Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРІ

Виконала: студентка 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Хоменчук Ж.С.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри КН
Озеранський В.С.
(прізвище та ініціали)
«04» 06 2025 р.

Рецензент: к.т.н., доцент кафедри САІТ
Варчук І.В.
(прізвище та ініціали)
«05» 06 2025 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«06» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"05" 05 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Хоменчук Жанні Станіславівні

1. Тема роботи: Програмний модуль бронювання квитків у кінотеатрі.
керівник роботи: Озеранський Володимир Сергійович, к.т.н., доцент
затверджені наказом вищого навчального закладу "10" 03 2025 року № 97

2. Строк подання студентом роботи 30.05.2025р.

3. Вихідні дані до роботи:
максимальна кількість фільмів на сторінці – не менше 10шт;
можливість пошуку фільмів за тегом;
термін зберігання плей-листа користувача – 14днів;
кількість одночасних користувачів – не менше 100чол;
Мова програмування – об'єктно-орієнтована.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
вступ; аналіз предметної області бронювання квитків у кінотеатрі; розробка програмного модуля бронювання квитків у кінотеатрі; програмна реалізація модуля бронювання квитків у кінотеатрі висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
схема загального алгоритму функціонування програмного модуля бронювання квитків у кінотеатрі; структура програмного модуля бронювання квитків у кінотеатрі; загальна UML-діаграма класів.

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Озеранський В.С., к.т.н., доцент		

6. Дата видачі завдання 05 травня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Аналіз предметної області бронювання квитків у кінотеатрі	05.05.25	02.05.25	
2.	Розробка програмного модуля бронювання квитків у кінотеатрі	08.05.25	11.05.25	
3.	Програмна реалізація модуля бронювання квитків у кінотеатрі	12.05.25	15.05.25	
4.	Розробка інструкції користувача.	18.05.25	26.05.25	
5.	Висновки та аналіз отриманих результатів	22.05.25	01.06.25	
6.	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студентка _____ Хоменчук Ж
(підпис) (прізвище та ініц

Керівник проєкту (роботи) _____ Озеранський
(підпис) (прізвище та ініц

АНОТАЦІЯ

Хоменчук Ж.С. Програмний модуль бронювання квитків у кінотеатрі. Бакалаврська кваліфікаційна робота складається з 75 сторінки формату А4, на яких є 25 рисунків, 3 таблиці, список використаних джерел містить 20 найменувань.

В роботі обґрунтовано доцільність розробки програмного модуля бронювання квитків у кінотеатри. Проведено аналіз переваг та недоліків сучасних програм-аналогів, які використовуються для бронювання квитків у кінотеатрі. Наведено короткий опис основних функцій, які виконують дані програми. Розроблено UML-діаграми програмного модуля бронювання квитків у кінотеатрі. Побудовано інформаційну модель основних модулів програмного забезпечення. Проведено вибір середовища розробки мови програмування. Для програмної реалізації модуля бронювання квитків у кінотеатрі було обрано ОС Android, як систему, що має відкритий тип, багатий функціонал і є найбільш поширеною.

Програмний модуль розроблено на об’єктно-орієнтованій мові програмування Java, враховуючи її кросплатформність та простоту розробки, в середовищі розробки Android Studio.

Ключові слова: Android, Java, бронювання, квиток, кінотеатр, мобільний пристрій.

ABSTRACT

Khomenchuk Zh.S. Software Module for Cinema Ticket Booking. The bachelor's qualification thesis consists of 75 A4-format pages, including 25 figures, 3 tables, and a list of references containing 20 sources.

The work substantiates the feasibility of developing a software module for booking tickets in cinemas. An analysis of the advantages and disadvantages of existing software analogs currently used for cinema ticket booking is provided. A brief description of the main functions performed by these programs is presented. UML diagrams of the software module for cinema ticket booking were developed. An informational model of the main software modules was constructed. The choice of programming language and development environment was justified.

For the software implementation of the cinema ticket booking module, the Android operating system was selected as a platform due to its open-source nature, rich functionality, and widespread use. The software module was developed in the object-oriented programming language Java, taking into account its cross-platform capabilities and ease of development, using the Android Studio development environment.

Keywords: Android, Java, booking, ticket, cinema, mobile device.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний рівень розвитку Інтернет-технологій призвів до того, що багато аспектів людської діяльності передаються через Інтернет. Цей аспект — процес купівлі-продажу. Прикладами систем електронної комерції та типів інтернет-магазинів є системи замовлення квитків у кінотеатри. Інтернет-магазин - це місце в Інтернеті, де відбувається безпосередній продаж товару споживачеві, а також куди відбувається доставка. Розміщення інформації про споживача, замовлення товарів і операції відбуваються там же, в межах однієї мережі (на сайті інтернет-магазину). У контексті системи замовлення квитків у кіно продукт – це квиток на сеанс, на фільм у кіно [1].

Сьогодні за допомогою сучасних інформаційних технологій можна ефективно організувати практично будь-яку компанію та велику компанію. Мультимедійні розважальні компанії не є винятком, тому автоматизація кінотеатрів є пріоритетом для багатьох власників прокату фільмів. Сьогодні існує багато сервісів, які пропонують автоматизацію процесу вибору та покупки квитків у кіно. Більшість сайтів пропонують пробний період, а в майбутньому — платний план на основі комісій або план, створений для спеціальної мережі кінотеатрів і використовується виключно цією мережею [2].

Автоматизація кінотеатру чи цілої мережі — сьогодні це не просто перевага, а нагальна необхідність. В умовах стрімкого розвитку цифрових технологій та зростаючих очікувань глядачів, автоматизовані системи стають основою ефективного функціонування будь-якого сучасного кінотеатру. Якісний звук, чітке зображення, комфортні сидіння та атмосфера залу — усе це важливо, але без злагодженої роботи автоматизованих сервісів, зокрема системи продажу й резервування квитків, створити по-справжньому позитивний досвід для відвідувача неможливо.

Сучасний глядач уже не готовий миритися з технічними недоліками або застарілими процесами. Неприпустимою вважається ситуація, коли кілька людей отримують квитки на одне і те ж місце — це не просто непорозуміння, а серйозний

удар по репутації кінотеатру. Крім того, відвідувачі очікують, що зможуть придбати або забронювати квитки зручним для себе способом — не виходячи з дому, через мобільний додаток, офіційний сайт або навіть за телефоном. Відсутність такої можливості сприймається як недолік і може стати причиною втрати потенційної аудиторії.

Автоматизовані системи забезпечують зручність, швидкість, точність та прозорість у взаємодії між кінотеатром і глядачем. Вони дозволяють оперативно оновлювати розклад сеансів, керувати наявністю місць у режимі реального часу, проводити аналітику продажів і розробляти ефективні маркетингові кампанії. Крім того, автоматизація сприяє підвищенню рівня безпеки та зменшує ризики людських помилок.

Отже, впровадження автоматизованих рішень — це не лише про комфорт і престиж, а про конкурентоспроможність і розвиток. У сучасному світі, де швидкість і зручність є ключовими цінностями, кінотеатри мають адаптуватися до нових реалій, аби відповідати запитам аудиторії та залишатися актуальними в культурному просторі.

Метою бакалаврської кваліфікаційної роботи є підвищення якості роботи програмного забезпечення бронювання квитків у кінотеатрі.

Об’єктом дослідження є процес бронювання квитків у кінотеатрі.

Предметом дослідження є програмні засоби бронювання квитків у кінотеатрі.

Для досягнення поставленої мети необхідно виконати такі задачі:

- проаналізувати предметну область бронювання квитків у кінотеатрі;
- проаналізувати переваги та недоліки програм-аналогів для бронювання квитків у кінотеатрі;
- розробити алгоритм бронювання квитків у кінотеатрі;
- здійснити програмну реалізацію модуля бронювання квитків у кінотеатрі;
- виконати тестування програмного модуля та провести аналіз отриманих результатів.

Результати дослідження впроваджено в роботу ТОВ «ІТІ».

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРІ

1.1 Аналіз технологій продажу квитків в кінотеатр

Процес автоматизації кінотеатру передбачає розробку та впровадження програмного забезпечення для продажу та автоматичного розрахунку квитків з урахуванням різних типів місць, цінової і акційної політики, поточних програм обслуговування клієнтів, систем знижок та інших акцій. Процес автоматизації нерозривно пов’язаний не тільки з оновленням програмного забезпечення, а й з оновленням, придбанням нового обладнання та витратами на його впровадження та обслуговування. Цей список має включати комп’ютер для кожного продавця-касира, серверне обладнання, принтер для квитків, скарбнички, а також різні перемикачі та комутації [1].

Система повинна забезпечувати централізоване управління всіма процесами, пов’язаними з прийняттям та виконанням замовлення, щоб керівник міг своєчасно отримувати достовірну інформацію та на її основі будувати правильну економічну політику підприємства [2].

Позитивний ефект від вмілого використання систем автоматичного керування в організаціях кінотеатрів незаперечний. А в умовах економічної кризи ІТ може стати серйозним інструментом для оптимізації управління, зниження витрат і надання незаперечних конкурентних переваг на ринку. Автоматизація бізнес-процесів є запорукою ефективного управління. Автоматизація бізнес-процесів призводить до скорочення рутинних операцій, дозволяє значно швидше обслуговувати клієнтів, дає більше контролю, бізнес-процеси стають більш «прозорими». Значно покращено роботу над плануванням пошуку та доставки та іншими перевагами. Все це, в свою чергу, значно збільшує прибуток, товарообіг і виручку, знижує витрати. Скорочення рутинної діяльності істотно сприяє зниженню витрат на персонал [2].

Автоматизація роботи кінозалу, а тим більше багатозального кінотеатру, сьогодні так само необхідна, як і встановлення екрану. Хороший звук, гарне

зображення та зручні сидіння за замовчуванням мають поєднуватися з прозорою роботою системи продажу та бронювання квитків. Адже сучасний кіноман уже звик до такого анахронізму, як продаж кількох квитків в одне місце. Йому також важко уявити, що квитки в кіно не можна замовити по телефону чи онлайн [2].

Система замовлення квитків у кіно є одним із видів онлайн-покупок. Інтернет-магазин - сторінка, з якої можна вибрати і замовити відповідний товар. У нашому випадку таким продуктом є квиток на сеанс у кінотеатрі. Інтернет-магазин повинен надавати розрахунки готівкою, а не як безкоштовні послуги. Основними елементами інтернет-магазину є оновлення наявного асортименту (товарів/товарів), можливість додавання товарів у «кошик», авторизація для авторизованих користувачів. Інтернет-магазини розробляються у вигляді веб-додатків. Веб-додаток - це розподілений додаток, у якому браузер є клієнтом, а веб-сервер - сервером. Браузер може бути частиною реалізації так званих тонких клієнтів. Браузер повинен відображати веб-сторінки та бути частиною операційної системи, а його оновлення та обслуговування має вирішувати постачальник операційної системи. Логіка програми виконується на сервері, а його функція браузера в основному полягає в тому, щоб відображати інформацію, завантажену по мережі з сервера, і надсилати дані назад користувачеві [3]. Перевага цього підходу полягає в тому, що клієнти не залежать від конкретної операційної системи клієнта, тому веб-додатки є міжплатформними сервісами. Великою перевагою створення веб-додатків, які підтримують стандартні функції браузера, є те, що ці функції повинні виконуватися незалежно від операційної системи користувача. Щоб не писати різні версії для операційних систем Microsoft Windows, Mac OS X, GNU/Linux, програма розроблена один раз для будь-якої платформи, на якій вона відображається, однак різні реалізації HTML, CSS, DOM та інших специфікацій у браузерах може викликати проблеми з розробкою та розробкою веб-додатків. Крім того, користувач має можливість змінити багато налаштувань браузера (наприклад, розмір шрифту), які можуть заважати роботі деяких програм [3].

Веб-додаток отримує запит від клієнта, потім виконує деякі обчислення, потім зчитує веб-сторінку і надсилає її користувачеві по мережі за допомогою протоколу

«http». Веб-додаток також є клієнтом інших служб, таких як база даних або інший веб-додаток, розташований на другому сервері. Новий підхід до розробки веб-додатків AJAX був розроблений для підвищення ефективності впровадження та підвищення продуктивності і тепер є стандартним інструментом розробки веб-додатків. За допомогою сторінки AJAX веб-додаток може відправляти веб-запити на сервер у фоновому режимі, і не перезавантажувати сторінку, а лише завантажувати деякі дані з сервера, що значно прискорює роботу та робить її більш зручною [3]. При створенні веб-додатків використовуються різні методи, найпоширенішим з яких є шаблон оформлення модель-вигляд-контролер. Model-view-controller - архітектурний шаблон, який використовується для проектування та розробки програмного забезпечення. Основною метою цього шаблону є інший дизайн програмного забезпечення, який полегшує зміну або розширення програм, а також надає можливість повторного використання деяких компонентів програмного продукту. Крім того, використання шаблону model-view-controller у великих системах зазвичай дещо організувати їх структуру та зробити їх придатними для повторного використання, краще зрозумілі через меншу складність. Шаблон архітектури Model-View-Controller (MVC) ділить програму на три частини. У тріаді обов’язки компонента Model полягають у зберіганні даних та забезпеченні інтерфейсу для них. View несе відповідальність за надання цієї інформації користувачеві. Контролер керує компонентами, отримує сигнали у відповідь на дії користувача та повідомляє про зміни компоненту Model. Ця внутрішня структура в цілому розділяє систему на незалежні частини і розподіляє відповідальність між різними компонентами [3].

1.2 Аналіз переваг та недоліків систем-аналогів бронювання квитків в кінотеатрі

Автоматизація процесів продажу квитків повинна включати розробку програмного модуля, який гарантує надійне збереження та обробку даних про замовлення. У сучасному кінотеатрі, де функціонує кілька залів різного типу (2D,

3D, IMAX тощо), відбувається постійний потік відвідувачів, а в репертуарі одночасно представлено десятки фільмів, виникає необхідність ефективного контролю над розсадкою глядачів. Система має в режимі реального часу фіксувати продаж квитків, відображати актуальний стан вакантних і зайнятих місць, забезпечувати бронювання, обробку знижок та акцій, а також інтеграцію з іншими підсистемами, такими як касовий апарат, мобільний додаток і веб-сайт для онлайн-бронювання. Автоматизований облік дозволяє мінімізувати людський фактор, уникати дублювання або втрати даних, прискорювати процес обслуговування клієнтів, а також збирати аналітику для подальшої оптимізації діяльності кінотеатру.

Однією з ключових вимог до створюваного програмного модуля бронювання квитків у кінотеатр є надійне зберігання вихідних даних, зокрема таблиць з інформацією про користувачів, фільми, розклади, доступні місця, заброньовані квитки тощо. Всі ці дані повинні зберігатися у зовнішньому файлі або базі даних таким чином, щоб забезпечити їхню цілісність і збереження навіть після завершення роботи програми або при непередбаченому її завершенні (наприклад, через збій системи).

У контексті локального зберігання це може бути реалізовано шляхом серіалізації об'єктів у форматах JSON, XML, CSV або за допомогою використання реляційної або нереляційної бази даних. Такий підхід дозволяє зберігати повний знімок поточного стану системи, включаючи вже здійснені бронювання, наявні вільні місця, історію транзакцій тощо.

Особливу увагу слід приділити синхронізації даних: усі зміни, які користувач або адміністратор вносить під час роботи з програмою (наприклад, підтвердження замовлення, скасування бронювання, зміна розкладу сеансів), мають одразу або з певною періодичністю записуватися у відповідний файл або базу даних. Це забезпечить консистентність інформації та дозволить відновити актуальний стан системи при наступному запуску програми.

Для уникнення втрати даних також доцільно передбачити механізми автоматичного резервного копіювання та відновлення інформації. Це може бути

реалізовано через періодичне створення копій основного файлу або бази на окремому носії чи у хмарному сховищі.

Таким чином, наявність механізму збереження та оновлення вихідних даних у файлі є не лише зручністю, а й необхідною умовою стабільної та надійної роботи програмного модуля. Це гарантує безперервність обслуговування клієнтів, збереження історії продажів, а також дає змогу проводити аналіз і облік усіх операцій, пов’язаних з квитками, навіть після перезапуску або оновлення системи.

База даних повинна включати:

- інформацію про фільм (назва, жанр, країна виробництва, рік виходу, тривалість, вікові обмеження);

- вартість квитка - залежить від часу сеансу (враховуються переваги, наприклад, для учнів та студентів) та місця в кімнаті (перший поверх, балкон, антресолі, ложі, VIP місця);

- інформацію про сеанс (тривалість, час і дата показу фільму типу кінозалу (великий зал, VIP-зал тощо));

- тематичні кінопокази (дитячі, документальні, авторські тощо);

- кількість місць у кінозалі (враховуються як продані, так і наявні місця).

Користувач системи повинен мати можливість виконувати такі запити:

- отримати список часу показу одного відео за датою або часу показу відео в будні та вихідні дні;

- опублікувати список фільмів, які будуть показані в певний день тижня, у вибраний період часу, у всіх кінотеатрах;

- отримати список та загальну кількість фільмів за назвою, жанром, тривалістю тощо;

- оприлюднювати перелік тематичних кінопоказів, які відбуваються в окремих кінотеатрах на визначений період часу;

- створити перелік цін на квитки на даний фільм залежно від часу показу, його тривалості, типу кінотеатру, місця в залі;

Програмне забезпечення бронювання квитків у кінотеатр має складатися з трьох основних модулів.

Модуль адміністрування, призначений для визначення геометрії місць для глядачів, створення та редагування геометричних і цінових схем, налаштування звітів, налаштування форм квитків, керування дозволами користувачів, а також загальних налаштувань і конфігурацій системи.

Модуль продажу та бронювання квитків повинен обслуговувати клієнтів швидко та легко, з можливістю застосування різноманітних знижок та обслуговування клубних карток. Обов’язково за результатами роботи автоматично складаються всі необхідні звіти для обліку та управління. Простий системний інтерфейс повинен дозволити в найкоротші терміни пройти навчання касирів.

Модуль підготовки звітності та взаєморозрахунків з орендарями повинен дозволяти повністю автоматизувати підготовку будь-яких звітів для відділу маркетингу та управління. Обов’язково, щоб облік за договорами з дистриб’юторами для кожного примірника фільму дозволяв автоматично створювати та експортувати облікові документи.

