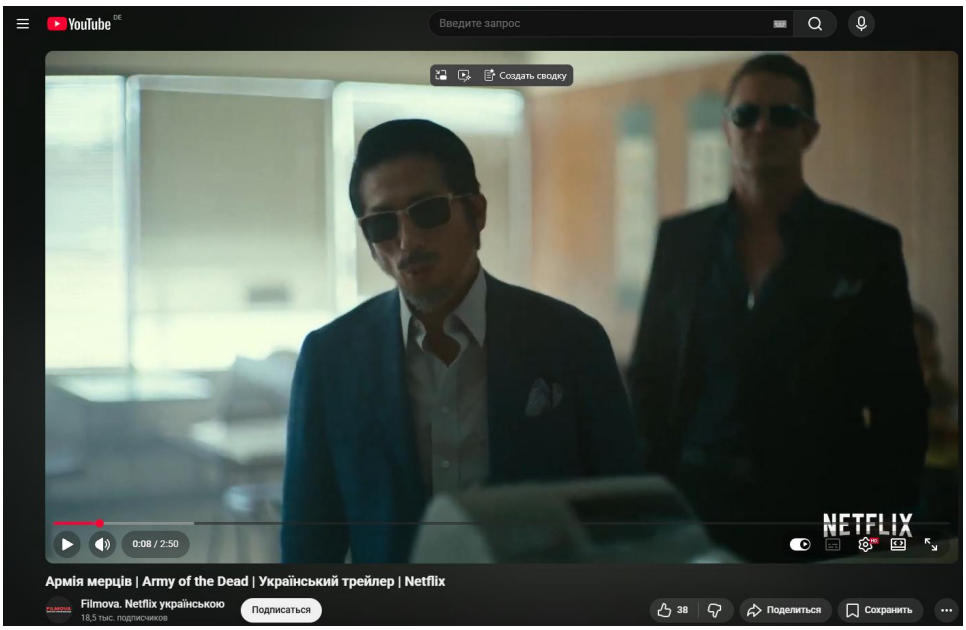


Рис. 3.18 Трейлер фільму відкритий у браузері

На рисунку 3.18 видно результат натискання кнопки «Відтворити» – браузер Microsoft Edge відкрив сторінку YouTube з трейлером відповідного фільму. Застосунок при цьому залишається відкритим у фоні і продовжує нормально працювати.

3.8 Реалізація ініціалізації бази даних SQLite

Для зберігання каталогу фільмів у CinemaHub використовується вбудована СУБД SQLite, яка підключена через модуль Qt6::Sql. Вся логіка першого запуску і налаштування бази даних зосереджена у класі Database, реалізованому у файлах database.h і database.cpp.

Клас Database не містить жодних полів і складається лише зі статичних методів. Це означає, що створювати об'єкт класу не потрібно – всі виклики виконуються напряму через ім'я класу. Точкою входу є метод Database::init(), який викликається у main.cpp самим першим, ще до того як відкривається будь–яке вікно застосунку.

Метод init() виконує три послідовні кроки. Спочатку він встановлює з'єднання з базою даних через Qt–драйвер QSQLITE. Шлях до файлу cinema.db визначається через QStandardPaths::AppDataLocation – це стандартна директорія

для даних застосунків у системі. На Windows це виглядає як AppData\Roaming\CinemaHub\cinema.db. Якщо такої папки ще не існує, метод створює її через QDir::mkpath() перед відкриттям файлу.

Після успішного підключення викликається метод createTables(). Він виконує SQL–команду CREATE TABLE IF NOT EXISTS для таблиці movies з такими полями: id (цілий первинний ключ), title, genre, description, image_path, cast_list (рядкові поля), year і duration (цілі числа), rating (дійсне число). Також створюється таблиця favorites з єдиним полем movie_id для зберігання ідентифікаторів обраних фільмів. Конструкція IF NOT EXISTS гарантує, що при повторному запуску таблиці не будуть перестворені і дані не постраждають.

Третій крок – метод populateIfEmpty(). Він спочатку перевіряє, чи є в таблиці movies хоча б один запис. Якщо таблиця порожня – значить, застосунок запускається вперше – виконується масове наповнення через SQL INSERT. Всі 103 записи вставляються в межах однієї транзакції: на початку виконується BEGIN TRANSACTION, після всіх INSERT – COMMIT. Такий підхід має принципове значення: без транзакції кожен окремий INSERT записується на диск одразу, що для ста записів займає помітний час. З транзакцією всі записи зберігаються одним блоком – перший запуск відбувається миттєво.

Читання даних при кожному наступному запуску застосунку реалізовано у функції loadFromDB() у файлі moviedata.cpp. Ця функція виконує SELECT–запит до таблиці movies, отримує всі рядки і для кожного з них формує структуру Movie, яка потім додається до глобального вектора g_movies. Завдяки прапорцю g_loaded функція виконується лише один раз за сесію незалежно від кількості звернень до даних. Це запобігає зайвим зверненням до бази і прискорює роботу застосунку після першого завантаження.

Практична цінність такої реалізації полягає в тому, що каталог фільмів став незалежним від вихідного коду. Щоб змінити назву фільму, додати новий або оновити опис – достатньо відкрити cinema.db у програмі DB Browser for SQLite, внести потрібні правки і зберегти. При наступному запуску CinemaHub автоматично прочитає оновлені дані без будь–якої перекомпіляції. На рисунку 3.19

наведено вигляд бази даних cinema.db у DB Browser for SQLite – видно таблицю movies з усіма 103 записами та змінену назву фільму для перевірки коректності читання.

Така архітектура зберігання повністю відповідає вимозі куратора, озвученій на передзахисті: дані підвантажуються з SQLite, не зберігаються безпосередньо у вихідному коді і доступні для редагування без перекомпіляції застосунку.

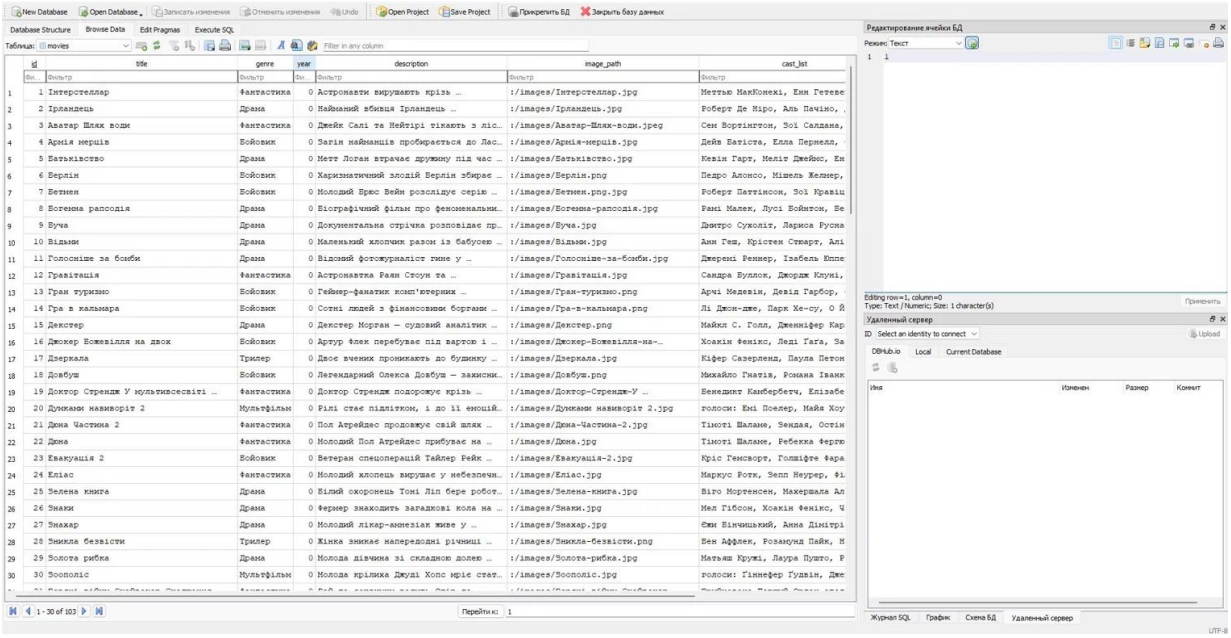


Рис. 3.19 Вигляд бази даних cinema.db у DB Browser for SQLite

3.9 Реалізація перемикання теми інтерфейсу

Тему інтерфейсу реалізовано через методи *toggleTheme()* і *applyTheme()*. Перший інвертує значення прапорця *m_isDark* і передає його до другого. *applyTheme()* оновлює іконку кнопки теми, формує набір кольорових змінних для темного або світлого варіанту і передає їх до рядка QSS–стилів через *setStyleSheet()*. Стиль накладається на головне вікно і успадковується всіма дочірніми елементами. Реалізацію методів наведено на рисунку 3.20. Екранну форму світлої теми наведено на рис. 3.21.

```cpp
void MainWindow::toggleTheme()
{
    m_isDark = !m_isDark;
    applyTheme(m_isDark);
}

void MainWindow::applyTheme(bool dark)
{
    m_isDark = dark;

    if (m_btnTheme) m_btnTheme->setText(dark ? "🌙" : "☀");

    const QString BG   = dark ? "#141414" : "#f5f5f5";
    const QString BG2  = dark ? "#1f1f1f" : "#ffffff";
    const QString TXT  = dark ? "#ffffff" : "#111111";
    const QString SUB  = dark ? "#a0a0a0" : "#555555";
    const QString BRD  = dark ? "#333333" : "#cccccc";
    const QString SBRD = dark ? "#3a3a3a" : "#bbbbbb";

    setStyleSheet(QString(R"(
        QMainWindow,QWidget { background: %1; color: %2; }
        QScrollArea         { background: transparent; border: none; }
        QScrollBar:vertical { background: %3; width: 5px; margin: 0; }
        QScrollBar::handle:vertical { background: %6; border-radius: 2px; min-height: 24px; }
        QScrollBar::add-line:vertical, QScrollBar::sub-line:vertical { height: 0; }
        QScrollBar:horizontal { height: 0; }
        QFrame  { background: %3; border-radius: 6px; }
        QLabel  { background: transparent; color: %2; }
        QPushButton { background: transparent; color: %4; }
        QLineEdit { background: %3; color: %2; border: 1px solid %5;
                    border-radius: 4px; padding: 0 10px; font-size: 13px; }
        QLineEdit:focus { border: 1px solid #E50914; }
    )").arg(BG, TXT, BG2, SUB, BRD, SBRD));
```

Рис. 3.20 Реалізація методів toggleTheme() та applyTheme() у mainwindow.cpp

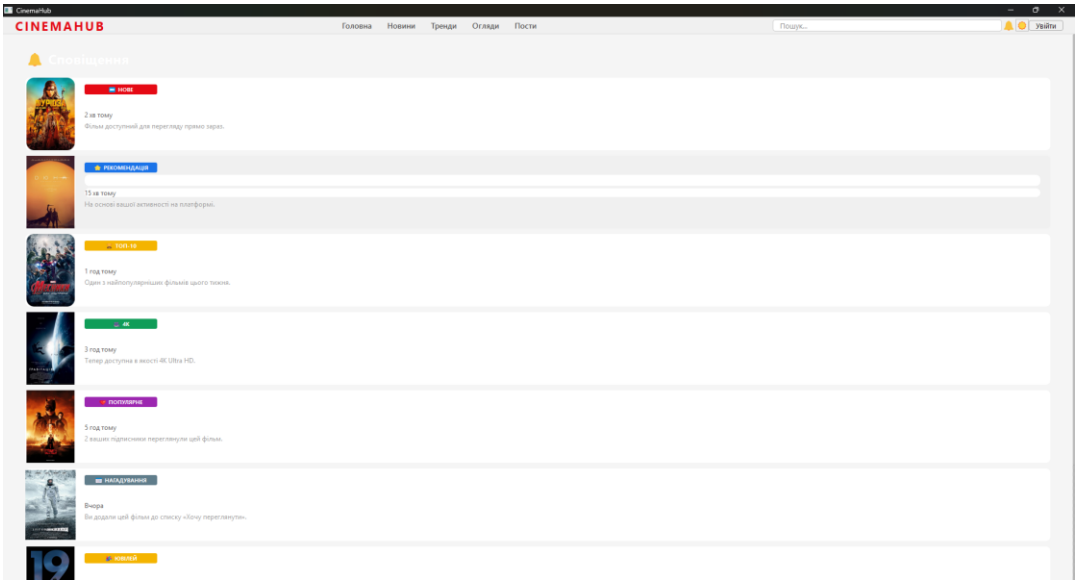


Рис. 3.21 Світла тема інтерфейсу CinemaHub

На рисунку 3.21 показано застосунок у світлій темі. Фон змінився з темно–сірого на світло–сірий, текст – з білого на темний. Навігаційна панель, картки і всі інші елементи автоматично отримали новий стиль через механізм успадкування QSS.

3.10 Реалізація профілю користувача

Профіль користувача реалізовано у класі *ProfileWindow*. Вікно відкривається при натисканні на аватар у навігаційній панелі. Відображається аватар – згенерований із першої літери нікнейму або завантажене фото – а також нікнейм і кількість фільмів у списку обраного.

Завантаження фото реалізовано у методі *onChangeAvatar()*. Відкривається стандартний діалог вибору файлу *QFileDialog::getOpenFileName()* з фільтром для зображень формату PNG, JPG та WEBP. Якщо шлях до файлу не порожній – він передається до *UserManager::instance().setAvatar()* і аватар оновлюється через *refreshAvatar()*. Реалізацію методу наведено на рисунку 3.22.

```
void ProfileWindow::onChangeAvatar()
{
    QString path = QFileDialog::getOpenFileName(
        this,
        "Оберіть аватар",
        QString(),
        "Images (*.png *.jpg *.jpeg *.webp)"
    );
    if (!path.isEmpty()) {
        UserManager::instance().setAvatar(path);
        refreshAvatar();
    }
}
```

Рис. 3.22 Реалізація завантаження фото аватара у profilewindow.cpp

Вихід із акаунту реалізовано у методі *onLogout()*. Викликається *UserManager::instance().logout()* для скидання даних користувача, після чого надсилається сигнал *loggedOut()* і вікно закривається через *accept()*. Реалізацію методу наведено на рисунку 3.23. Екранну форму профілю наведено на рис. 3.31.

```
void ProfileWindow::onLogout()
{
    UserManager::instance().logout();
    emit loggedOut();
    accept();
}
```

Рис. 3.23 Реалізація виходу з акаунту у profilewindow.cpp

3.11 Екранні форми інформаційних розділів

Застосунок містить п'ять інформаційних розділів, доступних через навігаційне меню у верхній частині вікна: «Новини», «Тренди», «Огляди», «Пости» і «Сповіщення». Кожен з них будується при першому відкритті і зберігається у стеку сторінок до завершення сесії.

Розділ «Новини» відображає стрічку кіноновин у вигляді вертикального списку. Кожен запис містить мініатюру постера зліва, кольорову мітку–категорію («Прем'єра», «Новинка», «Топ тижня», «Рекомендовано», «Для сім'ї» «Класика»), назву новини, дату і короткий анонс. Кольори міток відрізняються залежно від категорії – зелений для прем'єр, червоний для новинок, жовтий для топу тижня. Це дозволяє швидко орієнтуватись у стрічці без читання кожного запису. Екранну форму розділу наведено на рисунку 3.24.

Усі п'ять інформаційних розділів реалізовані як окремі сторінки у QStackedWidget головного вікна. Перехід між ними відбувається миттєво – без перезавантаження і без затримки, оскільки кожна сторінка будується один раз і зберігається у пам'яті до завершення сесії.

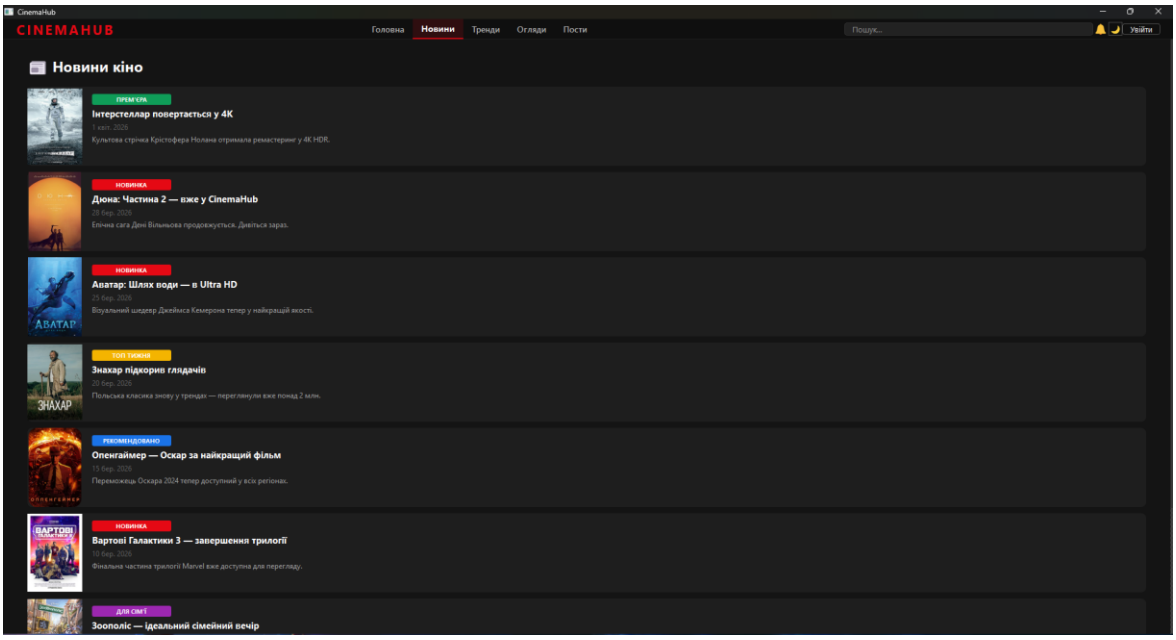


Рис 3.24. Розділ «Новини кіно»

Розділ «Тренди» поділено на два блоки. Перший – «Топ–10 сьогодні в Україні» – відображає десять найпопулярніших фільмів у вигляді великих пронумерованих карток із горизонтальним прокручуванням. На кожній картці видно постер, порядковий номер великим шрифтом, назву, жанр, а також інформаційну смугу з рейтингом, тривалістю і роком. Другий блок – «За жанрами» – містить окремі горизонтальні ряди для кожного жанру зі стрілками прокручування. Екранну форму розділу наведено на рисунку 3.25.

Розділ «Новини» відображає картки з актуальними матеріалами про кіноіндустрію. Кожна картка містить зображення, заголовок і короткий анонс. Розділ «Тренди» показує фільми, відсортовані за популярністю, у вигляді нумерованого списку карток із рейтингом та жанром. Розділ «Огляди та рецензії» представляє фільми з розгорнутими текстовими коментарями критиків і загальною оцінкою. Усі три розділи підтримують перемикання між темною і світлою темами та дозволяють відкрити детальну сторінку будь–якого фільму одним кліком.