Проаналізуємо сучасні сервіси, які дозволяють здійснити замовлення квитків у кінотеатр.

Сервіс Planeta-kino.ua надає користувачам зручну можливість онлайн-бронювання квитків до мережі кінотеатрів IMAX, які функціонують у різних містах України, зокрема в таких великих культурних і економічних центрах, як Київ, Львів, Одеса та Харків [2]. Цей сервіс є прикладом сучасного інтерфейсу для взаємодії користувача з багатофункціональною системою бронювання, що поєднує в собі простоту, швидкість та візуальну зручність.

Для кожного кінотеатру на сайті реалізовано індивідуальне облаштування глядацьких залів, яке відповідає їхній реальній конфігурації — включаючи кількість рядів, кількість місць у кожному ряду, розташування екрана, а також спеціальні зони, наприклад VIP-сектори, місця для людей з інвалідністю, або місця з підвищеним комфортом (RE'LUX тощо). Це забезпечує максимально наближене до реального середовища планування, що сприяє прийняттю обґрунтованого вибору при бронюванні.

Користувач може інтерактивно вибрати потрібний зал, сеанс і фільм, а також перейти до перегляду плану залу в інтерактивному форматі, де кожне місце представлено у вигляді клітинки з візуальним позначенням статусу: вільне, зайняте, заброньоване, або недоступне. Завдяки такому підходу система дозволяє в режимі реального часу оцінити наявність вільних місць та зробити точний вибір відповідно до власних уподобань — ближче до екрана, у центрі зали, на останньому ряду тощо.

На рисунку 1.1 продемонстровано приклад планування макету глядацького залу, який реалізовано в інтерфейсі Planeta-kino.ua. Даний макет є візуалізацією реального розташування сидінь у конкретному залі, що значно підвищує зручність користувача та мінімізує ймовірність помилкового вибору місця. Крім того, інтеграція з платіжною системою дозволяє одразу після вибору здійснити оплату квитка, після чого він зберігається в особистому кабінеті або надсилається на електронну пошту.

Таким чином, сервіс Planeta-kino.ua виступає ефективним прикладом автоматизованої системи продажу квитків з повноцінною підтримкою взаємодії з реальними просторовими даними кінотеатрів, що значно покращує досвід користувача.

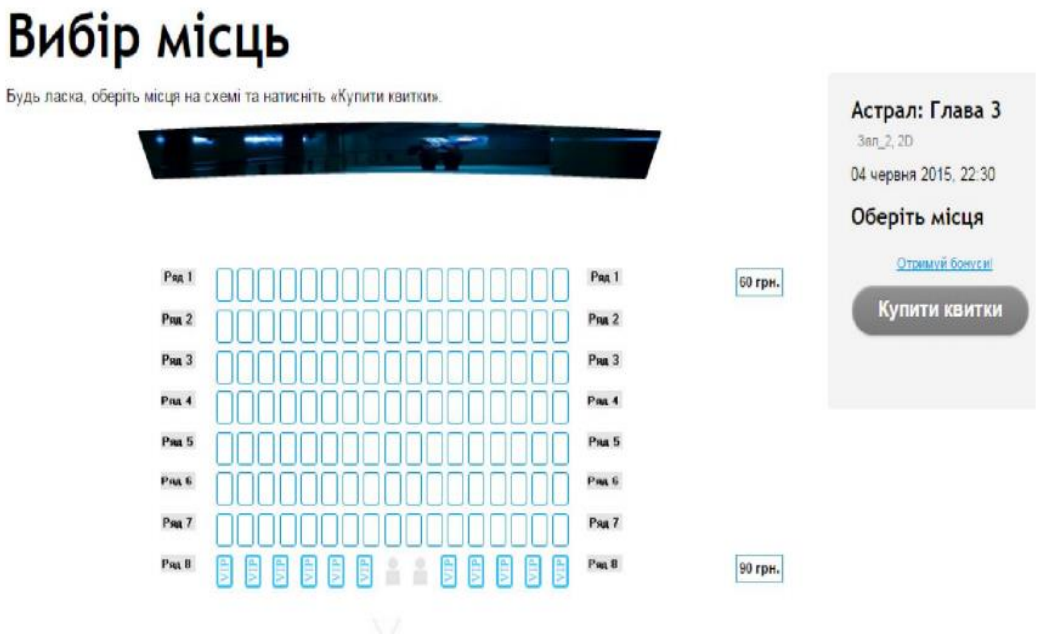

Рисунок 1.1 – Макет глядацького залу у сервісі Planeta-kino.ua

Щоб замовити квиток у кіно, користувачу необхідно зареєструватися на сайті. Для зареєстрованих користувачів доступна клубна карта, за допомогою якої можна отримати знижки.

На сайті сервісу Kinoafisha.ua можна ознайомитися з розкладами кіносеансів, які зараз доступні для прокату майже в усіх кінотеатрах України. Однак не всі кінотеатри можуть забронювати квитки, що створює певні незручності для користувачів. Для кожного кінотеатру, до якого можна замовити квитки на даний сеанс, розроблено макети існуючих кінозалів. Приклад планування кінозали наведено на рисунку 1.2 [3].

Щоб замовити квиток на певний сеанс даного фільму, реєстрація не потрібна, але необхідно увійти за допомогою облікового запису однієї з соціальних мереж [3, 4].

Рисунок 1.2 – Макет глядацького залу в сервісі Kinoafisha.ua

Ресурс MULTIPLEX - це велика мережа кінотеатрів в Україні заснована у 2004 році. MULTIPLEX - це 28 кінотеатрів та 141 кінозал у найбільших містах України.

Відповідно до світової практики корпоративного управління, у серпні 2018 року був сформований найвищий орган управління компанією MULTIPLEX – борд, до складу якого увійшли впливові українські підприємці, управлінці і лідери громадської думки. MULTIPLEX активно підтримує та інвестує у розвиток українського кінематографу (рис 1.3) [4].

Одним з основних недоліків розглянутих веб-сервісів є те, що їх розробка не використовує технологію AJAX, а оскільки цей дефект вимагає перезавантаження сторінки після кожної дії користувача, то це ускладнює використання веб-сервісу, особливо при повільному підключенні до Інтернету. Крім того, інтерфейс обох веб-ресурсів не адаптований для мобільних пристроїв.

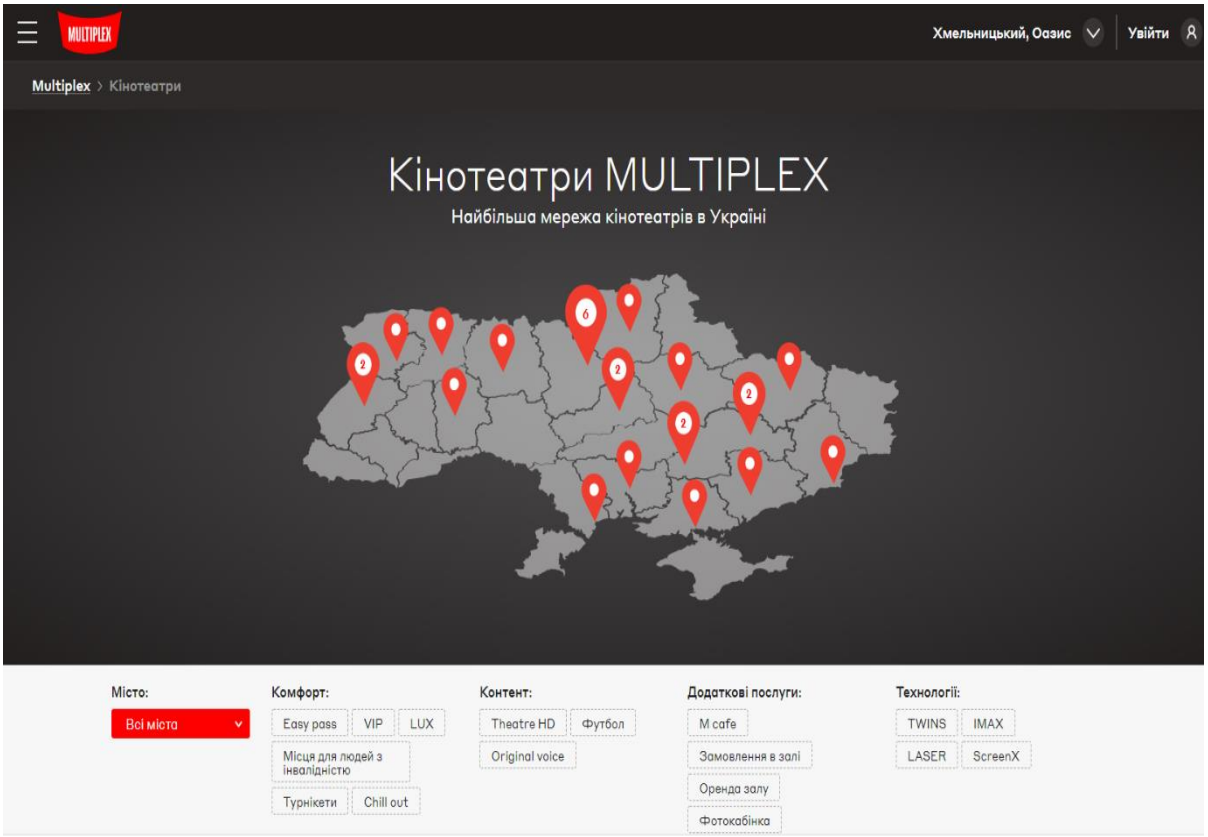

Рисунок 1.3 – Головне вікно сервісу MULTIPLEX

1.3 Висновок до розділу 1

В даному розділі було проаналізовано особливості автоматизації бронювання квитків у кінотеатри. Розглянуто принципи попереднього продажу, бронювання та

реалізації квитків у кінотеатри у квиткових касах. Проведено аналіз переваг та недоліків сучасних програм-аналогів, які використовуються для замовлення квитків у кінотеатр. Наведено короткий опис основних функцій, які виконують дані програми.

В результаті цього аналізу сформулюємо вимоги до програмного модуля бронювання квитків у кінотеатри:

- зручний та зрозумілий з точки зору UI/UX інтерфейс;
- висока швидкодія та зручність у користуванні сервісом;
- система повинна забезпечувати високу швидкість та надійність у роботі;
- система повинна забезпечувати можливість додання та редагування інформації про фільми, кінотеатри і сеанси, не змінюючи при цьому початковий екран сервісу, завдяки впровадженням адмін-панелі;
- забезпечувати можливість замовлення квитків.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРІ

2.1 Розробка структури програмного модуля бронювання квитків в кінотеатрі

Функціональні моделі, які використовуються на етапі проектування, призначені для опису функціональної структури проектованої системи. Побудовані блок-схеми DFD можна покращувати, розширювати та доповнювати новими конструкціями. Крім блок-схем DFD, використовуються інші діаграми, які відображають архітектуру програмної системи, ієрархію екранів і меню, блок-схеми програм тощо. Склад діаграм і ступінь їх деталізації визначаються повнотою опису системи, необхідної для безпосереднього переходу до її подальшої реалізації (програмування).

Наприклад, у випадку блок-схеми DFD перехід від моделі бізнес-процесу організації до моделі системного процесу може бути таким:

- зовнішні сутності в контекстній діаграмі замінені або доповнені технічними пристроями (наприклад, робочими станціями, принтерами тощо);

- для кожного потоку даних вказується, якими технічними пристроями виробляється чи передається інформація;

- процеси на діаграмі нульового контексту замінюються відповідними процесорами - пристроями обробки (процесорами можуть бути технічні пристрої - персональні комп'ютери кінцевих користувачів, робочі станції, сервери баз даних, які слугують такими - підрядниками);

- тип зв'язку між процесорами (наприклад, локальна мережа - LAN) визначено і показано на схемі;

- для кожного процесора визначаються завдання (додатки, необхідні для роботи системи), для них будуються відповідні діаграми та визначається тип зв'язку між завданнями;

- створюються зв'язки між завданнями і процесами блок-схем цих послідовних рівнів.

Ієрархія екранних форм моделюється за допомогою діаграм послідовності масок. Набір таких діаграм являє собою абстрактну модель інтерфейсу користувача системи, послідовність появи екранів виведення у додатку. Діаграми послідовності екранних масок будуються таким чином:

- У блок-схемі DFD вибираються інтерактивні процеси низького рівня. Інтерактивні процеси вимагають інтерфейсу користувача, щоб можна було визначити екран для кожного такого процесу;

- форма діаграми представлена у вигляді прямокутника для кожного інтерактивного процесу внизу діаграми;

- структура меню встановлена. Для цього інтерактивні процеси групуються в меню (як у моделі процесу, або інакше - за функціональними характеристиками або залежно від належності до конкретних об'єктів);

- форми з меню представлені над формами, що відповідають інтерактивним процесам, і пов'язані з ними переходами у вигляді стрілок, що вказують з меню на форми;

- встановлюється верхня форма (основна форма заявки), яка пов'язує всі форми з меню.

Кінотеатр — це громадська будівля (або його частина), пристосована для показу фільмів.

Сучасний кінотеатр має що запропонувати глядачам: цікавий фільм, якість зображення та гарний звук, комфортні кімнати зі зручними кріслами. Але кінотеатр починається зі звичайної каси, яка продає квитки.

Основним функціональним обов’язком касира кінотеатру є процес продажу квитків глядачам.

У кінотеатрі є розклад з інформацією про фільми та ціни на квитки. Також у кінотеатрі є каса, де відвідувач може придбати квиток на сеанс.

У цій базі даних зберігається інформація про час сеансів і вартість квитків, а також інформація про вільне місце для сеансу, інформацію про поточний фільм, жанр фільму, вікові обмеження на перегляд фільму.

Дані згруповані в розробленій системі таким чином:

- список проданих квитків (дата продажу квитків, для якого сеансу, місце, серіал, назва фільму, жанр, вікові обмеження);

- режим роботи кінотеатру (час сеансу, ціна квитка на сеанс);

- репертуар кіно на сьогодні (час показу, назва фільму, жанр);

- архів кінотеатру (дані про всі наявні в кінотеатрі фільми).

Розроблена система має можливість ведення даних: упорядкування таблиць, встановлення режиму роботи кінотеатру та підключення до них, введення та редагування даних у таблицях.

Крім того, розроблений продукт має такі вимоги:

- вилучення всіх проданих квитків на сеанс;

- зняття всіх проданих денних квитків;

- Вилучення всіх коли-небудь проданих квитків у кінотеатрі;

- розрахунок прибутку від продажу квитків за сеанс;

- розрахунок прибутку від продажу квитків за день;

- розрахунок загального прибутку кінотеатру від продажу всіх квитків.

Вхід – це інформація, яку користувач вводить у файл бази даних, заповнюючи необхідні поля обраної таблиці, а також вводячи інформацію в базу даних за допомогою SQL-запитів. В якості вихідних даних для розробленого програмного забезпечення буде використана така інформація:

- інформація про сеанси (час сеансу, ціна квитка на цей сеанс);

- інформація про придбані квитки (дата продажу, місце та кількість в аудиторії);

- інформація про доступні фільми (назва фільму, жанр, вікові обмеження для перегляду цього фільму).

Вихідна інформація - результат запиту, фільтрація даних, вихід необхідної інформації у звіт, друк інформації. Інформація, яка виводить або узагальнює дані в цілому або відповідно до визначених критеріїв.

Вихідною інформацією для цього проекту є інформація, яка дозволяє роздрукувати форму звітності: список проданих квитків. Публікація інформації про доходи кінотеатру за визначений період.

Опис користувачів і груп користувачів системи.

Програмним забезпеченням, створеним кінотеатром, можуть користуватися як працівники кінотеатру, так і гості. Працівник кінотеатру може забезпечити редагування наявної інформації про наявні фільми, змінити графік роботи кінотеатру, включити до репертуару кінотеатру щойно надійшли фільми; і відвідувач може побачити інформацію про розклад кінотеатру, ціни на квитки, фільми на сьогодні.

Кінокасова стійка – це касова стійка, яка використовує комп’ютер, сумісний з IBM PC. Перед побудовою діаграми потоку контекстних даних - DFD необхідно проаналізувати зовнішні події (зовнішні об'єкти), що впливають на функціонування інформаційно-керуючої системи кінотеатру. Зовнішні об'єкти взаємодіють з IP-адресою шляхом обміну з нею інформацією. З опису предметної області видно, що з програмною оболонкою працюють такі групи людей: касир і відвідувачі. Ці групи є зовнішніми об'єктами. Вони не тільки взаємодіють із системою, але й визначають її межі та представлені на початковій діаграмі контексту потоку даних DFD як термінатори (зовнішні сутності). Початкова контекстна діаграма потоків даних показана на рисунку 2.1. У використовуваних нотаціях зовнішні об’єкти позначаються прямокутниками, а процеси – колами.

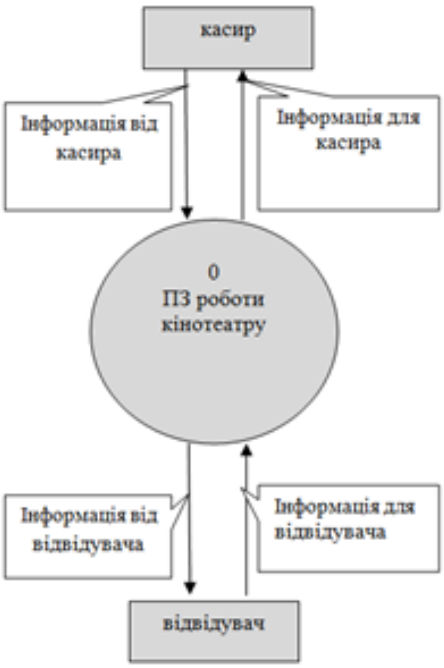

Рисунок 2.1 - Початкова контекстна діаграма потоків даних (DFD) процесу бронювання квитків у кінотеатрі

Список подій структурований у формі матриці списку подій (ELM) і описує різні види діяльності зовнішніх сутностей і реакцію на них IP. Ці дії є зовнішніми подіями, які впливають на систему - симулятор. Розрізняють такі види подій:

- NC (Normal Control) - нормальне управління;
- ND (Normal data) - нормальні дані;
- NCD (Normal Control / Data) - нормальний контроль / дані;
- TC (Temporary Control) - тимчасовий контроль;
- TD (temporary data) - дані часу;
- TCD (Temporary control / data) - тимчасовий контроль / дані.

Усі дії позначені як звичайні дані. Ці дані – це подія, яку IP сприймає безпосередньо, наприклад зміна інформації про особи, яка тестує, яку необхідно негайно записати. Вони відображаються на діаграмі потоку даних DFD як зупинений потік даних. Матриця списку подій (ELM) наведена в таблиці. 2.1. Аналіз функціонального аспекту поведінки системи закінчується побудовою повної контекстної діаграми і представляє діаграму нульового рівня. У цьому випадку процес «управління» розбивається на процеси, що відображають основні взаємодії з

системою. Існуючі «абстрактні» потоки даних між термінаторами та процесами перетворюються в потоки, що представляють обмін даними на більш детальному рівні (табл. 2.2).

Таблиця 2.1 - Матриця списку подій (ELM)

Опис події	Тип	Реакція системи
Перегляд списку реалізованих квитків	ND	Активізація вікна списку реалізованих квитків - дата продажу квитка, на який сеанс, місце, ряд, назву фільму, жанр, вікові обмеження
Перегляд режиму роботи кінотеатру	ND	Активізація вікна режиму роботи кінотеатру - час проведення сеансу, вартість квитка на цей сеанс
Перегляд репертуару кінотеатру на сьогодні	ND	Активізація вікна репертуару кінотеатру на сьогодні - час проведення сеансу, назва фільму, жанр
Ведення архіву кінотеатру	ND	Активізація вікна архіву кінотеатру - дані про усі фільми, наявні в кінотеатрі
Ведення БД квитків	ND	Виведення усіх квитків проданих за сеанс, виведення усіх квитків проданих за день, виведення усіх квитків, коли-небудь проданих в кінотеатрі
Ведення обліку оплати за квитки	ND	Підрахунок прибутку від реалізації квитків за сеанс; підрахунок прибутку від реалізації квитків за день; підрахунок загального прибутку кінотеатру від реалізації усіх квитків.

Таблиця 2.2 - Відповідність потоків даних на діаграмах різних рівнів

Потоки на діаграмі верхнього рівня	Потоки на діаграмі нульового рівня (конкретні)
Інформація від касира	Виведення усіх квитків проданих за сеанс; виведення усіх квитків проданих за день; виведення усіх квитків, коли-небудь проданих в кінотеатрі; підрахунок прибутку від реалізації квитків за сеанс; підрахунок прибутку від реалізації квитків за день; підрахунок загального прибутку кінотеатру від реалізації усіх квитків.
Інфорація для касира	список реалізованих квитків(дата продажу квитка, на який сеанс, місце, ряд, назву фільму, жанр, вікові обмеження); режим роботи кінотеатру(час проведення сеансу, вартість квитка на цей сеанс); репертуар кінотеатру на сьогодні(час проведення сеансу, назва фільму, жанр); архів кінотеатру(дані про усі фільми, наявні в кінотеатрі).
Інформація від відвідувача	Критерії для придбання квитка : фільм, жанр, який сеанс, місце, ряд.
Інформація для відвідувача	Інформація про сеанси (час проведення сеансу, вартість квитка на цей сеанс), інформація про наявні фільми(назва фільму, жанр, вікові обмеження на перегляд цього фільму)

Матриця списку подій показує, які потоки існують на цьому рівні: кожна подія в списку повинна створити потік (подія створює вхідний потік, реакція створює вихідний потік). Один «абстрактний» потік можна розділити на більше ніж один «конкретний» потік.

Повна контекстна діаграма потоків даних показана на рисунку 2.2. У цьому випадку сховище даних «кінематографічне програмне забезпечення» є абстрактним представленням сховища даних.

Аналіз функціонального аспекту поведінки системи дає уявлення про появу та перетворення даних в системі. Зв'язок між «абстрактними» і «конкретними» потоками даних на діаграмі нульового рівня виражається в діаграмах структури даних (рис. 2.3).

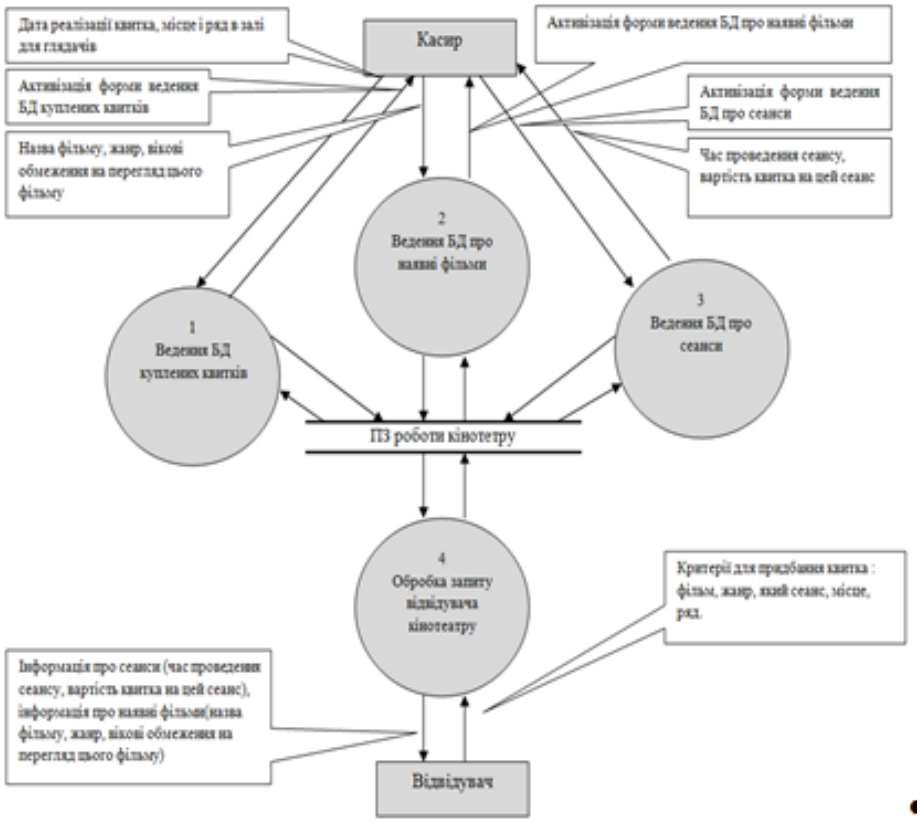

Рисунок 2.2 - Контекстна діаграма потоків даних програмного модуля бронювання квитків у кінотеатрі

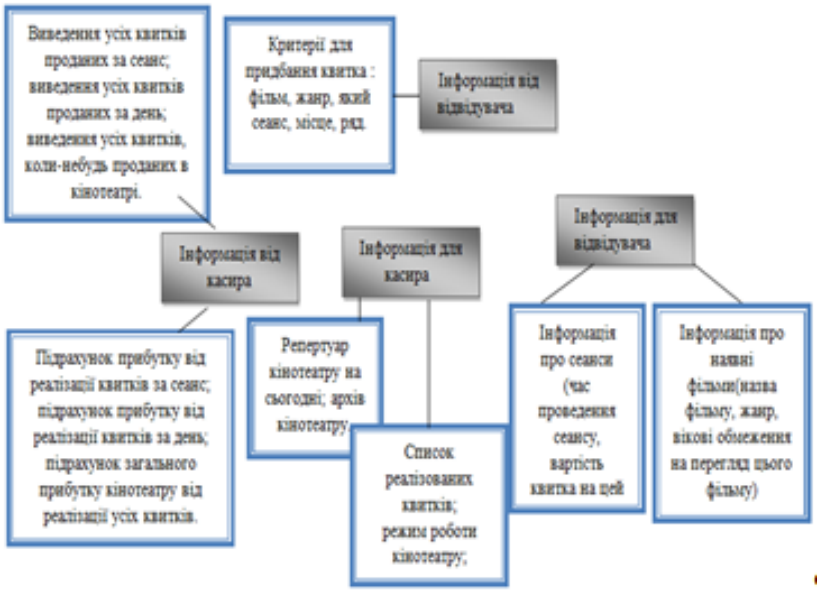

Рисунок 2.3 - Діаграма структур даних програмного модуля бронювання квитків у кінотеатрі

2.2 Розробка UML-діаграм програмного модуля бронювання квитків у кінотеатрі

Візуальне моделювання в UML можна представити як процес переходу від найбільш загальної та абстрактної концептуальної моделі вихідної системи до логічної, а потім фізичної моделі відповідної програмної системи. Для досягнення цих цілей спочатку будується модель у вигляді діаграми варіантів використання, яка описує функціональне призначення системи, іншими словами, що система буде робити під час роботи. Діаграма використання - це початкове концептуальне уявлення або концептуальна модель системи в процесі її проектування та розробки.

Розробка схеми варіантів використання спрямована на:

- визначення загальних меж і контексту змодельованої предметної області на початкових етапах проектування системи;

- сформулювати загальні вимоги до функціональної поведінки проектованої системи;

- розробка вихідної концептуальної моделі системи з метою її подальшої деталізації у вигляді логічної та фізичної моделей;

- підготувати вихідну документацію для взаємодії розробників системи з її клієнтами та користувачами.

Суть діаграми варіантів використання (Use Case Diagram) полягає в наочному представленні функціональних можливостей проєктованої системи через призму взаємодії зовнішніх сутностей з цією системою. У цьому типі UML-діаграм система моделюється як сукупність варіантів використання, які представляють функції, що можуть бути викликані зовнішніми учасниками — акторами. Такий підхід дозволяє не лише окреслити, що повинна робити система, але й визначити, хто саме буде ініціювати ці дії, зокрема користувачі або зовнішні системи.

Актор (actor) у контексті діаграми — це будь-яка зовнішня по відношенню до системи сутність, що ініціює взаємодію з системою для досягнення певної мети. Це може бути людина (наприклад, адміністратор, касир, відвідувач сайту), технічний пристрій (наприклад, сканер квитків, термінал самостійного обслуговування), інша програма або вебсервіс, який інтегрується з проєктованою системою (наприклад, платіжний шлюз або система аналітики). Головна ідея полягає в тому, що актор завжди розглядається як зовнішній по відношенню до системи, навіть якщо насправді він є частиною більшого загального процесу.

У свою чергу, варіант використання (use case) — це опис однієї з функцій або послуг, які система може надати актору. Варіанти використання моделюють поведінку системи у відповідь на дію актора та визначають, які дії система повинна виконати, щоб досягти певного результату, бажаного для користувача. Наприклад, варіантами використання можуть бути: «Зареєструвати користувача», «Забронювати квиток», «Оплатити замовлення», «Переглянути розклад фільмів» тощо.

Важливо зазначити, що діаграма варіантів використання не деталізує, як саме відбувається реалізація цих функцій — вона не містить інформації про внутрішню логіку, алгоритми чи структуру коду. Натомість вона фокусується виключно на інтерфейсі взаємодії між користувачем і системою, що робить її дуже зручною на початкових етапах розробки для збору вимог, проведення аналітики та обговорення архітектури з командою або замовником.

Крім того, UML дозволяє моделювати взаємозв’язки між варіантами використання:

- за допомогою зв’язків «include» — коли одна функція завжди включає іншу (наприклад, процес оплати завжди включає перевірку наявності місця),

- або «extend» — коли додаткова функціональність виконується лише за певних умов (наприклад, «Отримати знижку» може бути розширенням до «Оплатити квиток» при наявності промокоду).

Таким чином, діаграма варіантів використання є ключовим інструментом аналізу вимог, що дозволяє формалізувати очікувану поведінку системи, охопити всі можливі сценарії взаємодії користувачів із системою та надати чітке уявлення про функціональність майбутнього програмного продукту (рис. 2.4).

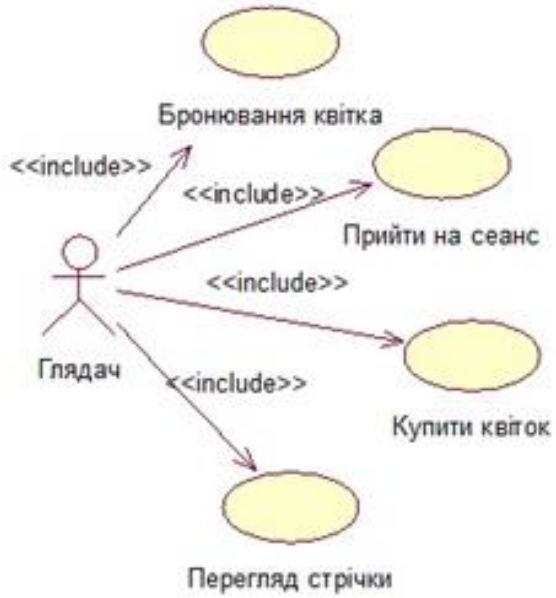

Рисунок 2.4 - Діаграма Use Case програмного модуля бронювання квитків у кінотеатрі

Розробка діаграми діяльності спрямована на:

- деталізація особливостей алгоритмічної та логічної реалізації прецедентів;

- визначити послідовні та паралельні потоки управління;

- готуємо детальну документацію для взаємодії розробників системи з її клієнтами та дизайнерами.

Графічно діаграма діяльності представлена у вигляді графа діяльності, вершинами якого є стани роботи або стан активності, а дуги - переходи від одного стану роботи/діяльності до іншого. Кожна діаграма діяльності повинна мати один початковий стан і один кінцевий стан (на практиці іноді на одному графіку можна побачити багато кінцевих станів, але для кращої читаності графіка один і той же стан відображається кілька разів). Граф активності прийнято розташовувати таким чином, щоб дії відбувалися зверху вниз. У цьому випадку початковий стан буде відображатися у верхній частині графіка, а кінцевий – унизу.

Діяльність, процес — це неатомарний етап обчислення, який займає час у машині. Стани активності можна далі розкласти, а виконану діяльність можна представити іншими діаграмами діяльності. Стани активності неатомарні, тобто можуть бути перериваними. Передбачається, що для їх завершення знадобиться багато часу.

Стан дії - стан, який представляє обчислення атомарної дії, зазвичай виклик операції. Операційні стани не можна розкласти. Вони атомарні, тобто всередині них можуть відбуватися різні події, але роботу, виконану в дії, перервати не можна. Нарешті, тривалість одного стану дії зазвичай вважається непомітно короткою. Дія може полягати в тому, щоб запустити іншу операцію, надіслати сигнал, створити або знищити об’єкт або просто обчислити, скажімо, значення виразу.

Стани активності та запущені стани мають однаковий стандартний графічний символ – прямокутник із заокругленими краями. Всередині такого символу записується будь-який вираз (дія - вираз), який має бути унікальним у межах однієї діаграми діяльності.

Початковий і кінцевий стани на діаграмах діяльності представлені кольоровим колом і кольоровим колом всередині кола відповідно.

Переходи - зв'язок між двома станами, який показує, що об'єкт у першому стані повинен виконати певні дії і перейти в другий стан. Коли дія або діяльність закінчуються в певному стані, потік управління негайно переходить у наступний стан виконання або дії. Для опису цього потоку використовуються переходи, які

показують шлях від одного стану дії або активності до наступного. В UML перехід зображується прямою стрілкою.

Прості переходи послідовності є найбільш поширеними, але самі по собі їх недостатньо для моделювання будь-якого потоку керування. Як і блок-схема, діаграма активності може розгалужуватися або множинні перетини із захисними умовами. Гілка описує різні способи виконання залежно від значення логічного виразу. Графічно точка розгалуження зображується ромбом. Рівно один прохід може входити в точку розгалуження, а два або більше – виходити. Для кожного результуючого переходу дається логічний вираз, який оцінюється лише один раз на вході в точку розгалуження. За відсутності двох результуючих переходів, умови сторожового таймера не повинні виконуватися одночасно, інакше потік управління буде неоднозначним. Але ці умови повинні включати всі можливі варіанти, інакше потік припиниться.

Прості послідовні розгалужені переходи найчастіше використовуються в діаграмах діяльності. Однак часто виникає потреба у візуалізації паралельних потоків, зокрема це стосується моделювання бізнес-процесів. UML використовує часову шкалу для позначення поділу та з’єднання таких паралельних потоків виконання, яка зображується як вертикальна або горизонтальна жирна лінія. У цьому випадку паралельний форк має одне вхідне і кілька вихідних одночасних підключень, навпаки, він має кілька вхідних і одне вихідне.

Кожна діаграма станів в UML описує всі можливі стани одного екземпляра даного класу і можливі послідовності його переходів з одного стану в інший, тобто моделює всі зміни стану об’єкта у відповідь на зовнішні дії.

Діаграми станів (State Machine Diagrams, або Statecharts у UML) є потужним інструментом моделювання, який використовується для відображення динамічної поведінки системи або окремих її об’єктів у відповідь на певні події. Цей тип діаграм особливо корисний тоді, коли необхідно описати, як система змінює свій стан протягом життєвого циклу в залежності від зовнішніх впливів або внутрішніх умов. Хоча найчастіше діаграми станів застосовуються для моделювання поведінки окремих об'єктів класу, їхнє застосування не обмежується лише цим. Вони також

можуть використовуватись для визначення функціональності параметрів використання, взаємодії з акторами (дійовими особами), підсистемами, операціями, а також окремими методами.

Діаграма станів — це особливий тип графічного представлення, який відображає поведінку системи як автомата з кінцевим числом станів. Вершинами графа виступають стани, які об'єкт може мати в певний момент часу, а дуги або стрілки між ними — це переходи, які описують зміну стану під впливом певних подій, умов або дій користувача. Кожен стан може мати вхідні дії (entry), вихідні дії (exit) та внутрішні дії (do), що уточнюють поведінку об’єкта під час перебування у цьому стані.

Наприклад, при моделюванні поведінки квитка в системі бронювання кінотеатру, можна визначити наступні стани:

- «Доступний»;
- «Заброньований»;
- «Оплачений»;
- «Скасований»;
- «Використаний».

Переходи між цими станами можуть відбуватися в результаті дій користувача (наприклад, бронювання або скасування), автоматичних процесів (наприклад, скасування броні після закінчення часу дії), або зовнішніх впливів (наприклад, помилки оплати).