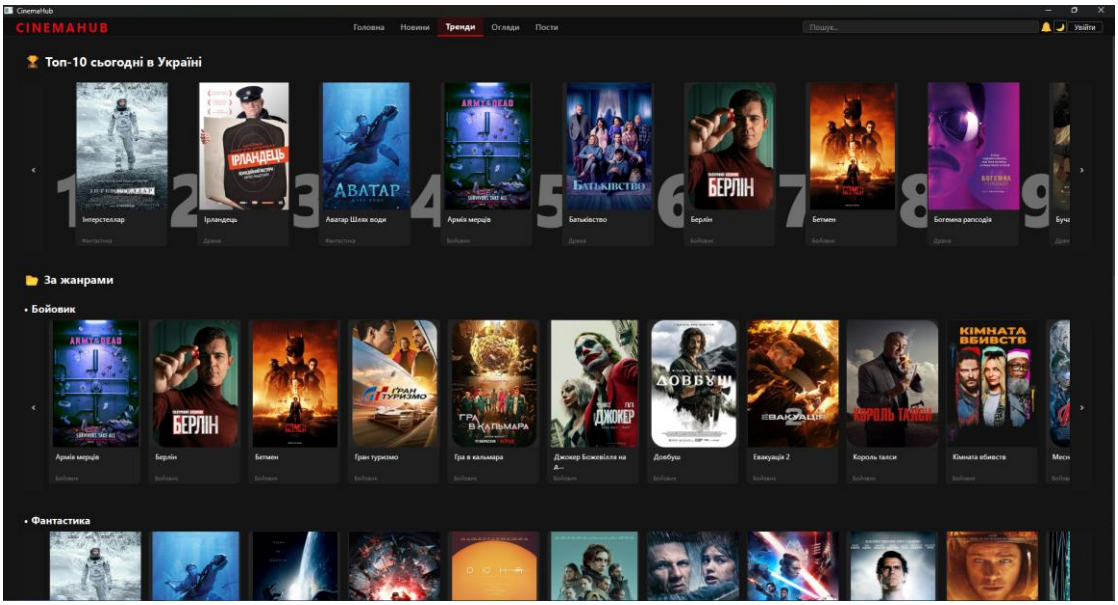


Рис. 3.25 Розділ «Тренди»

Розділ «Огляди» відображає список рецензій глядачів на фільми з каталогу. Кожен запис містить назву фільму жирним шрифтом, текст відгуку, ім'я автора і дату публікації. Праворуч відображається числова оцінка і ряд зірочок. Відгуки відсортовано за датою – найновіші зверху. Розділ дає змогу ознайомитись із

враженнями інших глядачів перед переглядом. Екранну форму розділу наведено на рисунку 3.26.

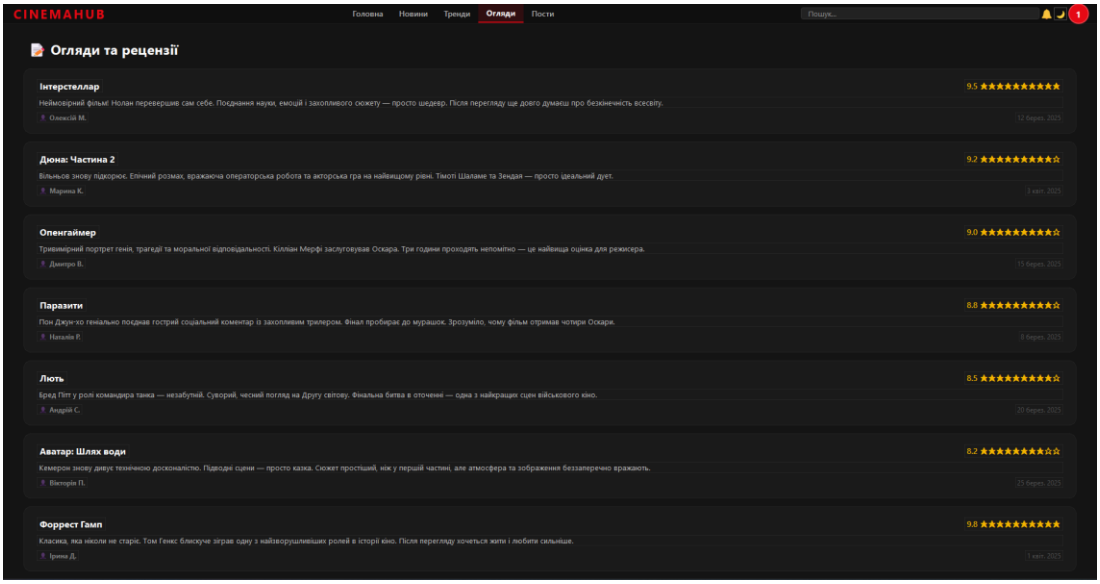


Рис. 3.26 Розділ «Огляди та рецензії»

Розділ «Пости» відображає весь каталог фільмів у вигляді рівномірної сітки карток. Кожна картка містить постер фільму, назву, скорочений опис сюжету і кнопку «Дивитися» з червоним акцентом у нижній частині. При наведенні курсору картка підсвічується червоною рамкою і тінню. Такий формат дозволяє переглянути весь каталог одним поглядом і швидко перейти до потрібного фільму. Екранну форму розділу наведено на рисунках 3.27 і 3.28.

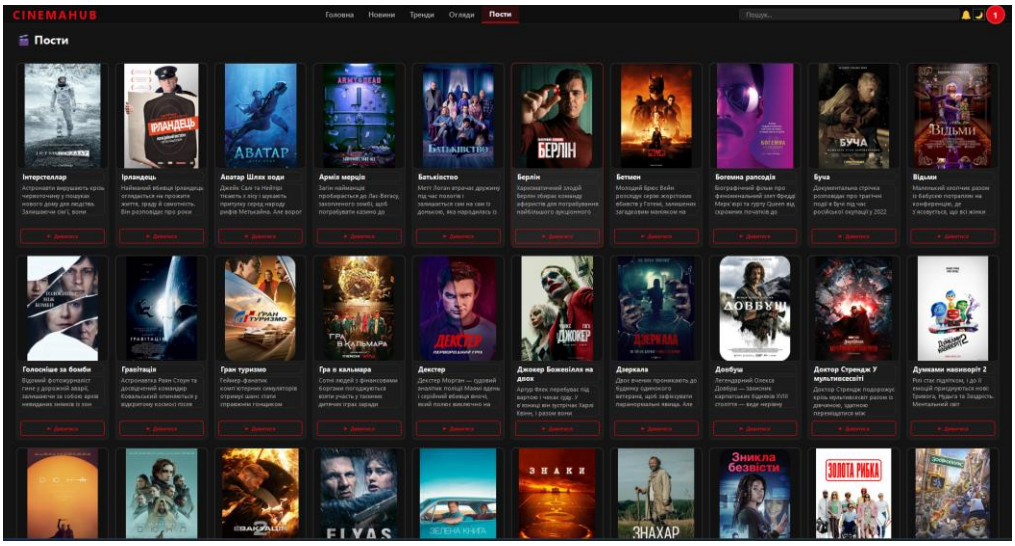


Рис. 3.27 Розділ «Пости» у темній темі

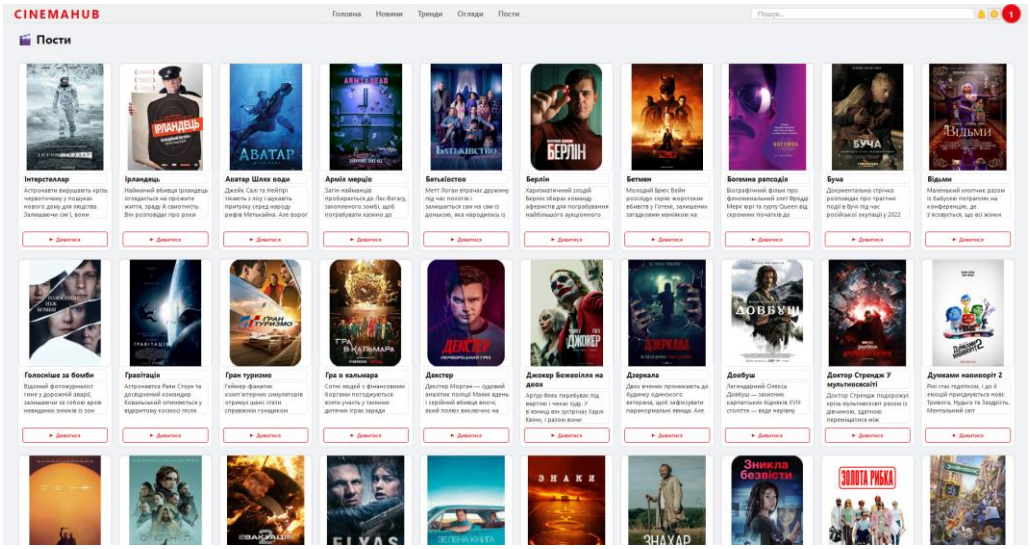


Рис. 3.28 Розділ «Пости» у світлій темі

На рисунку 3.29 наведено збільшений вигляд окремої картки фільму з розділу «Пости». Видно постер, назву «Батьківство», короткий текст опису і кнопку «Дивитися».

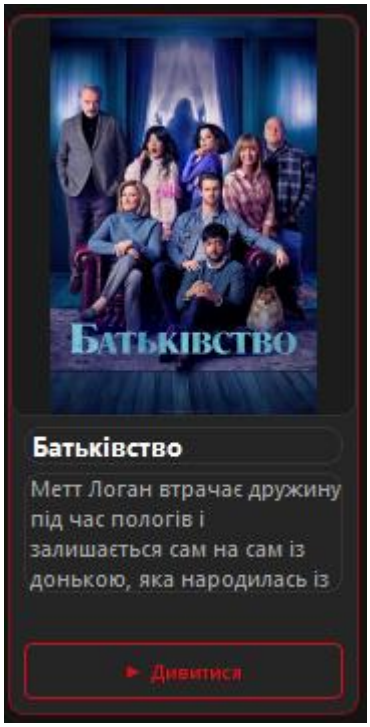


Рис. 3.29 Окрема картка фільму в розділі «Пости»

Розділ «Сповіщення» показує персоналізовані повідомлення для користувача. Кожне сповіщення має мініатюру постера, кольорову мітку–категорію, заголовок, час і короткий опис. Підтримуються такі категорії: «Нове»

(червона), «Рекомендація» (блакитна), «Топ–10» (жовта), «4K» (зелена), «Популярне» (фіолетова), «Нагадування» (сіра), «Ювілей» (помаранчева). Завдяки кольоровим міткам користувач одразу розуміє характер повідомлення. Екранну форму розділу наведено на рисунку 3.30.