Особливістю діаграм станів у UML є можливість використання вкладених станів (substates) — коли всередині одного стану моделюються більш деталізовані підстани. Це дозволяє зменшити складність діаграми, структурувати поведінку та уникнути дублювання. Також можуть бути застосовані паралельні регіони (orthogonal regions), які дозволяють описати об’єкт, що одночасно перебуває у кількох незалежних станах, що особливо корисно для багатопоточних або реактивних систем.

Діаграми станів широко використовуються:

- у розробці інтерактивних інтерфейсів, де важливо відстежувати, в якому стані перебуває користувач або додаток (наприклад, логін, введення даних, підтвердження);

- в інтегрованих системах управління пристроями, де поведінка змінюється залежно від сенсорних подій;

- при моделюванні складних бізнес-процесів, де кожен крок процесу має залежності від попередніх дій;

- в ігрових системах, де персонажі або об’єкти мають змінні стани поведінки (наприклад, «активний», «очікує дії», «поранений», «переміщується» тощо).

Таким чином, діаграми станів є надзвичайно корисними для аналізу, проєктування та документування логіки переходів між станами в системі. Вони дозволяють зрозуміти, як саме реагує система на події, як змінюється її внутрішній стан у процесі виконання, а також слугують основою для подальшої реалізації програмного забезпечення відповідно до визначених сценаріїв поведінки.

У мета-моделі UML автомат — це пакет, який визначає багато концепцій, необхідних для представлення поведінки змодельованого об’єкта у вигляді дискретного простору зі скінченною кількістю станів і переходів.

Тривалість системи в будь-якому з можливих станів значно перевищує час, затрачений на перехід з одного стану в інший. Передбачається, що під час переходу він може бути нульовим (якщо не вказано інше), тобто стан об’єкта може змінитися негайно.

При моделюванні ходу діаграм діяльності іноді буває корисно розділити статус діяльності на діаграмах на групи, кожна з яких представляє відділ компанії, відповідальний за дану посаду. В UML такі групи називаються доріжками для плавання, оскільки візуально кожна група відокремлена від сусідніх вертикальною лінією, як доріжки для плавання в басейні. Доріжки — це свого роду пакет, який описує пов’язаний набір робіт (рис. 2.5).

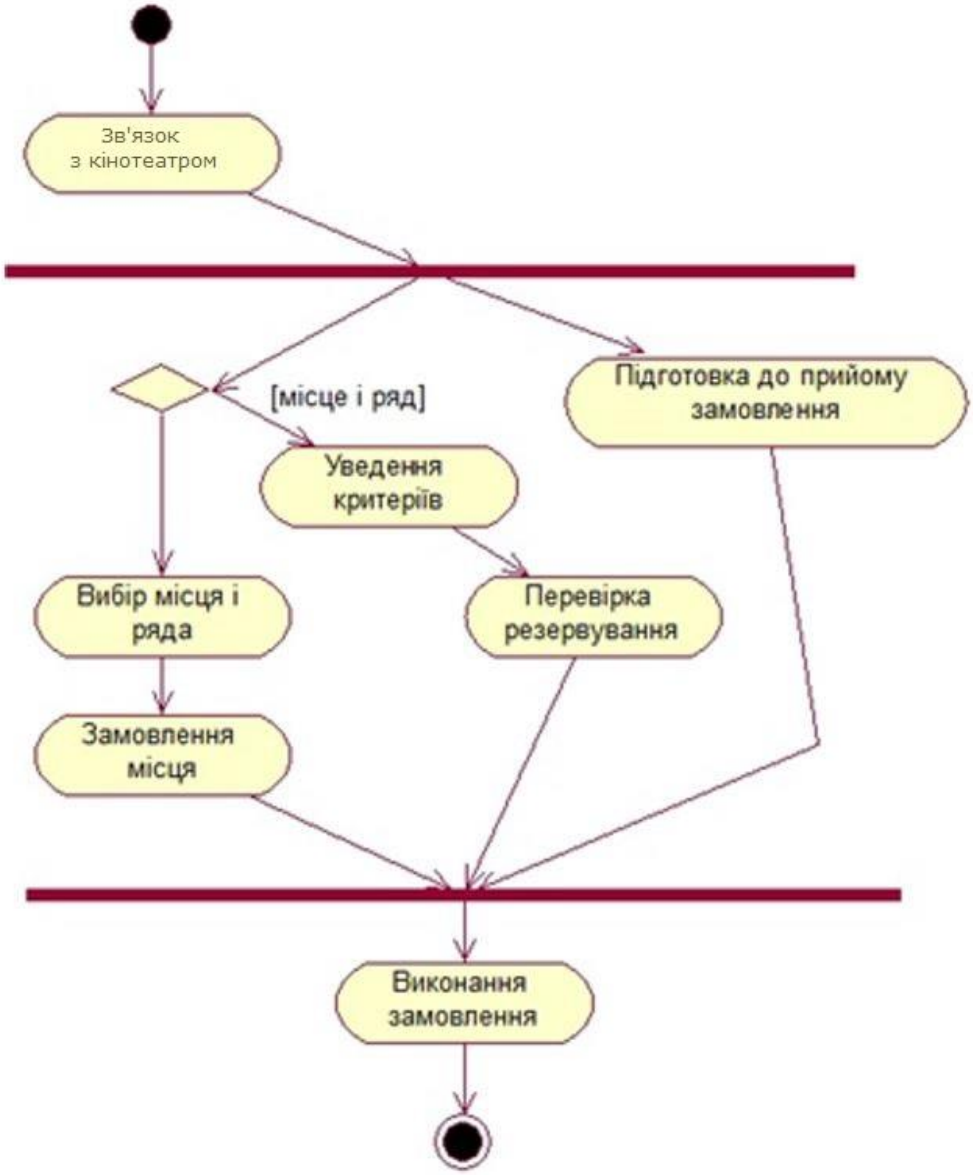

Рисунок 2.5 - Діаграма діяльності процесу бронювання квитків у кінотеатрі

Поведінка автомата в контексті діаграми станів моделюється як послідовне переміщення об'єкта або системи по вершинах графа, що представляють окремі стани, з урахуванням напрямку орієнтованих дуг, які відображають переходи між станами. Ці переходи активуються у відповідь на зовнішні або внутрішні події, що змінюють стан системи відповідно до певної логіки або умов.

Іншими словами, система поводиться як автомат з кінцевим числом станів (Finite State Machine), у якому в кожен конкретний момент часу об’єкт перебуває лише в одному визначеному стані, і тільки після виникнення певної події або виконання умови може перейти до іншого, наступного стану. Це забезпечує

структуровану, послідовну та передбачувану поведінку системи, де кожен перехід логічно обґрунтований і залежить від попереднього стану.

Кожна дуга (орієнтований перехід) має, як правило, назву події або умови, яка викликає зміну стану, а також може містити дії, що виконуються під час переходу. Наприклад, у моделі процесу бронювання квитка перехід зі стану «Забронbований» у стан «Оплачений» може відбутися після події «Успішна оплата». При цьому додатково може бути виконана дія — «Згенерувати QR-код квитка».

Дотримання орієнтації дуг (тобто напрямку від одного стану до іншого) є критично важливим, оскільки саме це визначає логіку проходження процесу: неможливо повернутись назад без визначеного зворотного переходу, що дозволяє контролювати допустимі шляхи та виключити некоректні сценарії поведінки (рис. 2.6).

Рисунок 2.6 - Діаграма станів процесу бронювання квитків у кінотеатрі

Для представлення статичної структури моделі системи в термінології класів об’єктно-орієнтованого програмування використовується діаграма класів. Зокрема,

діаграма класів може відображати різні відносини між різними суб’єктами предметної області, такими як об’єкти та підсистеми, і описувати їх внутрішню структуру та типи зв’язків. Ця діаграма не надає інформації про тимчасові аспекти функціонування системи. З цієї точки зору діаграма класів є подальшим розвитком концептуальної моделі системи, що проектується.

Діаграма класів (Class Diagram) є одним із ключових інструментів візуального моделювання в мові UML (Unified Modeling Language) та широко використовується для опису статичної структури об'єктно-орієнтованої системи. Вона являє собою графічне зображення елементів логічної моделі, де вершини графа відповідають класифікаторам (насамперед — класам, інтерфейсам, абстрактним класам), а ребра — це структурні відношення між ними, такі як асоціація, агрегація, композиція, успадкування (генералізація), залежність тощо.

Попри свою назву, діаграма класів охоплює не лише класи. Вона може також містити:

- інтерфейси, що визначають контракт (набір операцій), який можуть реалізувати класи;

- пакети — логічні блоки для організації моделей у більш загальні структури;

- об’єкти — конкретні екземпляри класів, що можуть бути зображені для ілюстрації взаємодій;

- зв’язки між екземплярами (лінки) — які демонструють реальні відношення між об’єктами на прикладі.

Говорячи про діаграму класів, зазвичай мають на увазі статичну модель, яка відображає структуру системи на концептуальному або логічному рівні, незалежно від її динамічної поведінки або часу виконання. Це означає, що діаграма класів описує елементи системи, які залишаються незмінними впродовж часу: типи сутностей, їхні атрибути, операції, а також сталі структурні зв’язки між ними.

Основні елементи діаграми класів включають:

- Класи — представлені прямокутниками, розділеними на три секції: ім’я класу, список атрибутів і список методів (операцій). Класи відображають типові об’єкти предметної області;

- Інтерфейси — позначають абстрактні типи, які описують лише сигнатури методів, без реалізації. Класи можуть реалізовувати інтерфейси (зазначається зв’язком «realize»);

- Асоціації — зв’язки між класами, які показують, як об’єкти одного класу «знають» або «використовують» об’єкти іншого. Асоціації можуть мати напрямок, ролі, множинність та іменування;

- Успадкування (генералізація) — показує ієрархію, де підклас успадковує атрибути та методи батьківського класу;

- Агрегація та композиція — представляють «частина-ціле» відношення. Композиція вказує на сильну залежність життєвого циклу частини від цілого;

- Залежності — відображають ситуації, в яких зміна одного елемента (наприклад, інтерфейсу) може вплинути на інший (клас, що його реалізує);

- Пакети — дозволяють групувати класи та інші елементи діаграми в логічні блоки, забезпечуючи краще структурування великих моделей. Якщо діаграма класів належить до певного пакета, усі її елементи мають відповідати його межам і правилам, а також можуть посилатися на елементи з інших пакетів за допомогою кваліфікованих імен або імпорту.

Таким чином, діаграма класів виконує роль схеми або креслення архітектури системи, де кожен клас — це «будівельний блок», а відношення між ними — «сполучення» або «взаємодія». Вона дозволяє:

- побачити структуру ієрархії класів;

- виявити зв’язки між об'єктами;

- оцінити залежності між частинами системи;

- забезпечити основу для реалізації програмного коду;

- полегшити обговорення моделі з аналітиками, замовниками та розробниками.

На рисунку 2.7 може бути представлена діаграма класів, яка відображає фрагмент моделі програмного модуля бронювання квитків: наприклад, класи Користувач, Квиток, Сеанс, Фільм, а також інтерфейси СистемаОплати або БазаДаних, що реалізуються відповідними підсистемами.

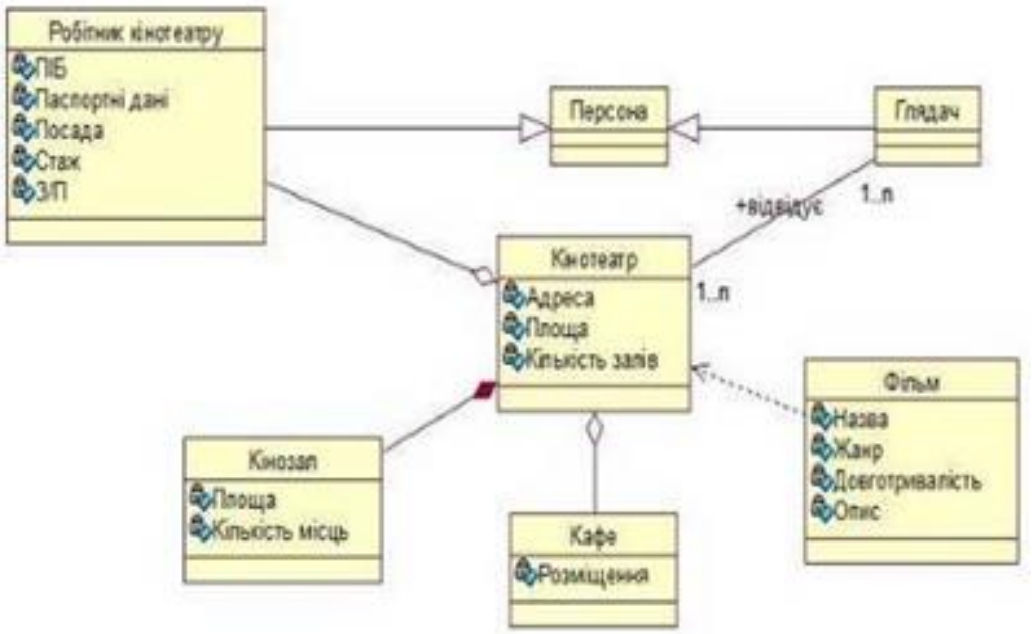

Рисунок 2.7 - Діаграма класів процесу бронювання квитків у кінотеатрі

2.3 Висновок до розділу 2

У даному розділі було розроблено UML-діаграми програмного модуля бронювання квитків у кінотеатрі. Побудовано інформаційну модель основних модулів програмного забезпечення, що складається із вхідних даних, даних для обробки, методів обробки, вихідних даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРІ

3.1 Обґрунтування середовища і мови програмування

Середовище розробки Android Studio [5] - інтегроване середовище розробки (IDE) для платформи Android. Це офіційне середовище розробки Android-програм від компанії Google для відповідної операційної системи, яке замінило плаґін ADT для середовища Eclipse і є зараз одним із найпопулярніших інструментів серед розробників. Версія Android Studio v.3.4 була випущена 17 квітня 2019 року. Дане середовище є абсолютно безкоштовним для завантаження й користування. Тут наявні макети для створення користувацького інтерфейсу, завдяки чому воно переважно і використовується при розробці застосунків для Android. У цьому середовищі є всі інструменти не тільки для розробки програм для планшетів, смартфонів, персональних комп’ютерів, а й для новіших рішень таких, як розумні телевізори (Android TV), годинники (Android Wear), автомобілі (Android Auto) та інші додаткові модулі. Розробки у середовищі Android Studio є дуже гнучкими. Це реалізовано за допомогою відображення всіх робочих файлів в одній програмі в одному вікні. Корисною функцією є можливість побачити всі візуальні зміни розроблюваної програми в режимі реального часу [7].

Середовище Android Studio надає можливість протестувати свій проект на різних пристроях як підключаючи смартфон, так і запустивши віртуальну машину прямо із середовища розробки. У вбудованому емуляторі Android-пристроїв можна встановити різні технічні конфігурації, версії Android і роздільну здатність екрану. Також можна отримати інформацію про рівень приблизної продуктивності для конкретного пристрою. Середовище розробки адаптоване для виконання завдань, які вирішуються у процесі розробки Android-додатків. У середовище включені засоби для спрощення розробки й тестування програм. Можна перевірити проект на сумісність з різноманітними пристроями такими, як смартфони, планшети, персональні комп’ютери, розумні годинники та інші.

Перевагами даного середовища розробки є [7]:

- підтримка багатьох мов програмування таких, як Java, C, C++, Kotlin [6];

-- доступність всіх функцій Java 7, 8 і багатьох функцій новіших релізів 9-12 [7, 8];

- зручне редагування коду;

- багатофункціональна консоль розробника;

- постійне оновлення, отже, можливість розробки для найсвіжіших версій OS Android.

Переглянувши системні вимоги, можна зробити висновок, що для розробки мобільного додатку необхідно використати потужний персональний комп'ютер. На сьогодні більшість проектів для Android створюється за допомогою середовища Android Studio. Оскільки середовище надає різноманітний і широкий функціонал, підтримує популярні мови програмування такі, як Java, C/C++ тощо, то це середовище є дуже популярним як серед новачків, так і серед професіоналів. Середовище Android Studio по праву вважається найкращим безкоштовним інтегрованим середовищем розробки для Android-програмістів. Також є ще середовище IntelliJ IDEA, але воно в основному використовується для проектів не тільки для Android.

Мова Java — об'єктно-орієнтована мова програмування [10]. Саме тому для неї притаманні три парадигми об'єктно-орієнтованого програмування:

- наслідування;

- інкапсуляція;

- поліморфізм.

Мова програмування Java була розроблена компанією Sun Microsystems. Програми, написані мовою Java, переважно компілюються в спеціальний байт-код, тому вони можуть працювати на будь-якій віртуальній Java-машині (JVM) незалежно від комп'ютерної архітектури. Дата офіційного релізу - 1995 рік. Сьогодні технологія Java надає можливості для перетворення сталих Web сторінок в динамічні інтерактивні документи, а також для створення окремих додатків, що не є залежними від платформи. Мова Java запозичила синтаксис із C і C++ [11]. Не

пов’язуючи розробку з конкретною платформою, розробник мови Java Ґослінг почав з розширення компілятора C++. Згодом він зрозумів, що мова C++, як її не розширюй, не зможе задовільнити всі потреби. Тому в середині 1991 року було розроблено мову Oak (згодом при пошуку торгової марки її назва була замінена на Java) [12]. Гослінг вважав браузер важливим компонентом “створення ринку” для додатків, серверів і середовищ розробки. У цих сервісах мова Java відіграє ключову роль. До появи Java веб-сторінка фактично представляла собою листок паперу. З появою Java браузер задає структуру і значно розширює можливості.