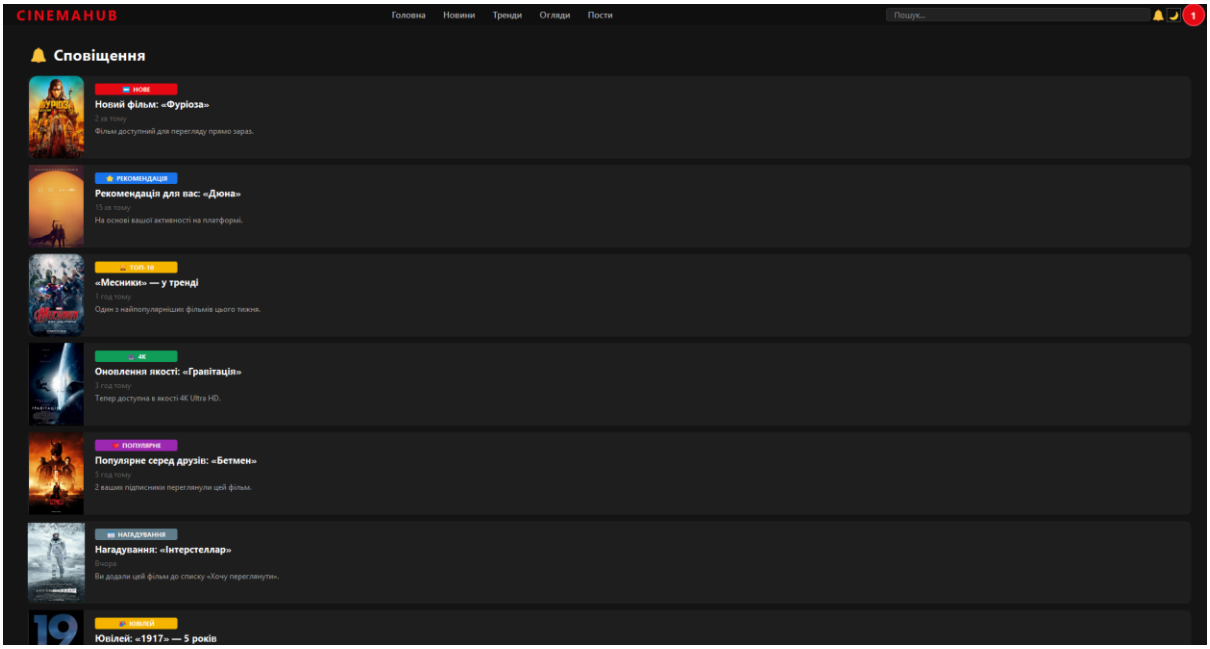


Рис. 3.30 Розділ «Сповіщення»

Вікно профілю відкривається в окремому невеликому вікні при натисканні на аватар у правому верхньому куті навігаційної панелі. У верхній частині розміщено червоний градієнтний банер і круглий аватар із першою літерою нікнейму – або завантаженим фото, якщо користувач його додав. Під аватаром відображається нікнейм і лічильник обраних фільмів із іконкою серця. Далі йде палітра з шістьма кольорами для зміни фону аватара, кнопка «Завантажити фото», поле для редагування імені або нікнейму та дві кнопки внизу – «Зберегти» і «Вийти». Екранну форму профілю наведено на рисунку 3.31.

Усі п'ять інформаційних розділів реалізовані як окремі сторінки у QStackedWidget головного вікна. Перехід між ними відбувається миттєво – без перезавантаження і без затримки, оскільки кожна сторінка будується один раз і зберігається у пам'яті до завершення сесії.

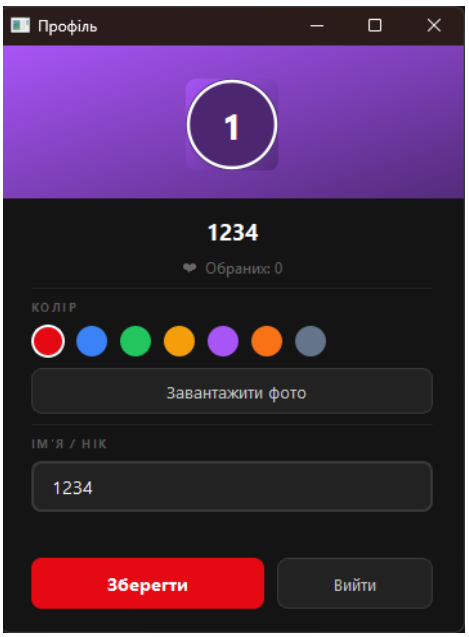


Рис. 3.31 Вікно профілю користувача

На рисунку 3.31 показано вікно профілю користувача. Зверху відображається круглий аватар із першою літерою нікнейму на червоному тлі. Під ним – нікнейм «1234» і лічильник обраних фільмів. Далі – палітра з шістьма кольорами, кнопка завантаження фото, поле для зміни нікнейму і дві кнопки.

Кнопка «Зберегти» застосовує зміни нікнейму і зберігає вибраний колір аватара через клас UserManager. Якщо користувач завантажив власне фото, воно відображається замість кольорового кола з літерою. Кнопка «Вийти» завершує поточну сесію і повертає користувача до вікна авторизації. Всі зміни профілю зберігаються локально через QSettings і автоматично застосовуються при наступному запуску застосунку.

4. ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Вибір методів тестування

Після завершення програмної реалізації CinemaHub проведено тестування з метою перевірки коректності роботи всіх функціональних модулів. Для цього обрано два методи – функціональне і модульне тестування.

Функціональне тестування проводиться з позиції кінцевого користувача: перевіряється, чи виконує застосунок саме те, що від нього очікується в конкретному сценарії. Основою для складання тест–кейсів стали функціональні вимоги з підрозділу 2.2 – кожна вимога перетворилась у один або кілька тест–кейсів з конкретними вхідними даними і очікуваним результатом.

Модульне тестування спрямоване на перевірку окремих функцій в ізоляції від решти системи. Цей метод дозволяє переконатись, що логіка конкретної функції правильна незалежно від того, яким чином вона буде викликана з інтерфейсу. Для CinemaHub модульному тестуванню піддано три функції, які є найбільш критичними з точки зору коректності: *searchMovies(), trailerUrl()* і *onLogin()*.

Обидва методи разом дають достатньо повне покриття: функціональне тестування перевіряє систему як єдине ціле, а модульне – окремі алгоритми. Тестування проводилось вручну на тестовому акаунті.

Вибір ручного тестування пояснюється специфікою проєкту. Значна частина функціоналу CinemaHub є інтерфейсним: правильність роботи перевіряється візуально – чи з'явився список підказок, чи змінилась тема, чи відкрилось потрібне вікно. Для таких перевірок автоматизовані тести потребують складної інфраструктури із симуляцією подій миші і клавіатури, яка збільшила б обсяг коду тестів в рази без практичної переваги для застосунку такого масштабу.

Для трьох функцій із чітко визначеним інтерфейсом вхід–вихід – *searchMovies(), trailerUrl()* і *onLogin()* – проведено повноцінне модульне тестування: вони не залежать від графічного стану і перевіряються в ізоляції від решти системи. Така комбінація ручного функціонального тестування для UI і

ізольованого модульного для бізнес–логіки є стандартним підходом для проєктів подібного класу.

4.2 Функціональне тестування

Функціональне тестування проводилось відповідно до вимог, сформульованих у підрозділі 2.2. Результати наведено в таблиці 4.1.

Таблиця 4.1

Результати функціонального тестування

№	Функція	Вхідні дані	Очікуваний результат	Статус
ТК–01	Вхід із коректними даними	Логін: admin, Пароль: 1234	Відкривається головне вікно	Пройдено
ТК–02	Вхід з некоректними даними	Логін: test, Пароль: 0000	Повідомлення про помилку	Пройдено
ТК–03	Реєстрація нового користувача	Логін: user1, Пароль: 5678	Успішна реєстрація	Пройдено
ТК–04	Паролі не збігаються	Пароль: 1234, Підтвердження: 4321	Повідомлення про невідповідність	Пройдено

Продовження таблиці 4.1

№	Функція	Вхідні дані	Очікуваний результат	Статус
ТК–05	Пошук за повною назвою	Запит: «Інтерстеллар»	Відображається 1 результат	Пройдено

ТК–06	Пошук за частиною назви	Запит: «ав»	Фільми зі збігом у назві	Пройдено
ТК–07	Пошук без результатів	Запит: «ааааа»	Повідомлення «Нічого не знайдено»	Пройдено
ТК–08	Автодоповнення	Введено «бу»	Список підказок	Пройдено
ТК–09	Перегляд деталей фільму	Клік на картку «Довбуш»	Відкривається вікно з інформацією	Пройдено
ТК–10	Відкриття трейлера	Натискання «Відтворити»	Браузер відкриває YouTube	Пройдено
ТК–11	Додавання до обраного	Клік на «+» у деталях фільму	Фільм додається до списку	Пройдено
ТК–12	Перемикання на світлу тему	Натискання	Змінює кольорову схему	Пройдено
ТК–13	Зміна кольору аватара	Клік на колір	Колір аватара	Пройдено
ТК–14	Вихід із акаунту	Натискання «Вийти» у профілі	Відкривається вікно авторизації	Пройдено
ТК–15	Навігація між розділами	Натискання всіх пунктів меню	Кожен розділ відображається коректно	Пройдено

Усі п'ятнадцять тест–кейсів виконано успішно. Відхилень від очікуваної поведінки не виявлено.

ТК–01 – ТК–04 перевіряли систему авторизації. ТК–01 підтвердив основний сценарій входу: при правильних даних застосунок закриває вікно авторизації і

відкриває головне вікно без затримок. ТК–02 перевірив протилежний сценарій – некоректні дані не дають доступу до головного вікна, а виводять повідомлення про помилку і очищують поле пароля. При цьому жодного аварійного завершення або виключення не відбувається – застосунок залишається стабільним. ТК–03 перевірив повний сценарій реєстрації нового акаунту, включно із збереженням через *UserManager*. ТК–04 перевірив граничний випадок – коли два поля пароля не збігаються. Застосунок виводить повідомлення про невідповідність і не зберігає дані, що є очікуваною поведінкою.