Програмний модуль бронювання квитків у кінотеатрі був реалізований під операційну систему Android. Вибір зупинився на цій операційній системі, оскільки за останніми даними близько 88% усіх смартфонів у світі працюють під ОС Android [13]. Операційна система Android [14] - це розробка компанії Android inc. Зараз вона є власністю компанії Google і використовується на багатьох сучасних пристроях, таких як смартфони, планшети, персональні комп’ютери, телевізори, годинники, автомобільні бортові комп’ютери тощо. Операційна система Android побудована на зміненому ядрі Linux і власній реалізації віртуальної машини Java [15]. Система Android дає можливість створювати Java-застосунки, які керують пристроєм через розроблені Google-бібліотеки. Операційна система Android працює на віртуальній машині JVM (Java Virtual Machine), яка здійснює виконання команд байт-кодів. Байт-код - це проміжне рішення, в яке може бути перекладено код програми. Відносно програмного коду, що є читабельним і зручним для користувача (програміста), байт-код — це компактне подання програми, яка є простішою для комп’ютера. Це подання проходить синтаксичний і семантичний аналізи. У ньому в явному вигляді закодовані типи, області видимості тощо. З технічного боку, байткод є машинно-незалежним кодом, що генерується транслятором і є кодом низького рівня. Крім використання мови Java, існує можливість створення додатків з використанням мови C++, проте сама компанія Google не рекомендує використовувати її, а лише в певних випадках, наприклад, якщо потрібне низькорівневе налаштування деяких апаратних складових пристрою. Переваги ОС Android:

- система Android зарекомендувала себе краще за своїх конкурентів, таких як IOS, Windows Phone mobile, щодо веб-серфінгу, інтеграції з сервісами Google;

- є відкритою платформою, що дає можливість реалізовувати на ній більше функцій;

- в Android-пристроях, як правило, присутній читач карт пам’яті, що робить можливим швидке перенесення файлів з комп’ютера на телефон і навпаки;

- повноцінна реалізація Bluetooth-стека, що забезпечує в тому числі передачу і прийом файлів;

- можливість встановлення додатків зі сторонніх джерел, що дає можливість не використовувати Інтернет-мережу й тестувати додатки;

- багатокористувацький режим.

Отже, для програмної реалізації модуля бронювання квитків у кінотеатрі було обрано саме ОС Android, як систему, що має відкритий тип, багатий функціонал і є найбільш поширеною.

3.2 Опис бібліотек і компонентів

Програмний модуль бронювання квитків у кінотеатрі реалізовано на основі використання і модифікації існуючих і розроблених класів і методів мови програмування Java. 4.1.

В архітектуру програмного забезпечення бронювання квитків у кінотеатрі входять такі складові:

- застосунок, встановлений на смартфоні з ОС Android;

- база даних, яка розміщується на хмарному сховищі.

Доступ до застосунку мають усі користувачі, в тому чисті адміністратор.

Доступ до бази даних має тільки адміністратор.

Додаток, встановлений на смартфоні, може передати користувачеві такі дані:

- інформацію про фільм;

- інформацію про сеанси;

- інформацію про вільні й заброньовані місця;

- повідомлення про те, що квиток заброньовано чи куплено.

У базі даних зберігається всі інформація, яка надається користувачеві програмного забезпечення. Також через базу даних вносяться зміни про деталі фільму, сеансів тощо. Адміністратор може переглядати, які саме місця були заброньовані та зняти броню з них.

Розроблене програмне забезпечення має у своєму складі ряд класів.

Головний клас MainActivity є звичайним класом мови програмування Java, на початку якого йдуть визначення пакета й імпорту зовнішніх пакетів. За замовчуванням він містить тільки один метод onCreate, в якому фактично і створюється весь інтерфейс програми. Клас MainActivity (рис. 3.1), крім методу onCreate, містить ще методи onStart, createRecycler, onFilmPressed, onLongPressed.

Клас MainPresenter містить метод LoadData, який завантажує з бази даних всю інформацію про фільми (рис. 3.2).

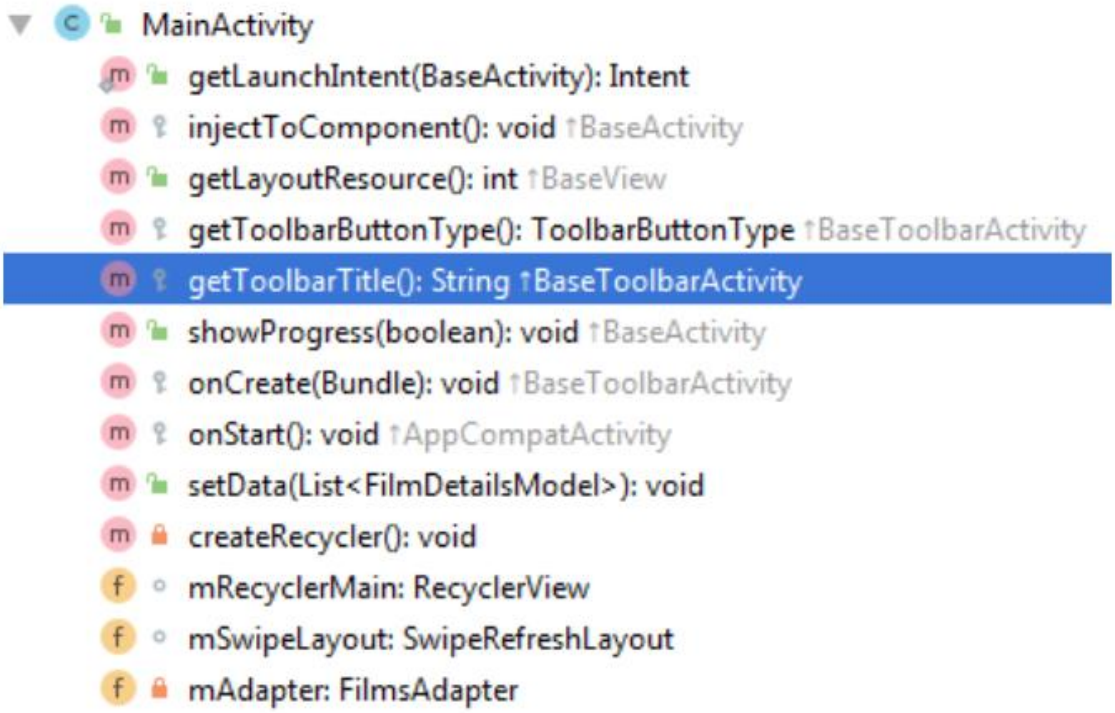

Рисунок 3.1 - Методи класу MainActivity

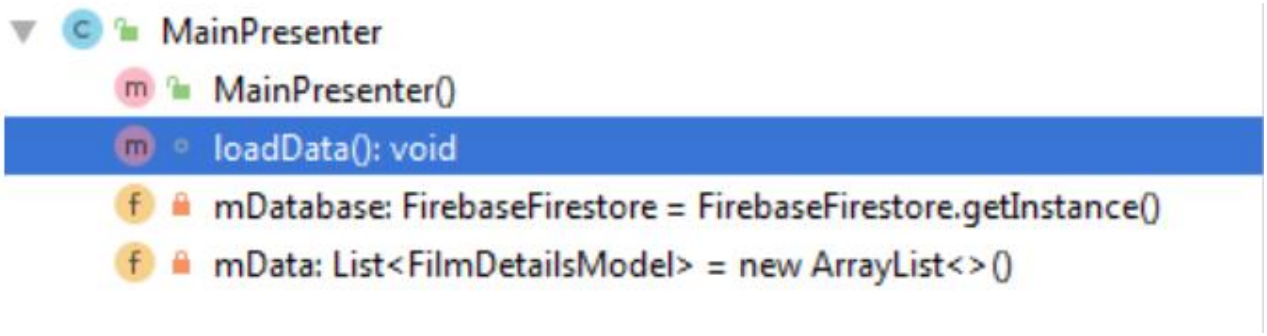

Рисунок 3.2 - Методи класу MainPresenter

Програмне забезпечення бронювання квитків у кінотеатр побудовано за архітектурою MVP (Model, View, Presenter). Тому в програмі є класи моделей, виглядів та презентерів. Розглянемо класи моделей. Клас FilmDetailModel (рис. 3.3) - це клас, що створює модель фільму, тут є такі методи, як FilmDetailsModel, getId, getName, setName, getDescription, setDescription та інші. Ці методи відповідають за подачу інформації про фільм користувачеві програми і передачу інформації про фільм від адміністратора до бази даних.

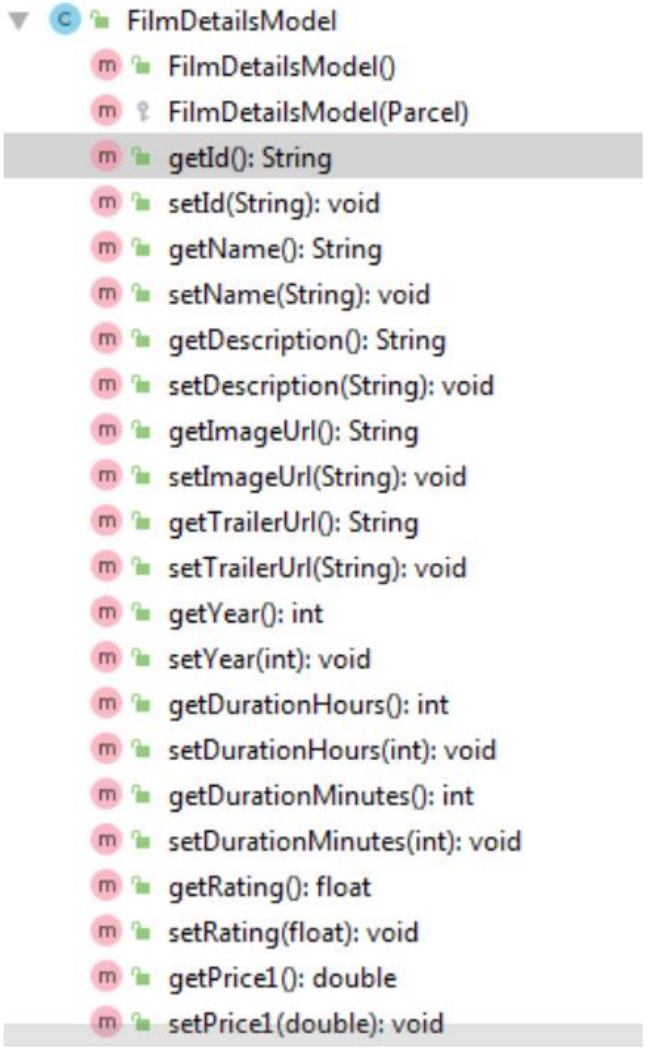

Рисунок 3.3 - Методи класу FilmDetailModel

Клас SeanseModel (рис. 3.4) - тут є всі методи для передачі інформації глядачеві та зчитування даних від адміністратора про сеанси фільму.

Клас SeansesListModel (рис. 3.5) — клас, в якому є методи для виводу списку сеансів.

Клас SeatModel (рис. 3.6) - це клас, в якому є методи для відображення кінозалу, тобто вигляду місць, перевірка, які є заброньовані, а які вільні, розподілу місць за ціновими категоріями.

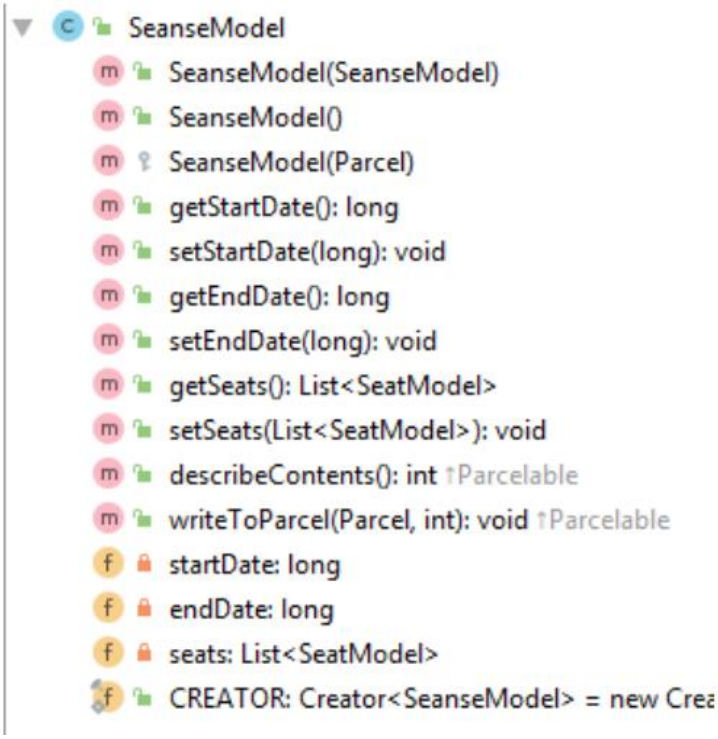

Рисунок 3.4 - Методи класу SeanseModel

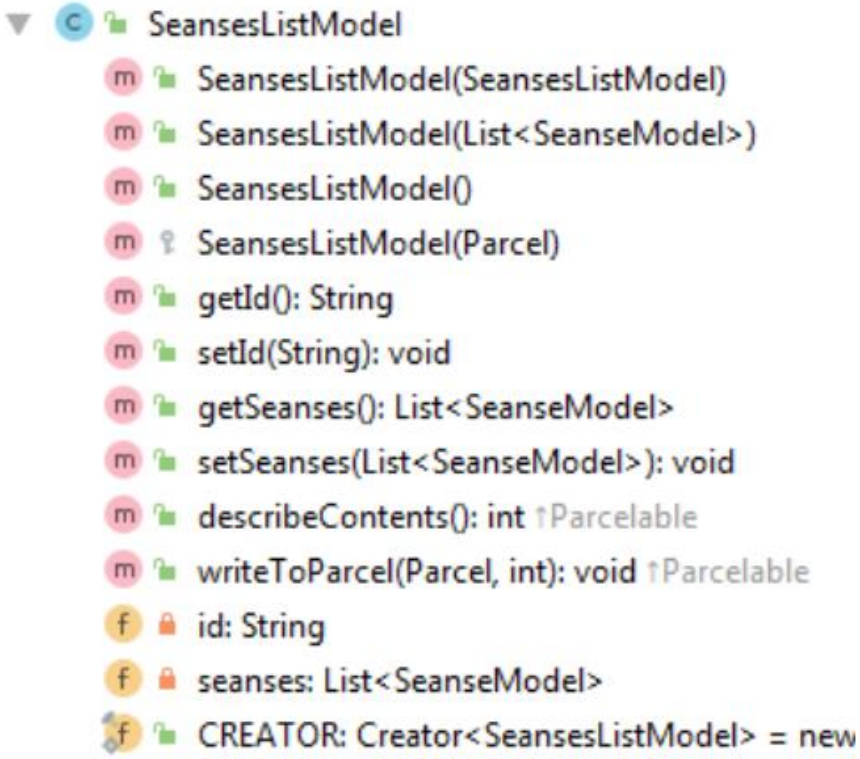

Рисунок 3.5 - Методи класу SeansesListModel

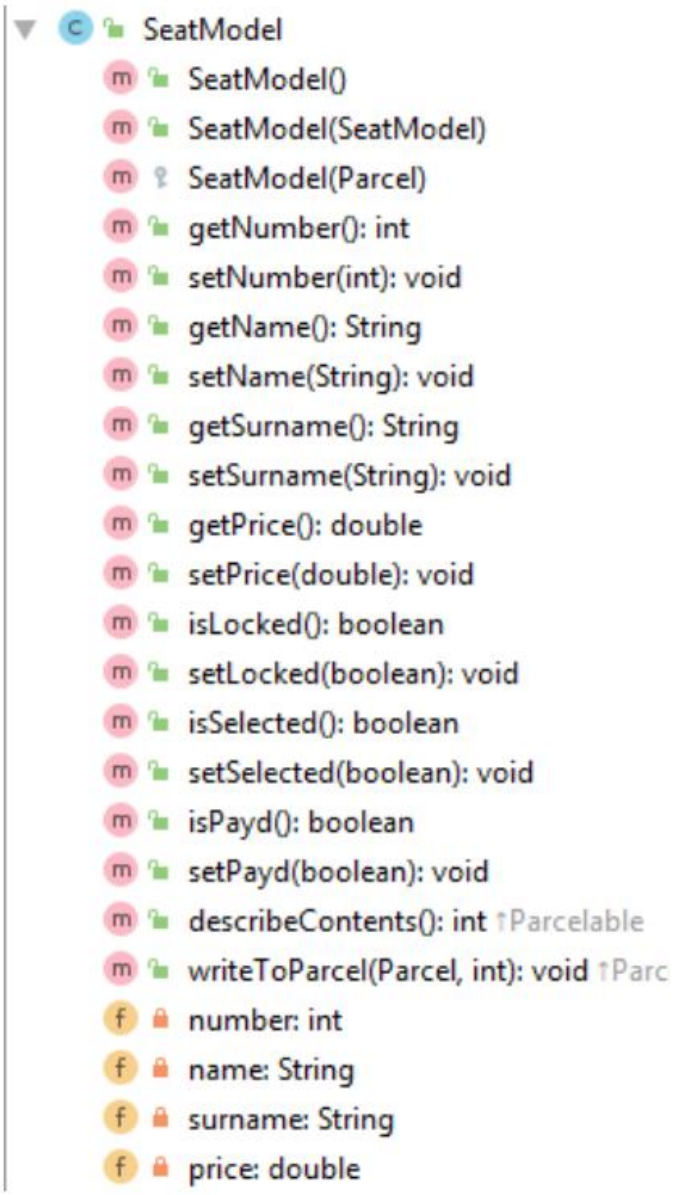

Рисунок 3.6 - Методи класу SeatModel

Таким чином, розроблені класи моделей дають можливість реалізувати роботу програмного забезпечення замовлення квитків у кінотеатр щодо надання користувачеві різноманітної інформації.

Генератор фільмів, сеансів і місць, як і вся програма, побудований за архітектурою MVP.

Моделлю в даному випадку виступає клас GeneratorActivity (рис. 3.7). Цей клас створено виключно для взаємодії “адміністратор-система”. Методи, що використовуються в ньому, описують всі поля, потрібні для створення сторінок з

фільмами, сеансами тощо. Генератор запитує в адміністратора з додатку чи зчитує інформацію з бази даних таку, як:

- всю інформацію про фільм (назва, опис, картинка, трейлер тощо);
- дату і час сеансу;
- ціни.

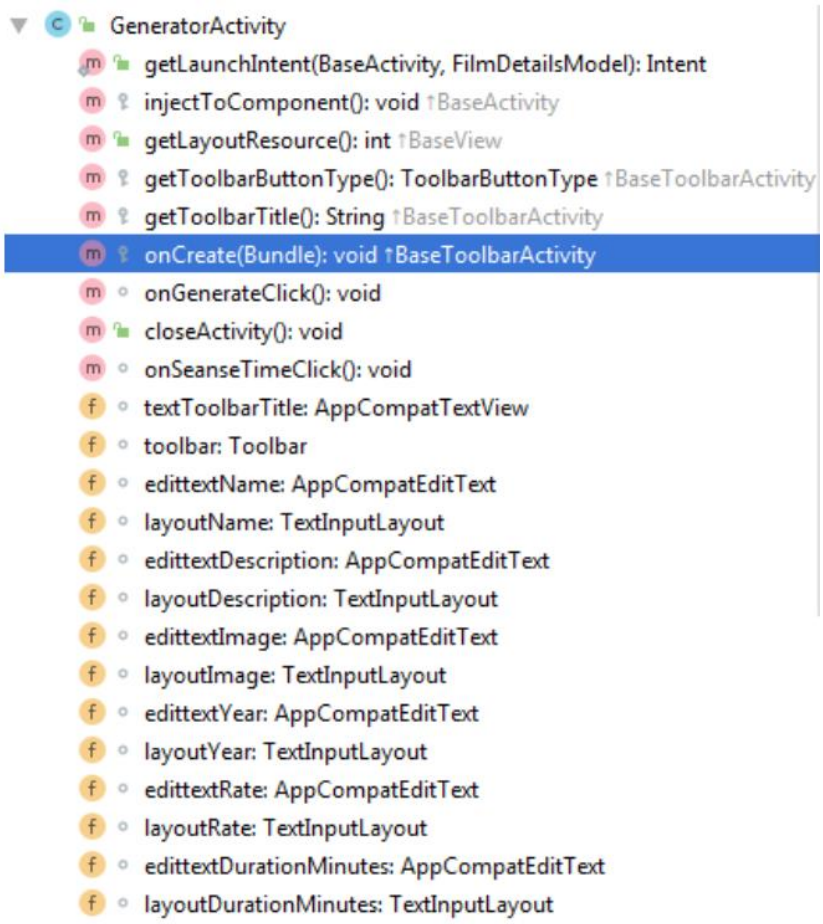

Рисунок 3.7 - Методи класу GeneratorActivity

Презентер генератора (клас GeneratorPresenter (рис. 3.8) звертається до класу GeneratorActivity та обробляє дані. Він задає модель фільму. Якщо її немає, то створюється новий. Якщо є, модель оновлюється.

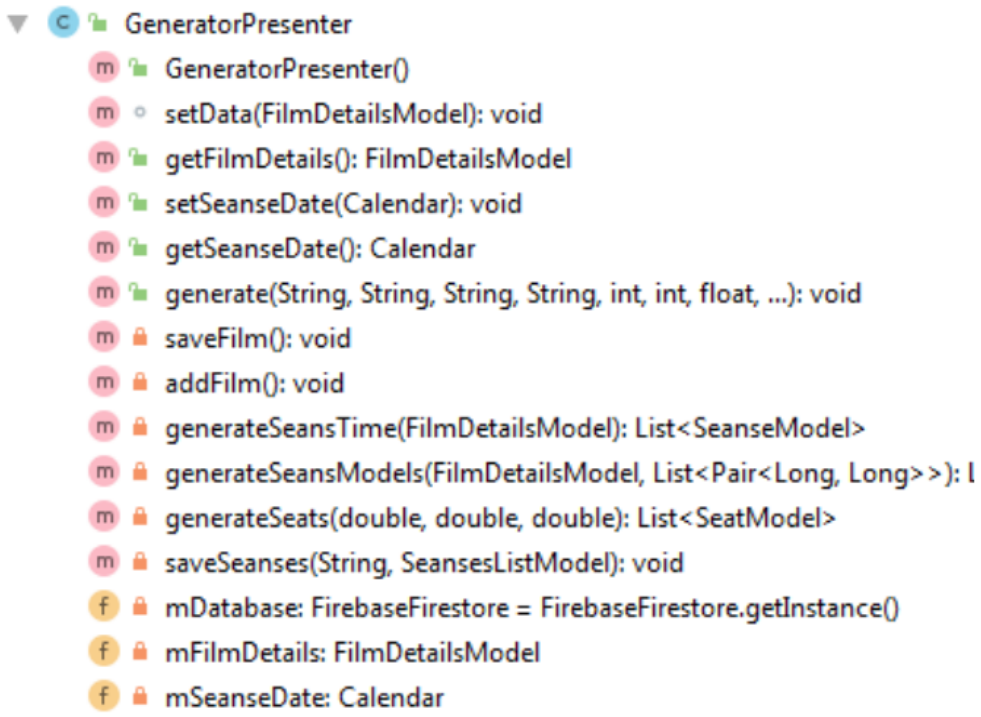

Рисунок 3.8 - Методи класу GeneratorPresenter

Після зчитування й перевірки всіх даних презентер генерує дані про фільм і зберігає їх на сервері. Якщо дані про фільм вже існують, то спрацьовує метод saveFilm, якщо ж ні - метод addFilm. Також після зчитування даних про тривалість фільму, генератор створює сеанси і розбиває їх згідно з тривалістю. Потім зчитує дані про ціни й генерує місця та записує їх на сервер. Вся інформація зберігається на хмарному сервісі FireBase. Для адміністратора (менеджера) системи надаються розширені права доступу, які включають можливість створення нових записів у базі даних, редагування вже наявних даних, а також їхнє видалення або оновлення. Це дозволяє оперативно реагувати на зміни у структурі інформації — наприклад, додавати нові фільми до репертуару, змінювати розклад сеансів, оновлювати інформацію про зали, акції чи спеціальні пропозиції.

У даному проєкті використовується база даних Firebase, яка принципово відрізняється від класичних реляційних СУБД, таких як MySQL. Замість таблиць із фіксованою структурою дані в Firebase організовані у вигляді вкладеного дерева — ієрархічної структури, що дає змогу більш гнучко і швидко працювати з великими обсягами динамічної інформації. На рисунку 3.9 наведено приклад такої структури.

Адміністратор має змогу додавати нові колекції (наприклад, "movies", "sessions", "tickets") або редагувати вже існуючі, змінюючи вміст окремих документів. Завдяки такій моделі зберігання даних забезпечується висока масштабованість та адаптивність системи — особливо важлива у випадках, коли обсяг і структура даних можуть змінюватися в режимі реального часу. Крім того, Firebase дозволяє легко реалізувати контроль доступу на рівні користувачів завдяки вбудованим правилам безпеки.

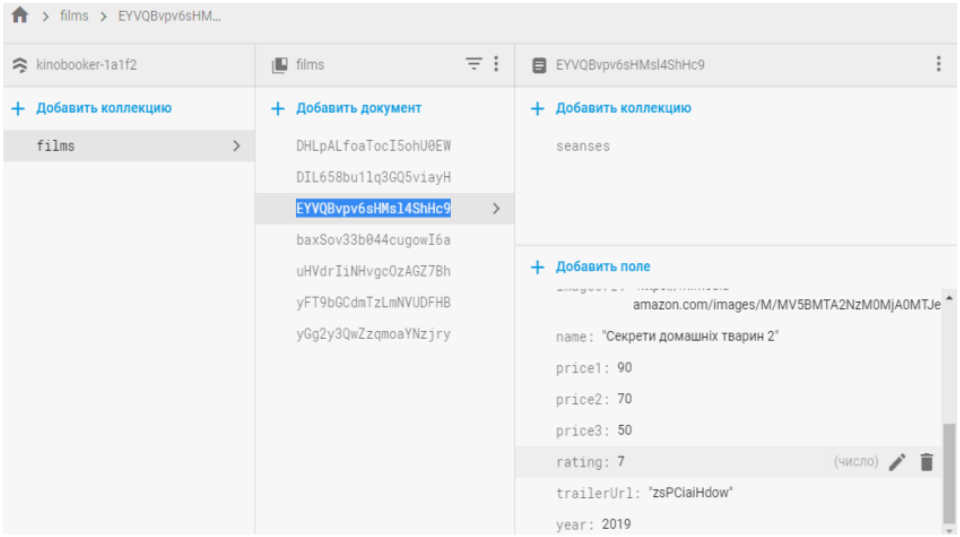

Рисунок 3.9 - Вигляд дерева колекцій на сервісі Firebase

Тут адміністратор може заповнити чи змінити основні дані про фільм та сеанси такі, як:

- поле description - опис фільму;

- поле duratioHours - тривалість фільму (повних годин);

- поле durationMinutes - залишкова тривалість;

- поле d - ідентифікатор фільму. Якщо залишити пустим (null), система створить його сама;

- поле imageUrl - посилання на постер до фільму;

- поле name - назва фільму;

- поля price1, price2, price3, … - цінова категорія місць;

- поле rating - оцінка imdb;

- поле trailerUrl - посилання на трейлер до фільму;

- поле year - рік виробництва.

Дані про бронювання чи купівлю квитка надходять у колекцію Seanses. У колекції Seanses адміністратор може побачити, яке місце забронюване, а яке вільне. Якщо воно забронюване, то значення атрибута “locked” дорівнює true, якщо вільне — false. Також тут міститься інформація про номер місця і вартість квитка. Кожне місце в кінозал на кожен сеанс і на кожен фільм має своє окреме місце у дереві бази даних. Воно має кілька атрибутів, наприклад: ідентифікатор (id), ціну (price), порядковий номер тощо. Як було сказано вище, щоб показати користувачеві, що місце зайняте, значення атрибута “locked” змінюється на true. А, щоб показати адміністраторові, що воно ще й оплачене, значення атрибута “payd” змінюється теж на true.

Загальну схему аналізу наявності вільного місця в залі кінотеатру і його відображення подано на рисунку 3.10.

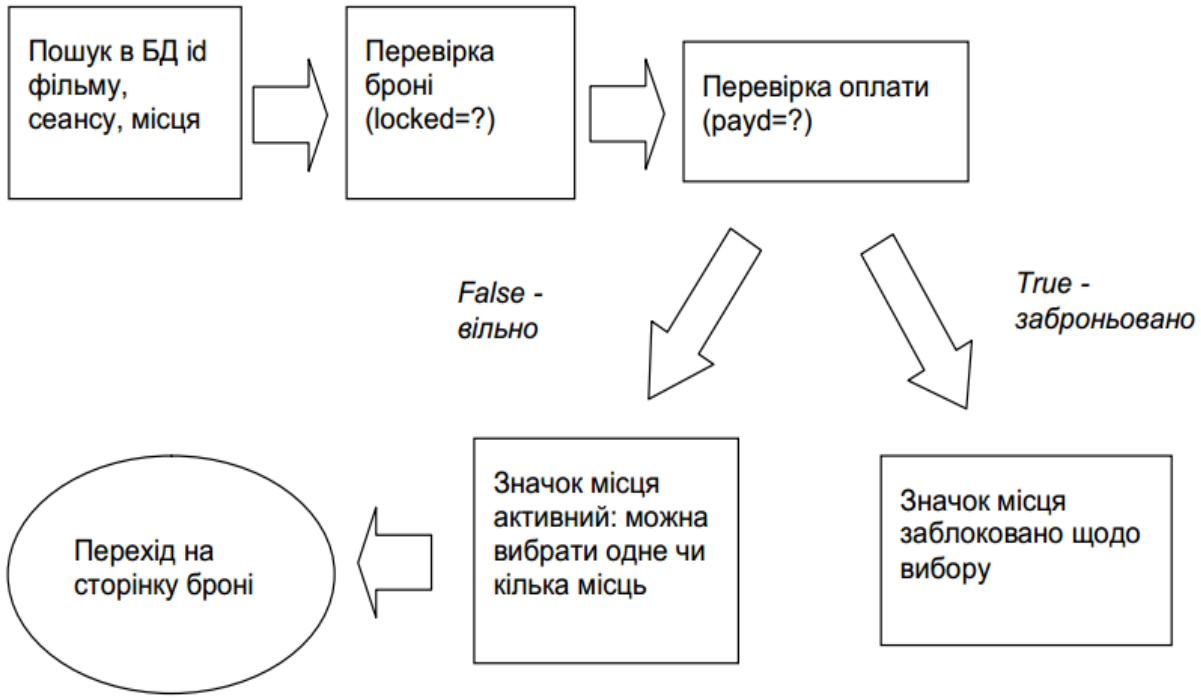

Рисунок 3.10 - Аналіз наявності вільного місця в кінотеатрі

Подання бази даних у формі дерева спрощує роботу адміністратора при додаванні чи зміні даних про фільм. За допомогою адмін-панелі сервісу Firebase

можна надати доступ до бази даних будь-якому користувачеві з правами власника, редактора чи глядача. Також можна додати права на окрему колекцію чи документ, що значно спрощує роботу адміністратора.

3.3 Тестування програмного модуля бронювання квитків у кінотеатрі та аналіз результатів його роботи

При запуску програми з’являється “екран привітання” (рис. 3.11). Це звичайна анімація, яка створюється класом Splash і використовується для того, щоб завантажити список фільмів, інформація про які зберігається в базі даних.

Рисунок 3.11 - Екран привітання

Завантажений список фільмів користувач побачить на екрані і зможе вибрати певний фільм (рис. 3.12).

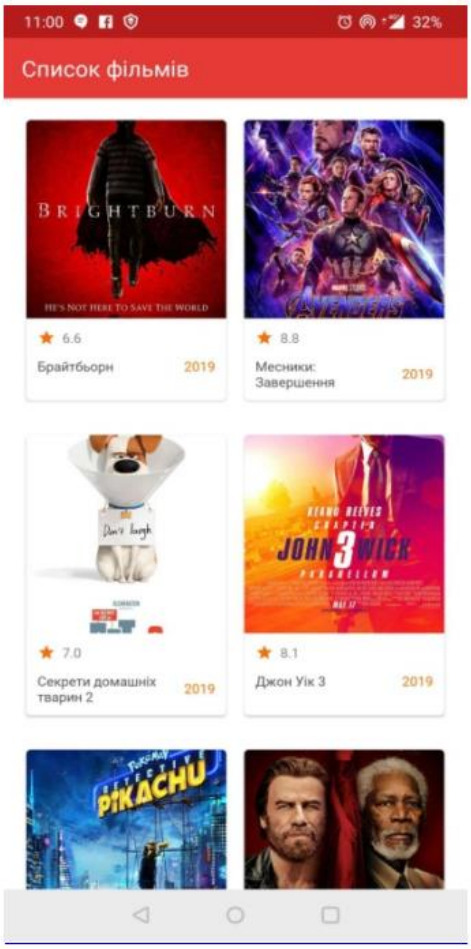

Рисунок 3.12 - Список фільмів

Після вибору фільму користувач потрапляє на його сторінку з постером, оцінкою, описом, тривалістю та кнопкою “замовити квитки”.

Після натиснення на сторінці фільму кнопки “замовити квитки” користувач потрапляє в меню сеансів, де вказано дату, час початку і час закінчення фільму, ціни квитків.

Після вибору вподобаного сеансу користувач потрапляє на сторінку кінозалу, де можна побачити всі місця за ціновою категорію, а також, які місця вільні, а які - заброньовані. Заброньовані місця є неактивними для натискання й позначені замком (рис. 3.13 - 3.14)

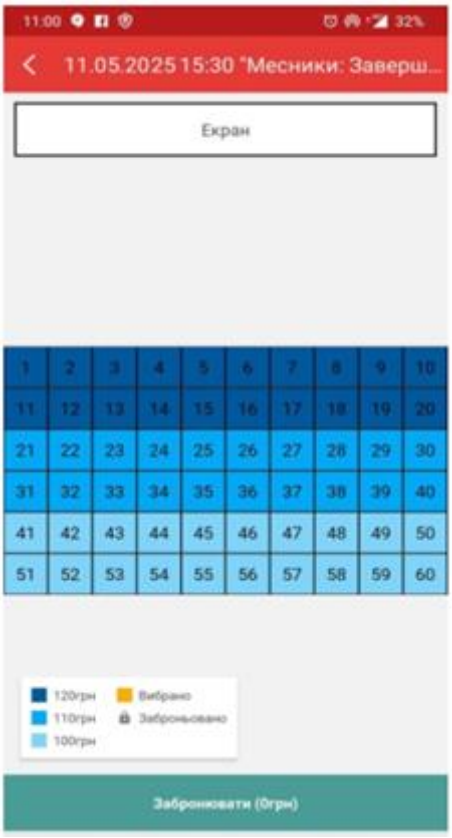

Рисунок 3.13 - Вільні місця (виділено кольором за ціновими категоріями)

Рисунок 3.14 - Заброньовані місця (позначено замком)

Вибравши відповідне місце, користувач може забронювати чи одразу й оплатити його. При бронюванні чи оплаті потрібно ввести свої контактні дані, система перевірить, чи не залишилися поля порожніми. При оплаті система проводить валідацію банківської карти. Для тестування додано валідність для карти 4242 4242 4242 4242 (рис. 3.15).

Рисунок 3.15 - Меню оплати квитка

Після резервування чи оплати користувач отримає сповіщення про успішну операцію, а дані будуть занесені в базу даних.

Для тестування розробленого програмного забезпечення було обрано десять IT спеціалістів. Оцінка роботи виставлялась від 1 до 10, в залежності від того чи дана програма була корисна у порівнянні з програмами аналогами. Результати тестування наведені в табл. 3.1

Таблиця 3.1 – Результати тестування програм бронювання квитків у кінотеатрі

Спеціаліст	1	2	3	4	5	6	7	8	9	10
Розроблене ПЗ	9	10	8	8	9	10	8	9	8	9
Kinoafisha.ua	9	9	8	7	7	10	7	8	9	8
Planeta-kino.ua	9	9	8	9	7	10	7	8	8	8

Проаналізувавши дані, маємо загальну оцінку:

Розроблене ПЗ – 88 балів;

Kinoafisha.ua - 82 бали;

Planeta-kino.ua - 83 бали.

Отже, можна дійти висновку що якість роботи нової програми бронювання квитків у кінотеатрі вища за аналоги приблизно на 5,6%.

3.4 Висновок до розділу 3

У даному розділі проведено вибір середовища розробки мови програмування, вибір бібліотек та компонентів, що найдоцільніше використовувати при програмній реалізації модуля бронювання квитків у кінотеатрі. Для програмної реалізації модуля бронювання квитків у кінотеатрі було обрано ОС Android, як систему, що має відкритий тип, багатий функціонал і є найбільш поширеною. Експериментально підтверджено, що ефективність роботи нової програми замовлення квитків у кінотеатр вища за аналоги приблизно на 5,6%.