Окремо від таблиці перевірявся сценарій порожніх полів при вході: якщо натиснути «Увійти» без введення будь–яких даних – застосунок не звертається до *UserManager*, а одразу виводить повідомлення про обов'язкові поля. Це важливо, бо звернення з порожнім логіном могло б призвести до некоректного стану.

ТК–05 – ТК–08 перевіряли пошук і автодоповнення. ТК–05 підтвердив пошук за повною назвою – при введенні «Інтерстеллар» результат відображається миттєво. ТК–06 перевірив пошук за частиною назви: введення «ав» повернуло кілька фільмів зі збігом. Пошук фільтрує за входженням, а не лише за початком рядка – це важливо для зручності. ТК–07 перевірив поведінку при запиті без збігів: застосунок виводить повідомлення «Нічого не знайдено» замість порожньої сторінки – користувач розуміє, що пошук спрацював, просто результатів немає. ТК–08 підтвердив роботу автодоповнення: підказки з'являються вже з першого введеного символу і оновлюються без натискання Enter. Додатково перевірявся пошук у верхньому регістрі: запит «АВАТАР» повернув той самий результат, що й «аватар», що підтверджує коректну роботу *Qt::CaseInsensitive*.

ТК–09 – ТК–11 перевіряли перегляд фільму і роботу з обраним. Відкриття сторінки деталей (ТК–09) перевірялось для карток у горизонтальних рядах, у розділі «Пости» і в результатах пошуку – в усіх трьох випадках вікно відкривалось коректно і відображало правильні дані. ТК–10 перевіряв відтворення трейлера для фільмів зі словника і без: у першому випадку браузер відкривав YouTube, у другому запускався резервний VideoPlayer. Застосунок при цьому залишався відкритим і доступним. ТК–11 перевіряв додавання і видалення з обраного: кнопка серця

коректно змінює стан і зберігає його до кінця сесії, лічильник у профілі оновлюється відповідно.

ТК–12 – ТК–15 перевіряли налаштування і навігацію. Перемикання теми (ТК–12) перевірялось у кожному із шести розділів застосунку. У всіх випадках тема змінювалась миттєво і повністю: фон, текст, картки і навігаційна панель одночасно отримували новий стиль без залишкових елементів попереднього. Зміна кольору аватара (ТК–13) після натискання «Зберегти» одразу відображалась у навігаційній панелі без перезапуску. Вихід з акаунту (ТК–14) коректно повертав до вікна авторизації і скидав дані поточного користувача. Навігація між усіма розділами (ТК–15) перевірялась як у послідовному режимі, так і в довільному порядку – QStackedWidget у всіх випадках відображав коректний розділ.

Під час проведення функціонального тестування окремо перевірялась робота застосунку при нестандартних вхідних даних. Наприклад, при спробі входу з порожніми полями система коректно виводить повідомлення про помилку і не допускає переходу на головний екран. При пошуку з введенням спеціальних символів застосунок не завершується з помилкою, а просто повертає порожній результат. Це підтверджує, що базові сценарії захисту від некоректних даних реалізовано правильно.

Окремо перевірялась коректність роботи перемикання теми: після зміни теми всі дочірні елементи інтерфейсу – картки, навігаційна панель, поле пошуку – оновлювались одночасно без затримки. Навігація між розділами після зміни теми зберігала поточний стан – якщо користувач був на сторінці «Тренди» і перемкнув тему, він залишився на тій самій сторінці.

Також перевірялась поведінка застосунку при послідовному виконанні кількох дій: наприклад, додавання фільму до обраного, потім перегляд профілю де відображається лічильник, потім вихід і повторний вхід. Лічильник скидається при виході – це очікувана поведінка, оскільки список обраного зберігається в пам'яті і не персистується між сесіями. Загалом поведінка застосунку у всіх перевірених сценаріях відповідала очікуванням.

Окремо перевірялись граничні сценарії, які не відображені в таблиці, але важливі для стабільності застосунку. При введенні в поле пошуку рядка з кирилицею у верхньому регістрі – наприклад «АВАТАР» – пошук повернув правильний результат «Аватар: Шлях води», що підтвердило коректну роботу Qt::CaseInsensitive. При введенні порожнього рядка після попереднього запиту – застосунок повернув користувача на головну сторінку без жодних помилок.

Перевірялась також поведінка при швидкому перемиканні між розділами: якщо натискати кнопки навігації дуже швидко, підряд кілька разів, застосунок не зависає і не показує некоректний стан. Це пов'язано з тим, що механізм навігації просто перемикає поточний індекс без перебудови сторінок.

Тестування відтворення трейлерів показало, що при відсутності підключення до інтернету і натисканні «Відтворити» браузер відкривається, але YouTube показує повідомлення про відсутність з'єднання.

Така поведінка є очікуваною і прийнятною – застосунок не може контролювати стан мережевого підключення браузера. Для фільмів без YouTube–посилання у словнику в цьому сценарії запускається резервний VideoPlayer, який відтворює локальний відеофайл без потреби в інтернеті. Таким чином застосунок залишається частково функціональним навіть без мережі.

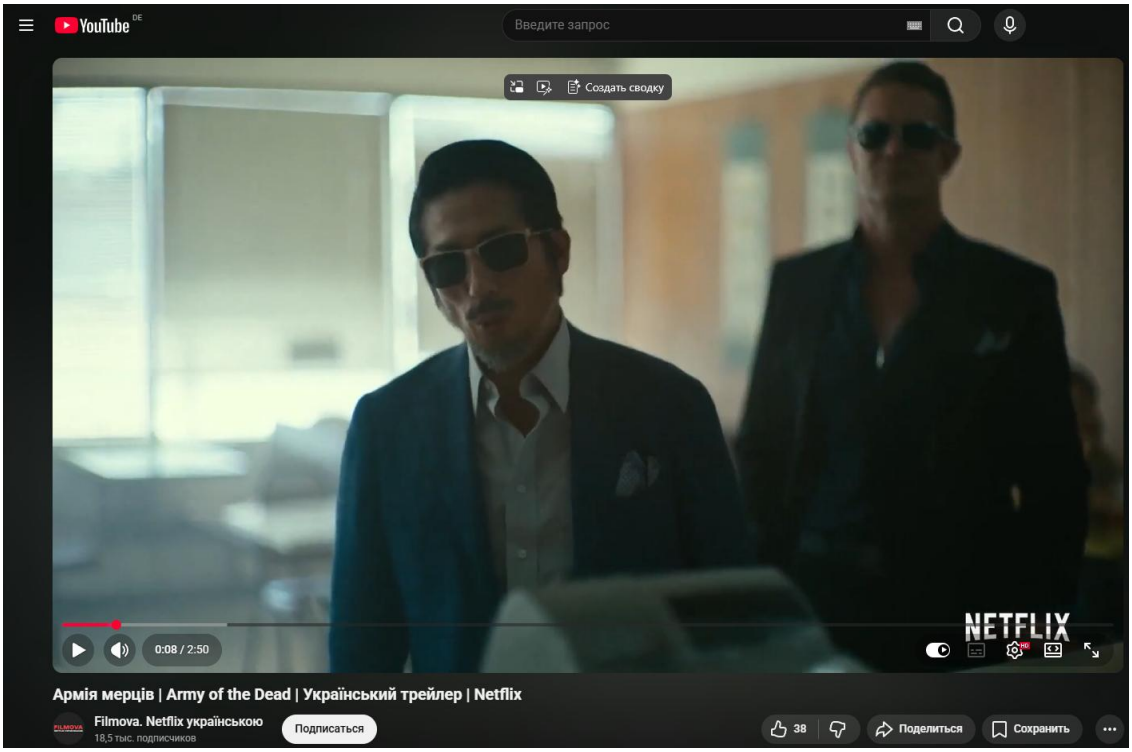


Рис. 4.1 Результат тест–кейсу ТК–10: трейлер відкритий у браузері

4.3 Модульне тестування

Модульне тестування проводилось для трьох ключових функцій застосунку. Результати наведено в таблиці 4.2.

Таблиця 4.2

Результати модульного тестування

№	Функція	Тестовий випадок	Очікуваний результат	Статус
МТ–01	*searchMovies()*	Запит «інтер»	Фільми з «інтер» у назві	Пройдено
МТ–02	*searchMovies()*	Запит «ІНТЕР»	Той самий результат що й «інтер»	Пройдено

Продовження таблиці 4.2

№	Функція	Тестовий випадок	Очікуваний результат	Статус
МТ–03	searchMovies()	Порожній рядок	Порожній список	Пройдено
МТ–04	searchMovies()	Запит «ааааа»	Порожній список	Пройдено
МТ–05	trailerUrl()	Назва «Інтерстеллар»	Посилання на YouTube	Пройдено
МТ–06	trailerUrl()	Назва без запису словнику	Порожній рядок	Пройдено
МТ–07	onLogin()	Коректні логін і пароль	Виклик accept()	Пройдено
МТ–08	onLogin()	Некоректний пароль	Повідомлення про помилку	Пройдено
МТ–09	onLogin()	Порожні поля	Повідомлення про помилку	Пройдено

Функція пошуку стабільно працює з будь–яким рядком – порожнім, у верхньому регістрі або таким, що не має збігів. Це важливо, бо пошук запускається при кожній зміні тексту в полі і помилка тут одразу б проявилась у інтерфейсі. Функція обробки трейлерів коректно розрізняє два сценарії: коли посилання знайдено у словнику – відкривається браузер, коли ні – запускається резервний програвач. Функція авторизації правильно розрізняє три сценарії: коректні дані, некоректні дані і порожні поля.

Функція пошуку є найбільш критичною за частотою виклику – вона запускається при кожному введеному символі, потенційно десятки разів за секунду. Перші два тести підтвердили коректну роботу при різних регістрах: однаковий результат при введенні «інтер» і «ІНТЕР». Наступні два перевірили граничні випадки – порожній рядок і рядок без збігів. В обох випадках функція повертає порожній список без жодних помилок.