ВИСНОВКИ

Всі завдання, поставлені в бакалаврській кваліфікаційній роботі були виконані в повному обсязі. Обґрунтувано доцільність розробки програмного модуля бронювання квитків у кінотеатри. Розглянуто принципи попереднього продажу, бронювання та реалізації квитків у кінотеатри у квиткових касах. Проведено аналіз переваг та недоліків сучасних програм-аналогів, які використовуються для замовлення квитків у кінотеатр. Наведено короткий опис основних функцій, які виконують дані програми. Розроблено UML-діаграми програмного модуля бронювання квитків у кінотеатрі. Побудовано інформаційну модель основних модулів програмного забезпечення, що складається із вхідних даних, даних для обробки, методів обробки, вихідних даних. Проведено вибір середовища розробки мови програмування, вибір бібліотек та компонентів, що найдоцільніше використовувати при програмній реалізації модуля бронювання квитків у кінотеатрі. Для програмної реалізації модуля бронювання квитків у кінотеатрі було обрано ОС Android, як систему, що має відкритий тип, багатий функціонал і є найбільш поширеною. Експериментально підтверджено, що якість роботи нової програми бронювання квитків у кінотеатрі вища за аналоги приблизно на 5,6%.

Отже всі поставлені задачі виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Планета Кіно [Електронний ресурс]: Мережа кінотеатрів IMAX в Україні. Режим доступу до матеріалу: http://planetakino.ua (10.04.2025) - Назва з екрану.

2. Кіноафіша [Електронний ресурс]: – Режим доступу до матеріалу: http://kinoafisha.ua (10.04.2025) - Назва з екрану.

3. Романюк О.Н. /Веб-дизайн і комп'ютерна графіка. Навчальний посібник/ О.Н. Романюк, Д.І. Катєльніков, О.П. Косовець. – Вінниця: ВНТУ, 2007. – 147с.

4. Модель «Сутність-зв'язок» [Електронний ресурс]: Матеріал з Вікіпедії – вільної енциклопедії. Режим доступу до матеріалу: https://uk.wikipedia.org/wiki/Модель_«сутність_—_зв'язок» (10.04.2025) - Назва з екрану.

5. Клієнт-серверна архітектура та ролі серверів [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0 (10.04.2025) - Назва з екрану.

6. Django [Електронний ресурс]: The web framework for perfectionists with deadlines. Режим доступу до матеріалу: https://www.djangoproject.com (10.04.2025) - Назва з екрану.

7. Android-додаток "Планета Кіно" [Електронний ресурс]. — Режим доступу: https://play.google.com/store/apps/details?id=com.planet.imax

8. Android-додаток "SmartCinema" [Електронний ресурс]. — Режим доступу: https://play.google.com/store/apps/details?id=ua.kinoteatr.SmartCinema

9. Android Studio Release Notes [Електронний ресурс]. — Режим доступу: https://developer.android.com/studio/releases/index.html

10. Wharton J. Android's Java 9, 10, 11, and 12 Support [Електронний ресурс]. — Режим доступу: https://jakewharton.com/androids-java-9-10-11-and-12-support/

11. Аутентифікація і авторизація: що це і в чому відмінність[Електронний ресурс] – Режим доступу: https://qagroup.com.ua/publications (дата звернення 22.05.2025).

12. Основні принципи візуального дизайну в UX[Електронний ресурс] – Режим доступу: https://blog.ithillel.ua/articles/basic-principles-of-visual-design-in-ux (дата звернення 22.05.2025).

13. Android Studio system requirements [Електронний ресурс]. — Режим доступу: https://developer.android.com/studio

14. Learning Java [Електронний ресурс]. — NetBeans, 2015. — P. 3-16. — Режим доступу: https://netbeans.org/kb/articles/learn-java.html

15. Android Captures Record 88% Share of Global Smartphone Shipments in Q3 2016 [Електронний ресурс]. — Режим доступу: https://www.linkedin.com/pulse/androidcaptures-record-88-share-global-smartphone-q3-stepanyan

16. Що таке OS Android [Електронний ресурс]. — Режим доступу: http://ittehnolog.com/statti/scho-take-os-android

17. Firebase: Руководство [Електронний ресурс]. — Режим доступу: https://developer.android.com/distribute/best-practices/develop/build-withfirebase?hl=RU

18. Маркін, А. В. Побудова запитів і програмування на SQL. Навчальний посібник / А.В. Маркін. - М .: Діалог-Міфи, 2014. - 384 с.

19. Asp.Net Web Api [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/uk-ua/aspnet/core/introduction-to-aspnet-core?view=aspnet core-9.0 (дата звернення: 28.05.2025)

20. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. – Вінниця : ВНТУ, 2023. – (PDF, 54 с.).

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль бронювання квитків у кінотеатрі

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 19,23 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Озеранський В.С.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Хоменчук Ж.С.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (довідниковий)
Інструкція користувача

При запуску програми з’являється “екран привітання” (рис. Б.1). Це звичайна анімація, яка створюється класом Splash і використовується для того, щоб завантажити список фільмів, інформація про які зберігається в базі даних.

Рисунок Б.1 - Екран привітання

Завантажений список фільмів користувач побачить на екрані і зможе вибрати певний фільм (рис. Б.2).

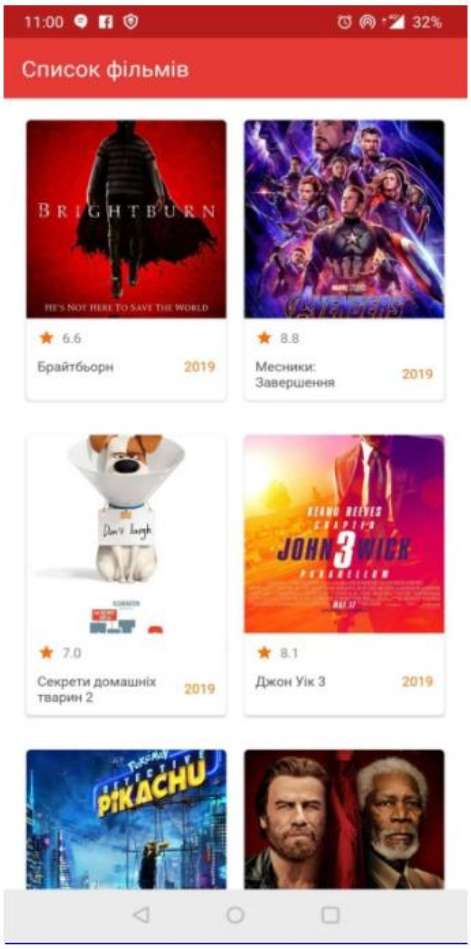

Рисунок Б.2 - Список фільмів

Після вибору фільму користувач потрапляє на його сторінку з постером, оцінкою, описом, тривалістю та кнопкою “замовити квитки”.

Після натиснення на сторінці фільму кнопки “замовити квитки” користувач потрапляє в меню сеансів, де вказано дату, час початку і час закінчення фільму, ціни квитків.

Після вибору вподобаного сеансу користувач потрапляє на сторінку кінозалу, де можна побачити всі місця за ціновою категорію, а також, які місця вільні, а які - заброньовані. Заброньовані місця є неактивними для натискання й позначені замком (рис. Б.3 - Б.4)

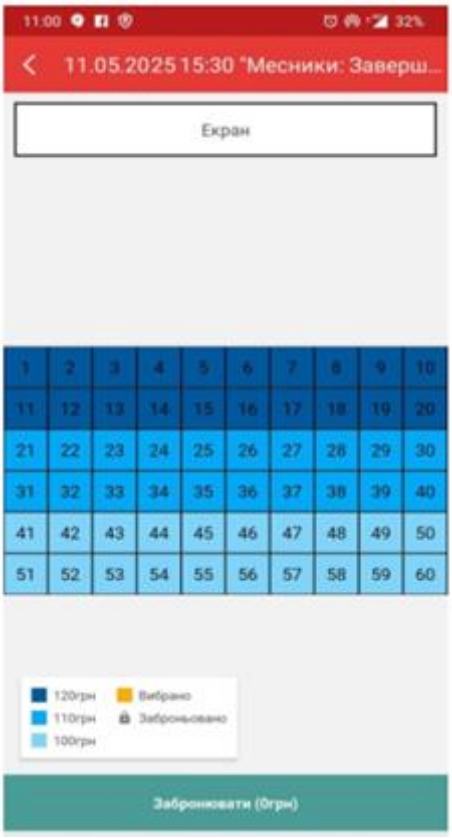

Рисунок Б.3 - Вільні місця (виділено кольором за ціновими категоріями)

Рисунок Б.4 - Заброньовані місця (позначено замком)

Вибравши відповідне місце, користувач може забронювати чи одразу й оплатити його. При бронюванні чи оплаті потрібно ввести свої контактні дані, система перевірить, чи не залишилися поля порожніми. При оплаті система проводить валідацію банківської карти. Для тестування додано валідність для карти 4242 4242 4242 4242 (рис. Б.5).

Рисунок Б.5 - Меню оплати квитка

Після резервування чи оплати користувач отримає сповіщення про успішну операцію, а дані будуть занесені в базу даних.

ДОДАТОК В (обов'язковий)
Лістинг програми

Клас MainMenu

```
package com.kino.booker.ui.generator;
import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.ScrollView;
import com.google.android.material.textfield.TextInputLayout;
import com.kino.booker.R;
import com.kino.booker.base.view.ToolbarButtonType;
import com.kino.booker.base.view.activity.BaseActivity;
import com.kino.booker.base.view.activity.BaseToolbarActivity;
import com.kino.booker.model.FilmDetailsModel;
import com.kino.booker.utils.DateUtils;
import com.kino.booker.utils.Extras;
import com.kino.booker.utils.card.CardSpan;
import java.util.Calendar;
import androidx.annotation.NonNull;
import androidx.appcompat.widget.AppCompatButton;
import androidx.appcompat.widget.AppCompatEditText;
import androidx.appcompat.widget.AppCompatTextView;
import androidx.appcompat.widget.Toolbar;
import butterknife.BindView;
import butterknife.OnClick;
public class GeneratorActivity extends BaseToolbarActivity<GeneratorPresenter>
implements GeneratorView {
@BindView(R.id.text_toolbar_title) AppCompatTextView textToolbarTitle;
@BindView(R.id.toolbar) Toolbar toolbar;
@BindView(R.id.edittext_name) AppCompatEditText edittextName;
@BindView(R.id.layout_name) TextInputLayout layoutName;
@BindView(R.id.edittext_description) AppCompatEditText edittextDescription;
@BindView(R.id.layout_description) TextInputLayout layoutDescription;
@BindView(R.id.edittext_image) AppCompatEditText edittextImage;
@BindView(R.id.layout_image) TextInputLayout layoutImage;
@BindView(R.id.edittext_year) AppCompatEditText edittextYear;
@BindView(R.id.layout_year) TextInputLayout layoutYear;
@BindView(R.id.edittext_rate) AppCompatEditText edittextRate;
@BindView(R.id.layout_rate) TextInputLayout layoutRate;
@BindView(R.id.edittext_duration_minutes) AppCompatEditText
edittextDurationMinutes;
@BindView(R.id.layout_duration_minutes) TextInputLayout
layoutDurationMinutes;
@BindView(R.id.edittext_youtube_id) AppCompatEditText edittextYoutubeId;
@BindView(R.id.layout_youtube_id) TextInputLayout layoutYoutubeId;
@BindView(R.id.edittext_price_1) AppCompatEditText edittextPrice1;
@BindView(R.id.layout_price_1) TextInputLayout layoutPrice1;
```

```
@BindView(R.id.edittext_price_2) AppCompatEditText edittextPrice2;
@BindView(R.id.layout_price_2) TextInputLayout layoutPrice2;
@BindView(R.id.edittext_price_3) AppCompatEditText edittextPrice3;
@BindView(R.id.layout_price_3) TextInputLayout layoutPrice3;
@BindView(R.id.layout_root) LinearLayout layoutRoot;
@BindView(R.id.scroll_view_root) ScrollView scrollViewRoot;
@BindView(R.id.button_generate) AppCompatButton buttonGenerate;
@BindView(R.id.button_seanse_time) AppCompatButton buttonSeanseTime;
public static Intent getLaunchIntent(final BaseActivity activity, FilmDetailsModel
filmDetails) {
Intent intent = new Intent(activity, GeneratorActivity.class);
intent.putExtra(Extras.FILM_DETAILS, filmDetails);
return intent;
}
@Override
protected void injectToComponent() {
getActivityComponent().inject(this);
}
@Override
public int getLayoutResource() {
return R.layout.activity_generator;
}
@NonNull
@Override
protected ToolbarButtonType getToolbarButtonType() {
return ToolbarButtonType.BACK;
}
@Override
protected String getToolbarTitle() {
return "Генератор фільму";
}
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
if (getIntent().getExtras() != null &&
getIntent().getExtras().getParcelable(Extras.FILM_DETAILS) != null) {
buttonSeanseTime.setVisibility(View.GONE);
}
getPresenter().setData(getIntent().getExtras().getParcelable(Extras.FILM_DETAILS)
);
edittextName.setText(getPresenter().getFilmDetails().getName());
edittextDescription.setText(getPresenter().getFilmDetails().getDescription());
edittextImage.setText(getPresenter().getFilmDetails().getImageUrl());
edittextYoutubeId.setText(getPresenter().getFilmDetails().getTrailerUrl());
edittextYear.setText(getPresenter().getFilmDetails().getYear() + "");
edittextDurationMinutes.setText(getPresenter().getFilmDetails().getDurationHours()
* 60 + getPresenter().getFilmDetails().getDurationMinutes() + "");
edittextRate.setText(getPresenter().getFilmDetails().getRating() + "");
edittextPrice1.setText((int)
Math.round(getPresenter().getFilmDetails().getPrice1()) + "");
edittextPrice2.setText((int)
Math.round(getPresenter().getFilmDetails().getPrice2()) + "");
```

```
edittextPrice3.setText((int)
Math.round(getPresenter().getFilmDetails().getPrice3()) + "");
}
@OnClick(R.id.button_generate)
void onGenerateClick() {
layoutName.setError(null);
layoutDescription.setError(null);
layoutImage.setError(null);
layoutYoutubeId.setError(null);
layoutYear.setError(null);
layoutDurationMinutes.setError(null);
layoutRate.setError(null);
layoutPrice1.setError(null);
layoutPrice2.setError(null);
layoutPrice3.setError(null);
if (edittextName.getText().toString().isEmpty()) {
layoutName.setError("Має бути не пустим");
return;
}
if (edittextDescription.getText().toString().isEmpty()) {
layoutDescription.setError("Має бути не пустим");
return;
}
if (edittextImage.getText().toString().isEmpty()) {
layoutImage.setError("Має бути не пустим");
return;
}
if (edittextYoutubeId.getText().toString().isEmpty()) {
layoutYoutubeId.setError("Має бути не пустим");
return;
}
if (edittextYear.getText().toString().isEmpty()) {
layoutYear.setError("Має бути не пустим");
return;
}
if (edittextDurationMinutes.getText().toString().isEmpty()) {
layoutDurationMinutes.setError("Має бути не пустим");
return;
}
if (edittextRate.getText().toString().isEmpty()) {
layoutRate.setError("Має бути не пустим");
return;
}
if (edittextPrice1.getText().toString().isEmpty()) {
layoutPrice1.setError("Має бути не пустим");
return;
}
if (edittextPrice2.getText().toString().isEmpty()) {
layoutPrice2.setError("Має бути не пустим");
return;
}
if (edittextPrice3.getText().toString().isEmpty()) {
```

```
layoutPrice3.setError("Має бути не пустим");
return;
}
String name = edittextName.getText().toString().trim();
String description = edittextDescription.getText().toString().trim();
String imageUrl = edittextImage.getText().toString().trim();
String trailerUrl = edittextYoutubeId.getText().toString().trim();
int year = Integer.parseInt(edittextYear.getText().toString().trim());
int durationMinutes =
Integer.parseInt(edittextDurationMinutes.getText().toString().trim());
float rating = Float.parseFloat(edittextRate.getText().toString().trim());
double price1 = Double.parseDouble(edittextPrice1.getText().toString().trim());
double price2 = Double.parseDouble(edittextPrice2.getText().toString().trim());
double price3 = Double.parseDouble(edittextPrice3.getText().toString().trim());
if (getPresenter().getSeanseDate() == null) {
showMessage(null, "Виберіть дату першого сеансу");
return;
}
getPresenter().generate(name, description, imageUrl, trailerUrl, year,
durationMinutes, rating, price1, price2, price3);
}
@Override
public void closeActivity() {
finish();
}
@OnClick(R.id.button_seanse_time)
void onSeanseTimeClick() {
final Calendar currentDate = Calendar.getInstance();
Calendar date = Calendar.getInstance();
new DatePickerDialog(this, R.style.TimePickerDialogTheme, (view, year,
monthOfYear, dayOfMonth) -> {
date.set(year, monthOfYear, dayOfMonth);
new TimePickerDialog(GeneratorActivity.this,
R.style.TimePickerDialogTheme, (view1, hourOfDay, minute) -> {
date.set(Calendar.HOUR_OF_DAY, hourOfDay);
date.set(Calendar.MINUTE, minute);
getPresenter().setSeanseDate(date);
buttonSeanseTime.setText(DateUtils.getDateStringByPattern(date.getTime(),
DateUtils.FULL_DATE_PATTERN_2));
}, currentDate.get(Calendar.HOUR_OF_DAY),
currentDate.get(Calendar.MINUTE), true).show();
}, currentDate.get(Calendar.YEAR), currentDate.get(Calendar.MONTH),
currentDate.get(Calendar.DATE)).show();
}
}
```