Варто уточнити, що при порожньому запиті функція повертає порожній список. Це навмисне рішення: алгоритм пошуку перевіряє довжину рядка і при порожньому значенні перемикає інтерфейс назад на головну сторінку, не запускаючи пошук взагалі. Тест МТ–03 підтвердив коректність такої поведінки. Тест МТ–03 підтвердив, що навіть якщо функція отримає порожній рядок – результат буде коректним.

Функція обробки трейлерів є словником із майже ста записів – парами «назва фільму – YouTube–посилання». МТ–05 підтвердив, що для відомого фільму повертається правильне посилання. МТ–06 перевірив поведінку для фільму без запису у словнику: функція повертає порожній результат, що є сигналом для запуску резервного відеоплеєра замість браузера. Кнопка «Відтворити» завжди щось робить – ніколи не залишається мовчки неактивною.

Функція авторизації перевірялась у трьох сценаріях. МТ–07 – коректні дані, вікно авторизації закривається і відкривається головний екран. МТ–08 – некоректний пароль: виводиться повідомлення про помилку, поле пароля очищується. МТ–09 – порожні поля: перевірка виконується до звернення до бази користувачів, тому одразу виводиться повідомлення про обов'язкові поля.

Загалом результати тестування підтвердили, що реалізований застосунок виконує всі заявлені функції відповідно до вимог, визначених у розділі 2. Відхилень від очікуваної поведінки не виявлено.

Варто зазначити, що модульне тестування для цього проєкту проводилось без використання спеціалізованих фреймворків – наприклад, Qt Test або Google Test. Це пояснюється тим, що функції які перевірялись мають простий і передбачуваний інтерфейс: подати рядок на вхід і перевірити що вийшло. Для складніших систем з великою кількістю залежностей автоматизоване тестування було б доцільнішим, але для поточного обсягу проєкту ручна перевірка дала достатній рівень впевненості.

4.4 Тестування сумісності інтерфейсу

Окрім функціонального і модульного тестування, проведено перевірку коректності відображення інтерфейсу в різних умовах. Перевірялась поведінка застосунку при різних роздільних здатностях екрану.

На екрані з роздільною здатністю 1920×1080 пікселів застосунок відображається коректно: навігаційна панель не переповнюється, горизонтальні ряди карток мають достатньо місця, текст читається без проблем.

На роздільній здатності 1280×720 – мінімальній заявленій – застосунок також функціонує коректно. Деякі горизонтальні ряди показують менше карток одночасно, але прокручування стрілками залишається доступним.

Перевірялась поведінка при перемиканні між темною і світлою темами у різних розділах. У всіх перевірених сценаріях тема змінювалась миттєво і коректно – без залишкових елементів старого стилю.

Окремо перевірялась робота обох тем із картками фільмів. Клас *MovieCard* підтримує метод *setDarkTheme()*, що окремо оновлює кольори тексту і фону картки. Перевірка показала, що після перемикання теми картки у всіх розділах – Пости, Тренди, результати пошуку – також змінюють оформлення.

Результати перевірки сумісності підтвердили, що застосунок стабільно працює в усіх перевірених умовах і не має критичних візуальних помилок.

Під час тестування сумісності окремо перевірялась поведінка застосунку при зміні масштабу системи. На Windows із масштабуванням інтерфейсу 125% і 150% застосунок відображається коректно: елементи збільшуються пропорційно, текст залишається читабельним, постери масштабуються без артефактів. Це забезпечується тим, що Qt6 за замовчуванням підтримує High–DPI режим і автоматично враховує системний масштаб.

Тестування банерного каруселя проводилось на двох роздільних здатностях. На 1920×1080 банер займає повну ширину вікна. На 1280×720 він також коректно масштабується, оскільки QLabel із банером налаштований на режим scaledContents. Заголовок і опис залишаються читабельними на обох варіантах.

Перевірялась також поведінка горизонтальних рядів карток при різній ширині вікна. При розширенні вікна більше 1920 пікселів ряди не ламаються – ширина карток фіксована, додатковий простір залишається після рядів. При звуженні до 1280 пікселів кількість видимих карток зменшується, але кнопки прокручування залишаються доступними і коректно працюють.

Перевірка *QScrollArea* у MovieDetailsWindow показала коректну роботу вертикального прокручування: при відкритті сторінки фільму вміст відображається з верхньої позиції, прокручування плавне, горизонтальна смуга відсутня.

Окремо тестувалась поведінка при швидкому відкритті і закритті MovieDetailsWindow кілька разів підряд. Нових вікон не накопичувалось і пам'ять не витікала – завдяки атрибуту *WA_DeleteOnClose*, який знищує об'єкт при закритті вікна.

Загальні результати тестування підтвердили, що застосунок CinemaHub відповідає всім сформульованим вимогам. П'ятнадцять функціональних тест–кейсів пройдено успішно, дев'ять модульних тестів для трьох ключових функцій також без відхилень, тестування сумісності показало коректну роботу на різних роздільних здатностях і масштабах інтерфейсу.

Така поведінка є очікуваною і прийнятною – застосунок не може контролювати стан мережевого підключення браузера. Для фільмів без YouTube–посилання у словнику в цьому сценарії запускається резервний VideoPlayer, який відтворює локальний відеофайл без потреби в інтернеті. Таким чином застосунок залишається частково функціональним навіть без мережі.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розроблено десктопний застосунок CinemaHub для керування каталогом фільмів на основі мови C++ і фреймворку Qt версії 6.

У ході виконання роботи вирішено всі поставлені задачі.

Проведено аналіз існуючих програмних рішень – IMDb, Letterboxd, Kodi та MediaMonkey. Встановлено, що жодне з них не поєднує офлайн–режим, сучасний інтерфейс та орієнтацію саме на керування персональною кінотекою. На основі аналізу визначено вимоги до розроблюваного застосунку.

Досліджено можливості мови C++ та фреймворку Qt 6 для побудови крос–платформних десктопних застосунків. Обґрунтовано вибір C++17 за продуктивність та контроль над ресурсами, Qt 6 – за вбудовану підтримку мультимедіа і гнучку систему стилізації, CMake – як стандартний інструмент збірки.

Спроєктовано архітектуру застосунку з тринадцяти класів та підключено локальну базу даних SQLite. Побудовано діаграму варіантів використання і сформульовано функціональні вимоги. Розроблено структуру даних і концепцію інтерфейсу з темною і світлою темами.

Реалізовано анімовану заставку при запуску застосунку та повноцінну систему авторизації і реєстрації користувача з перевіркою введених даних і захистом від некоректного введення.

Реалізовано основний функціонал застосунку: перегляд каталогу у вигляді карток з постерами, горизонтальне прокручування по рядках добірок, пошук з автодоповненням, відображення детальної інформації про фільм, відкриття трейлерів через системний браузер, управління списком обраного, перегляд новин, трендів, оглядів та сповіщень. Реалізовано перемикання між темною і світлою темами без перезапуску застосунку та профіль користувача з можливістю зміни аватара і нікнейму.

Проведено тестування застосунку. П'ятнадцять функціональних тест–кейсів охопили всі основні сценарії роботи. Дев'ять модульних тестів перевірили три

ключові функції – пошуку, обробки трейлерів та авторизації – в ізоляції від графічного інтерфейсу. Усі тести пройдено успішно, відхилень не виявлено.

Практична цінність роботи полягає в тому, що розроблений застосунок закриває реальну потребу – дає змогу вести особисту кінотеку в офлайн–режимі з інтерфейсом, що відповідає сучасним стандартам. Застосунок реалізовано без сторонніх бібліотек, що спрощує його розповсюдження.

Перспективи подальшого розвитку включають інтеграцію з онлайн–базою даних фільмів, хмарну синхронізацію профілю між пристроями і підтримку кількох облікових записів.

Розроблений застосунок демонструє, що сучасний десктопний продукт рівня комерційних стрімінгових сервісів можна реалізувати засобами відкритих технологій без залежності від хмарної інфраструктури. Поєднання C++17, Qt 6 і SQLite забезпечило повну офлайн–функціональність, швидку роботу з медіаданими і зручне розповсюдження готового застосунку. Результати роботи підтверджують обґрунтованість обраного підходу і можуть бути використані як основа для подальшого розширення функціоналу.

Розроблений застосунок демонструє, що сучасний десктопний продукт рівня комерційних стрімінгових сервісів можна реалізувати засобами відкритих технологій без залежності від хмарної інфраструктури. Поєднання C++17, Qt 6 і SQLite забезпечило повну офлайн–функціональність, швидку роботу з медіаданими і зручне розповсюдження готового застосунку. Результати роботи підтверджують обґрунтованість обраного підходу і можуть бути використані як основа для подальшого розширення функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CMake Documentation and Community. CMake – Upgrade Your Software Build System. URL: https://cmake.org/documentation/ (дата звернення: 28.04.2026).
2. The C++ framework Qt. ionos Digital Guide. URL: https://www.ionos.com/digitalguide/server/know–how/qt/ (дата звернення: 28.04.2026).
3. Qt 6.11. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/ (дата звернення: 28.04.2026).
4. Multimedia Overview | Qt Multimedia | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/multimediaoverview.html (дата звернення: 28.04.2026).
5. Qt Widgets | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/qtwidgets–index.html (дата звернення: 28.04.2026).
6. Signals & Slots | Qt Core | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/signalsandslots.html (дата звернення: 28.04.2026).
7. The Qt Resource System | Qt Core | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/resources.html (дата звернення: 28.04.2026).
8. QMediaPlayer Class | Qt Multimedia | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/qmediaplayer.html (дата звернення: 28.04.2026).
9. Qt projects | CLion. CLion Help. URL: https://www.jetbrains.com/help/clion/qt–tutorial.html (дата звернення: 28.04.2026).
10. All C++ APIs per Module | Qt 6.11. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/modules.html (дата звернення: 28.04.2026).
11. Qt projects | CLion. CLion Help. URL: https://www.jetbrains.com/help/clion/qt–tutorial.html#edit–ui (дата звернення: 28.04.2026).
12. Qt Examples And Tutorials | Qt 6.11. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/qtexamplesandtutorials.html (дата звернення: 28.04.2026).
13. An Image Viewer | The Qt 6 Book. Qt | Tools for Each Stage of Software Development Lifecycle. URL: https://www.qt.io/product/qt6/qml–book/ch06–controls–image–viewer (дата звернення: 28.04.2026).

14. Images | Qt Design Studio Documentation 4.8.1. Qt Documentation | Home. URL: https://doc.qt.io/qtdesignstudio/quick–images.html (дата звернення: 28.04.2026).
15. Image QML Type | Qt Quick | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/qml–qtquick–image.html (дата звернення: 28.04.2026).
16. Introduction to Qt Quick for C++ Developers – Qt Wiki. Qt Wiki. URL: https://wiki.qt.io/Introduction_to_Qt_Quick_for_C++_Developers (дата звернення: 28.04.2026).
17. Login UI: Components | Qt Design Studio Documentation 4.8.1. Qt Documentation | Home. URL: https://doc.qt.io/qtdesignstudio/qtdesignstudio–loginui1–example.html (дата звернення: 28.04.2026).
18. QMainWindow Class | Qt Widgets | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/qmainwindow.html (дата звернення: 28.04.2026).
19. Qt for Beginners – Qt Wiki. Qt Wiki. URL: https://wiki.qt.io/Qt_for_Beginners (дата звернення: 28.04.2026).
20. The Animation Framework | Qt Core 5.15.19. Qt Documentation | Home. URL: https://doc.qt.io/archives/qt–5.15/animation–overview.html (дата звернення: 28.04.2026).
21. Video Overview | Qt Multimedia | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/videooverview.html (дата звернення: 28.04.2026).
22. Window and Dialog Widgets | Qt Widgets | Qt 6.11.0. Qt Documentation | Home. URL: https://doc.qt.io/qt–6/application–windows.html (дата звернення: 28.04.2026).
23. Зиков М. В., Гавор А. С. Застосування мови C++ та фреймворку Qt для розробки десктопного застосунку з керування каталогом фільмів. // VII Міжнародна науково-технічна конференція «Сучасний стан та перспективи розвитку IoT» : тези доп. (м. Київ, травень 2026 р.). Київ : ДУІКТ, 2026. С. 45–46.
24. Зиков М. В. Програмна реалізація десктопного застосунку CinemaHub для керування каталогом фільмів засобами C++ та Qt 6. // Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація» : тези доп. (м. Київ, травень 2026 р.). Київ : ДУІКТ, 2026. С. 32–33.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Розробка десктопного застосунку для керування каталогом фільмів на основі мови C++ і Qt

Виконав студент 4 курсу
Групи ТЦР-44
Зиков Михайло Віталійович
Керівник роботи
Викладач кафедри ТЦР Гавор Артур Станіславович

Київ-2026

Мета, об'єкт та предмет дослідження

Мета роботи: розробити десктопний застосунок CinemaHub для керування каталогом фільмів із сучасним графічним інтерфейсом, функціями пошуку, фільтрації, перегляду деталей, відтворення відео та керування профілем на основі мови C++ і фреймворку Qt.

Об'єкт дослідження: процес проєктування та розробки десктопного програмного забезпечення для керування мультимедійним каталогом.

Предмет дослідження: методи та засоби проєктування і реалізації десктопного застосунку для керування каталогом фільмів на основі мови C++ і фреймворку Qt.

АНАЛІЗ АНАЛОГІВ

Критерій	IMDb	Letterboxd	Kodi	MediaMonkey	CinemaHub
Робота офлайн	-	-	+	+	+
Десктопний застосунок	-	-	+	+	+
Пошук з автодоповненням	+	+	-	-	+
Відтворення відео	-	-	+	-	+
Сучасний інтерфейс	+	+	-	-	+
Персоналізація профілю	+	+	-	-	+
Зміна теми	-	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- перегляд каталогу фільмів у вигляді карток з постерами і рейтингом.
- пошук фільмів з автодоповненням при кожному введеному символі.
- фільтрація за жанром.
- відображення детальної інформації про фільм (опис, акторський склад, рейтинг).
- відкриття трейлера через системний браузер або резервний відеоплеєр.
- авторизація та реєстрація користувача з перевіркою даних.
- управління списком обраних фільмів.
- перегляд розділів: новини, тренди, огляди, сповіщення.
- перемикання між темною і світлою темою інтерфейсу.
- персоналізація профілю: нікнейм, колір аватара, фото.

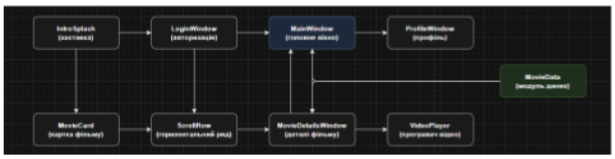
Нефункціональні вимоги:

- повна робота в офлайн-режимі без інтернету.
- час відклику інтерфейсу менше 100 мс.
- підтримка Windows 10/11.
- підтримка роздільних здатностей від 1280x720.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

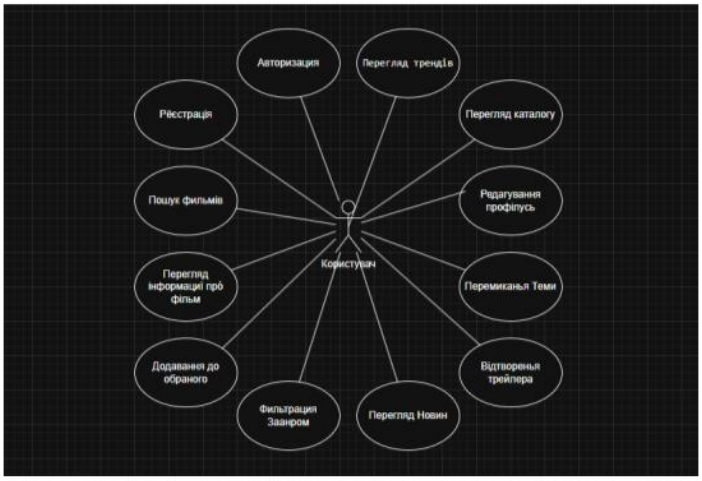
АРХІТЕКТУРА ЗАСТОСУНКУ

- Компонентна архітектура - 9 незалежних класів
- Кожен клас відповідає за один екран або модуль
- Взаємодія між класами через механізм сигналів і слотів
- Принцип єдиної відповідальності



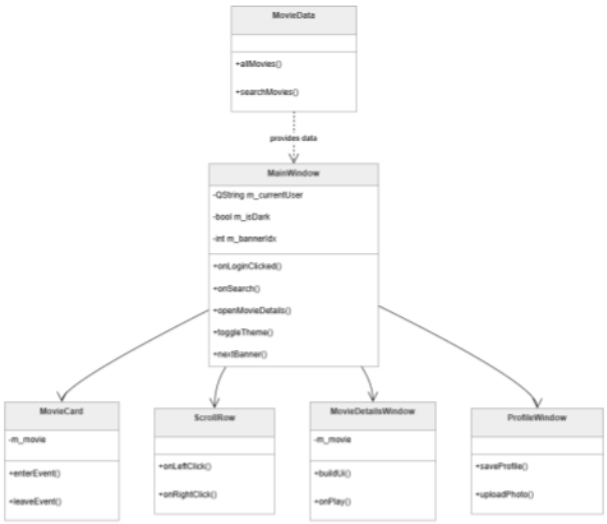
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

Діаграма відображає взаємодію користувача з основними функціями застосунку



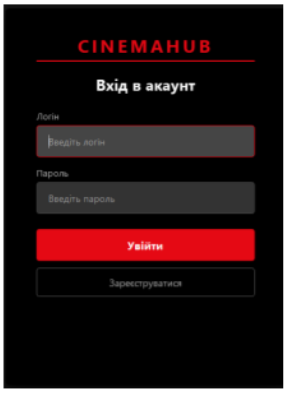
ДІАГРАМА КЛАСІВ

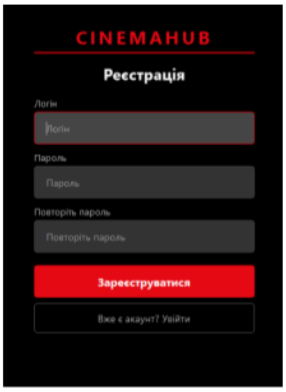
Діаграма класів показує структуру компонентів і зв'язки між ними



ЕКРАННІ ФОРМИ: ЗАСТАВКА ТА АВТОРИЗАЦІЯ

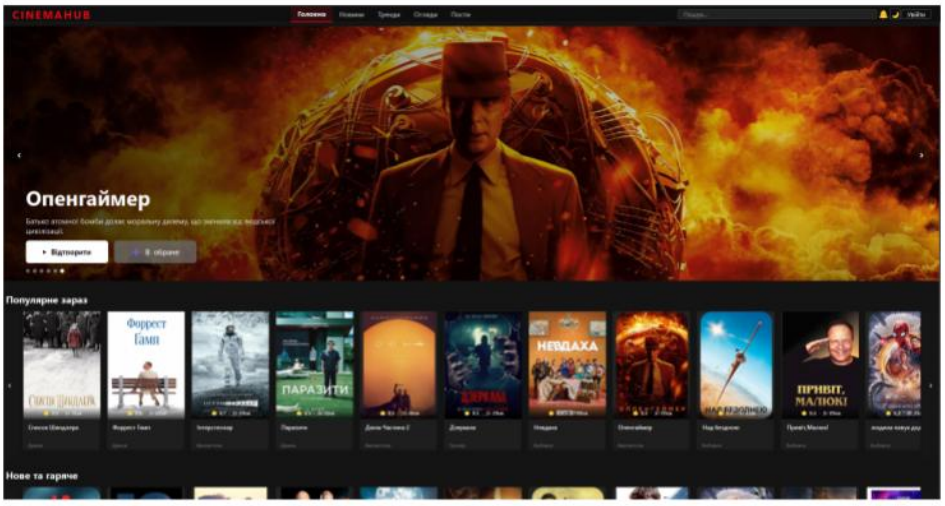
Заставка та авторизація





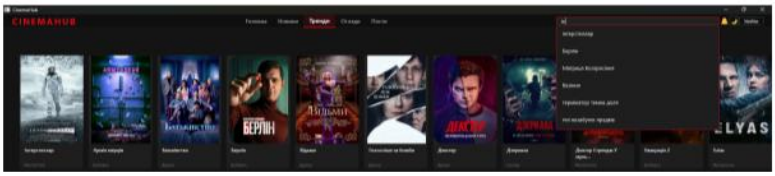
ЕКРАННІ ФОРМИ: ГОЛОВНА СТОРІНКА

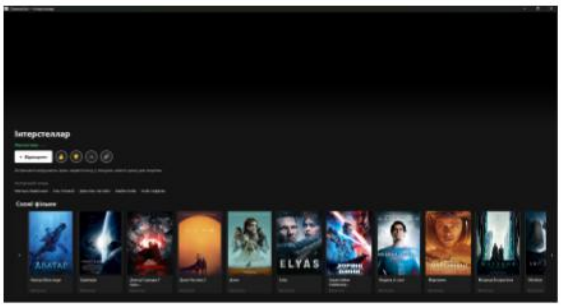
Головна сторінка



ЕКРАННІ ФОРМИ: ПОШУК І ДЕТАЛІ ФІЛЬМУ

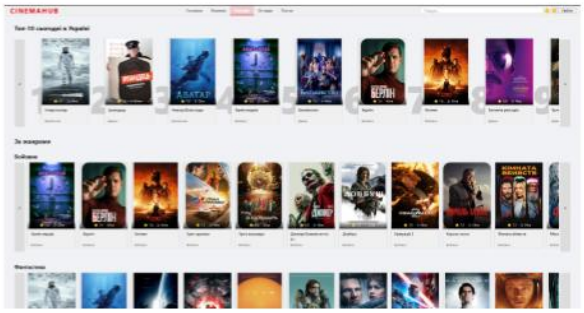
Пошук та перегляд деталей

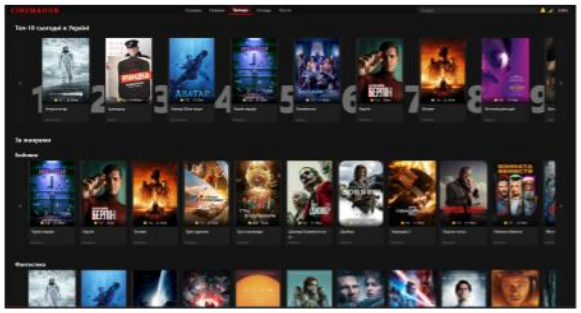




ЕКРАННІ ФОРМИ: ТЕМИ ТА ПРОФІЛЬ

Перемикання теми





АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Зиков М. В., Гавор А. С. Застосування мови C++ та фреймворку Qt для розробки десктопного застосунку з керування каталогом фільмів : тези доп. VII Міжнар. наук.-техн. конф. «Сучасний стан та перспективи розвитку IoT». Київ : ДУІКТ, 2026.

2. Зиков М. В. Програмна реалізація десктопного застосунку CinemaHub для керування каталогом фільмів засобами C++ та Qt 6 : тези доп. Всеукр. наук.-техн. конф. «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. К.: ДУІКТ, 2026, (подано до друку).

ВИСНОВКИ

Проаналізовано існуючі аналоги, жоден не поєднує офлайн-режим і сучасний інтерфейс

Обрано технологічний стек: C++17, Qt 6, CMake

Спроєктовано компонентну архітектуру з 9 незалежних класів

Реалізовано повний функціонал: каталог, пошук, трейлери, авторизація, профіль, теми

Проведено 15 функціональних і 9 модульних тестів — усі пройдено успішно

Застосунок працює в повному офлайн-режимі без залежності від мережі

ДОДАТОК Б. ЛІСТИНГ ВИХІДНОГО КОДУ ЗАСТОСУНКУ CINEMAHUB

```
//main.cpp
#include "mainwindow.h"
#include "introsplash.h"
#include <QApplication>
#include <QTimer>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    a.setApplicationName("CinemaHub");

    auto* splash = new IntroSplash();
    splash–>show();

    auto* w = new MainWindow();

    QObject::connect(splash, &IntroSplash::finished,
splash, [splash, w]() {
        splash–>hide();
        splash–>deleteLater();
        w–>showMaximized();
        QTimer::singleShot(150, w,
&MainWindow::refreshBanner);
    });

    return a.exec();
}

//moviedata.h

#ifndef MOVIEDATA_H
#define MOVIEDATA_H

#include <QString>
#include <QStringList>
#include <QVector>
#include <QSet>

class QPixmap;

struct Movie {
    int         id;
    QString     title;
    QString     genre;
    int         year;
    QString     description;
    QString     imagePath;
    QStringList cast;
    float       rating   = 0.0f;
    int         duration = 0;
};

const QVector<Movie>& allMovies();
const Movie*          movieById(int id);
QVector<Movie>        moviesByGenre(const QString&
genre);
QVector<Movie>        searchMovies(const QString&
query);

QPixmap loadPoster(const QString& imagePath);

QSet<int>& favorites();
void       toggleFavorite(int id);
bool       isFavorite(int id);

#endif


//introsplash.cpp
#include "introsplash.h"
#include <QVBoxLayout>
#include <QApplication>
#include <QScreen>
#include <QCoreApplication>
#include <QFileInfo>
#include <QDebug>
#include <QPropertyAnimation>
#include <QEasingCurve>
#include <QAbstractAnimation>

IntroSplash::IntroSplash(QWidget* parent)
    :  QWidget(parent,  Qt::FramelessWindowHint  |
Qt::WindowStaysOnTopHint)
{
    const QRect screen = QApplication::primaryScreen()–
>geometry();
    setGeometry(screen);
    setStyleSheet("background:#000000;");

    QVBoxLayout* layout = new QVBoxLayout(this);
    layout–>setContentsMargins(0,0,0,0);
    layout–>setSpacing(0);

    m_videoWidget = new QVideoWidget(this);
    m_videoWidget–
>setStyleSheet("background:#000000;");
    m_videoWidget–
>setSizePolicy(QSizePolicy::Expanding,
QSizePolicy::Expanding);
    layout–>addWidget(m_videoWidget);

    m_player = new QMediaPlayer(this);
    m_audio  = new QAudioOutput(this);
    m_audio–>setVolume(1.0f);
    m_player–>setAudioOutput(m_audio);
    m_player–>setVideoOutput(m_videoWidget);
    m_player–>setPlaybackRate(0.80);

    const          QString          appDir          =
QCoreApplication::applicationDirPath();
    const QString cwd    = QDir::currentPath();
```

```
    QStringList candidates = {
        appDir + "/videos/videos–intro.mp4",
        appDir + "/videos/intro.mp4",
        cwd    + "/videos/videos–intro.mp4",
        cwd    + "/videos/intro.mp4",
        "videos/videos–intro.mp4",
        "videos/intro.mp4",
    };

    QString videoPath;
    for (const QString& p : candidates) {
        if (QFileInfo::exists(p)) { videoPath = p; break; }
    }

    if (videoPath.isEmpty()) {
        qWarning() << "[IntroSplash] Video not found! Tried:" << candidates;
        QTimer::singleShot(0, this, &IntroSplash::finished);
        return;
    }

    qDebug() << "[IntroSplash] Loading:" << videoPath;
    m_player–>setSource(QUrl::fromLocalFile(videoPath));

    connect(m_player, &QMediaPlayer::mediaStatusChanged,
            this, &IntroSplash::onMediaStatusChanged);
    connect(m_player, &QMediaPlayer::errorOccurred,
            this, &IntroSplash::onErrorOccurred);

    m_safetyTimer = new QTimer(this);
    m_safetyTimer–>setSingleShot(true);
    m_safetyTimer–>setInterval(6000);
    connect(m_safetyTimer, &QTimer::timeout, this, &IntroSplash::doFadeOut);
    m_safetyTimer–>start();

    m_effect = new QGraphicsOpacityEffect(this);
    m_effect–>setOpacity(1.0);
    setGraphicsEffect(m_effect);

    m_player–>play();
    qDebug() << "[IntroSplash] play() called, source:" << m_player–>source();

}

void IntroSplash::onMediaStatusChanged(QMediaPlayer::MediaStatus status)
{
    qDebug() << "[IntroSplash] status:" << status;
    if (status == QMediaPlayer::EndOfMedia)
        doFadeOut();
}

void IntroSplash::onErrorOccurred(QMediaPlayer::Error error, const QString& msg)
{
    qWarning() << "[IntroSplash] Error" << error << ":" << msg;
    doFadeOut();
}

void IntroSplash::doFadeOut()
{
    if (m_fadingOut) return;
    m_fadingOut = true;
    if (m_safetyTimer) m_safetyTimer–>stop();
    if (m_player)      m_player–>stop();
    if (!m_effect)     { emit finished(); return; }

    auto* anim = new QPropertyAnimation(m_effect, "opacity", this);
    anim–>setDuration(600);
    anim–>setStartValue(1.0);
    anim–>setEndValue(0.0);
    anim–>setEasingCurve(QEasingCurve::InOutQuad);
    connect(anim, &QPropertyAnimation::finished, this, [this]() {
        emit finished();
    });
    anim–>start(QAbstractAnimation::DeleteWhenStopped);
}

//loginwindow.cpp (фрагмент)
void LoginWindow::onLogin()
{
    const QString login = m_loginEdit–>text().trimmed();
    const QString pass  = m_passEdit–>text();
    if (login == VALID_LOGIN && pass == VALID_PASS) {
        m_loginErr–>setVisible(false);
        m_user = login;
        accept();
    } else {
        m_loginErr–>setText("⚠  Невірний логін або пароль");
        m_loginErr–>setVisible(true);
        m_passEdit–>clear();
        m_passEdit–>setFocus();
    }
}
```