Презентер

```
package com.kino.booker.ui.generator;
import android.util.Pair;
import com.google.firebase.firestore.DocumentReference;
import com.google.firebase.firestore.FirebaseFirestore;
import com.kino.booker.base.presenter.BasePresenter;
import com.kino.booker.di.scopes.ConfigPersistentScope;
```

```
import com.kino.booker.model.FilmDetailsModel;
import com.kino.booker.model.SeanseModel;
import com.kino.booker.model.SeansesListModel;
import com.kino.booker.model.SeatModel;
import com.kino.booker.utils.Constants;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import javax.inject.Inject;
@ConfigPersistentScope
public class GeneratorPresenter extends BasePresenter<GeneratorView> {
private FirebaseFirestore mDatabase = FirebaseFirestore.getInstance();
private FilmDetailsModel mFilmDetails;
private Calendar mSeanseDate;
@Inject
public GeneratorPresenter() {
}
/**
* Задає модель фільму. Якщо її немає - то створюється новий. Якщо є - то
апдейтиться.
*
* @param filmDetails
*/
void setData(FilmDetailsModel filmDetails) {
if (filmDetails == null) {
mFilmDetails = new FilmDetailsModel();
// mFilmDetails.setName("Термінатор Генезис");
// mFilmDetails.setDescription("Опис фільму термінатор");
//
mFilmDetails.setImageUrl("https://m.mediaamazon.com/images/M/MV5BMjM1NTc0NzE4OF5BMl5Ba
nBnXkFtZTgwNDkyNj
Q1NTE@._V1_.jpg");
// mFilmDetails.setTrailerUrl("jNU_jrPxs-0");
// mFilmDetails.setYear(2015);
// mFilmDetails.setDurationHours(2);
// mFilmDetails.setDurationMinutes(6);
// mFilmDetails.setRating(7);
// mFilmDetails.setPrice1(110);
// mFilmDetails.setPrice2(110);
// mFilmDetails.setPrice3(110);
} else {
mFilmDetails = filmDetails;
mSeanseDate = Calendar.getInstance();
}
}
public FilmDetailsModel getFilmDetails() {
return mFilmDetails;
}
public void setSeanseDate(Calendar seanseDate) {
this.mSeanseDate = seanseDate;
}
public Calendar getSeanseDate() {
```

```
return mSeanseDate;
}
/**
* Генерує фільм і зберігає його на сервері.
*/
/**
* @param name
* @param description
* @param imageUrl
* @param trailerUrl
* @param year - рік випуску
* @param durationMinutes - тривалість (хв)
* @param rating - рейтинг IMDB
* @param price1- ціна 1
* @param price2- ціна 2
* @param price3 - ціна 3
*/
public void generate(String name, String description, String imageUrl, String
trailerUrl,
int year, int durationMinutes, float rating, double price1, double
price2, double price3) {
int hours = durationMinutes / 60;
int minutes = durationMinutes % 60;
mFilmDetails.setName(name);
mFilmDetails.setDescription(description);
mFilmDetails.setImageUrl(imageUrl);
mFilmDetails.setTrailerUrl(trailerUrl);
mFilmDetails.setYear(year);
mFilmDetails.setDurationHours(hours);
mFilmDetails.setDurationMinutes(minutes);
mFilmDetails.setRating(rating);
mFilmDetails.setPrice1(price1);
mFilmDetails.setPrice2(price2);
mFilmDetails.setPrice3(price3);
if (mFilmDetails.getId() == null) {
addFilm();
} else {
saveFilm();
}
}
/**
* Зберігає редагований фільм
*/
private void saveFilm() {
getView().showProgress(true);
DocumentReference ref =
mDatabase.document(Constants.COLLECTION_FILMS + "/" +
mFilmDetails.getId());
ref.set(mFilmDetails).addOnSuccessListener(documentReference -> {
getView().showProgress(false);
getView().closeActivity();
}).addOnFailureListener(e -> {
```

```
getView().showProgress(false);
getView().showError("Сталася помилка", "" + e.getMessage());
});
}
/**
* Додає новий фільм
*/
private void addFilm() {
getView().showProgress(true);
SeansesListModel seansesList = new
SeansesListModel(generateSeansTime(mFilmDetails));
mDatabase.collection(Constants.COLLECTION_FILMS)
.add(mFilmDetails)
.addOnSuccessListener(documentReference -> {
mFilmDetails.setId(documentReference.getId());
saveSeanses(mFilmDetails.getId(), seansesList);
})
.addOnFailureListener(e -> {
getView().showProgress(false);
getView().showError("Сталася помилка", "" + e.getMessage());
});
}
/**
* Генерує час сеансівю
*
* @param film
* @return
*/
private List<SeanseModel> generateSeansTime(FilmDetailsModel film) {
Calendar start = (Calendar) mSeanseDate.clone();
List<Pair<Long, Long>> times = new ArrayList<>();
for (int i = 0; i < 10; i++) {
long first = start.getTimeInMillis();
start.add(Calendar.HOUR_OF_DAY, film.getDurationHours());
start.add(Calendar.MINUTE, film.getDurationMinutes());
long second = start.getTimeInMillis();
times.add(new Pair<>(first, second));
start.add(Calendar.DAY_OF_MONTH, 1);
start.set(Calendar.HOUR_OF_DAY,
mSeanseDate.get(Calendar.HOUR_OF_DAY));
start.set(Calendar.MINUTE, mSeanseDate.get(Calendar.MINUTE));
}
return generateSeansModels(film, times);
}
/**
* Генерує сеанси
*
* @param film
* @param times
* @return
*/
private List<SeanseModel> generateSeansModels(FilmDetailsModel film,
```

```
List<Pair<Long, Long>> times) {
List<SeanseModel> seansesList = new ArrayList<>();
for (Pair<Long, Long> time : times) {
SeanseModel seansModel = new SeanseModel();
seansModel.setStartDate(time.first);
seansModel.setEndDate(time.second);
seansModel.setSeats(generateSeats(film.getPrice1(), film.getPrice2(),
film.getPrice3()));
seansesList.add(seansModel);
}
return seansesList;
}
/**
* Генерує місця
*
* @param price1
* @param price2
* @param price3
* @return
*/
private List<SeatModel> generateSeats(double price1, double price2, double
price3) {
List<SeatModel> seats = new ArrayList<>();
for (int i = 1; i < 21; i++) {
SeatModel seatModel = new SeatModel();
seatModel.setNumber(i);
seatModel.setPrice(price1);
seatModel.setLocked(false);
seats.add(seatModel);
}
for (int i = 21; i < 41; i++) {
SeatModel seatModel = new SeatModel();
seatModel.setNumber(i);
seatModel.setPrice(price2);
seatModel.setLocked(false);
seats.add(seatModel);
}
for (int i = 41; i < 61; i++) {
SeatModel seatModel = new SeatModel();
seatModel.setNumber(i);
seatModel.setPrice(price3);
seatModel.setLocked(false);
seats.add(seatModel);
}
return seats;
}
/**
* Зберігає сеанси
*
* @param filmId
* @param seansesList
*/
```

```
private void saveSeanses(String filmId, SeansesListModel seansesList) {
mDatabase.collection(Constants.COLLECTION_FILMS + "/" + filmId + "/" +
Constants.COLLECTION_SEANSES)
.add(seansesList)
.addOnSuccessListener(documentReference -> {
getView().showProgress(false);
getView().closeActivity();
})
.addOnFailureListener(e -> {
getView().showProgress(false);
getView().showError("Сталася помилка", "" + e.getMessage());
});
}
}
```

ДОДАТОК Г (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРІ»

Виконала: студентка 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Хоменчук Ж.С.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри КН
Озеранський В.С.
(прізвище та ініціали)

Вінниця ВНТУ - 2025 рік

Рисунок Г.1 – Програми аналоги бронювання квитків у кінотеатрі

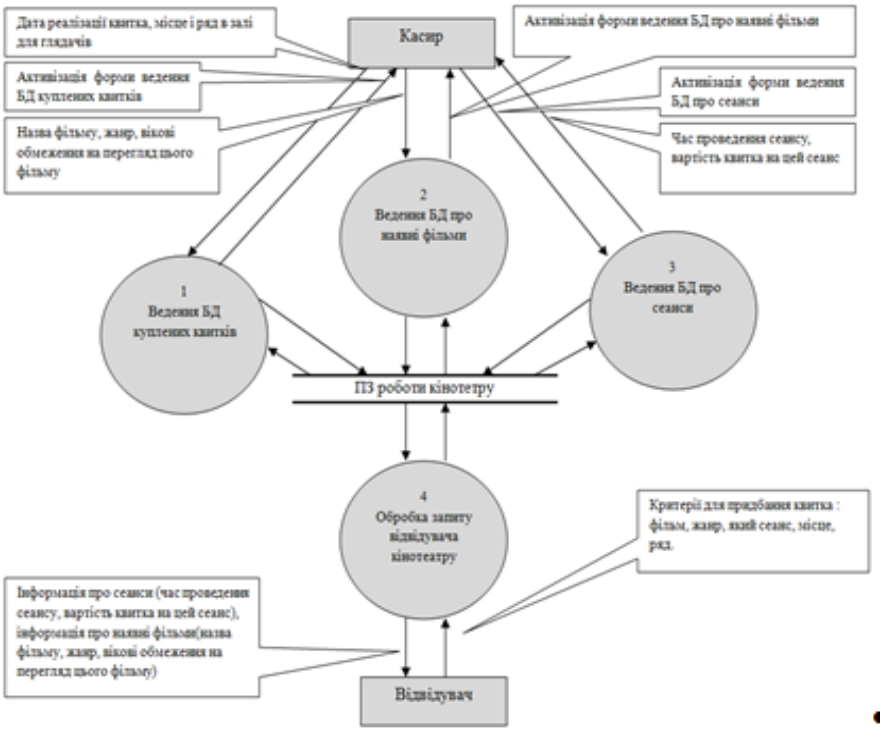

Рисунок Г.2 – Контекстна діаграма потоків даних процесу бронювання квитків у кінотеатрі

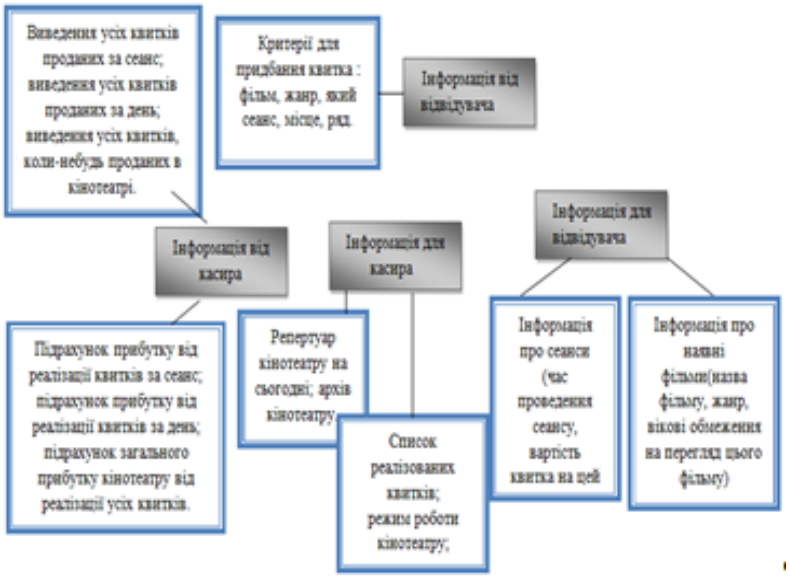

Рисунок Г.3 – Діаграма структур даних програмного модуля бронювання квитків у кінотеатр

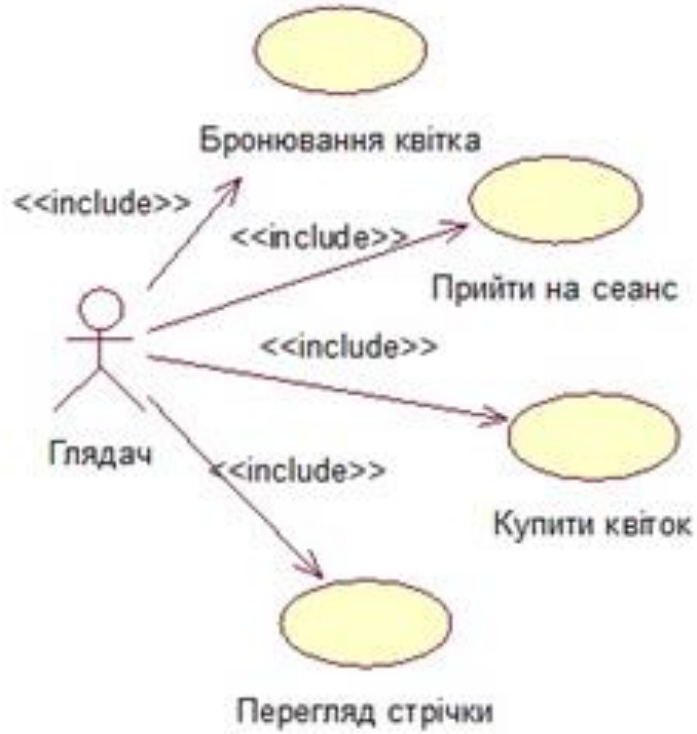

Рисунок Г.4 – Use Case - діаграма програмного модуля бронювання квитків у кінотеатрі

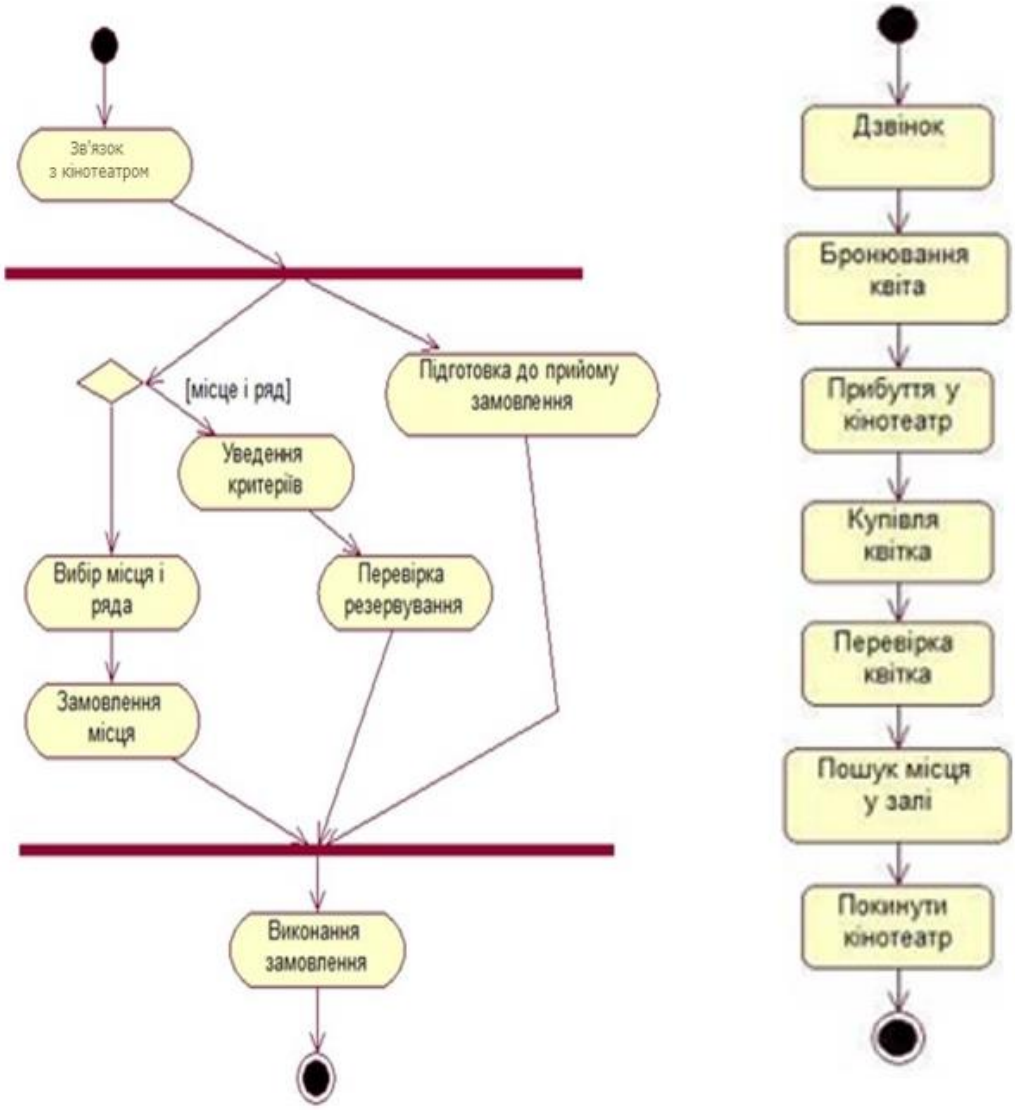

Рисунок Г.5 – Діаграми діяльності процесу бронювання квитків у кінотеатрі

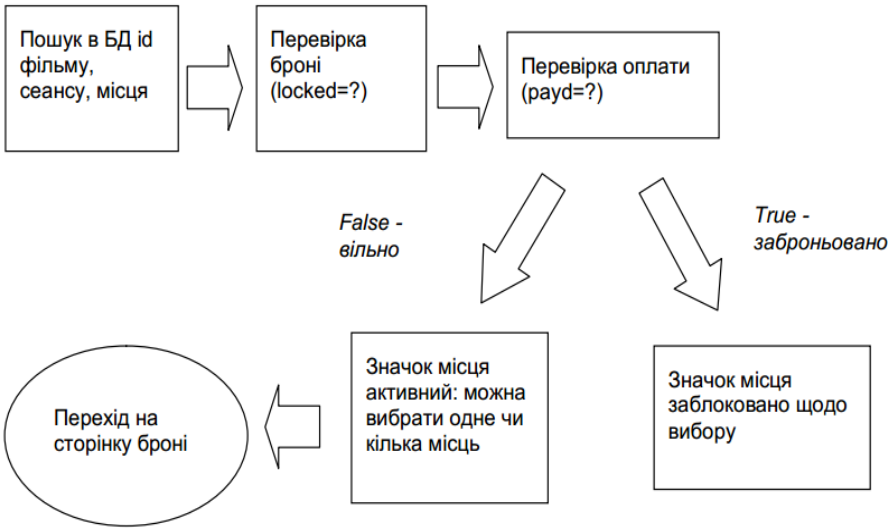

Рисунок Г.6 – Діаграма аналізу наявності вільного місця в кінотеатрі

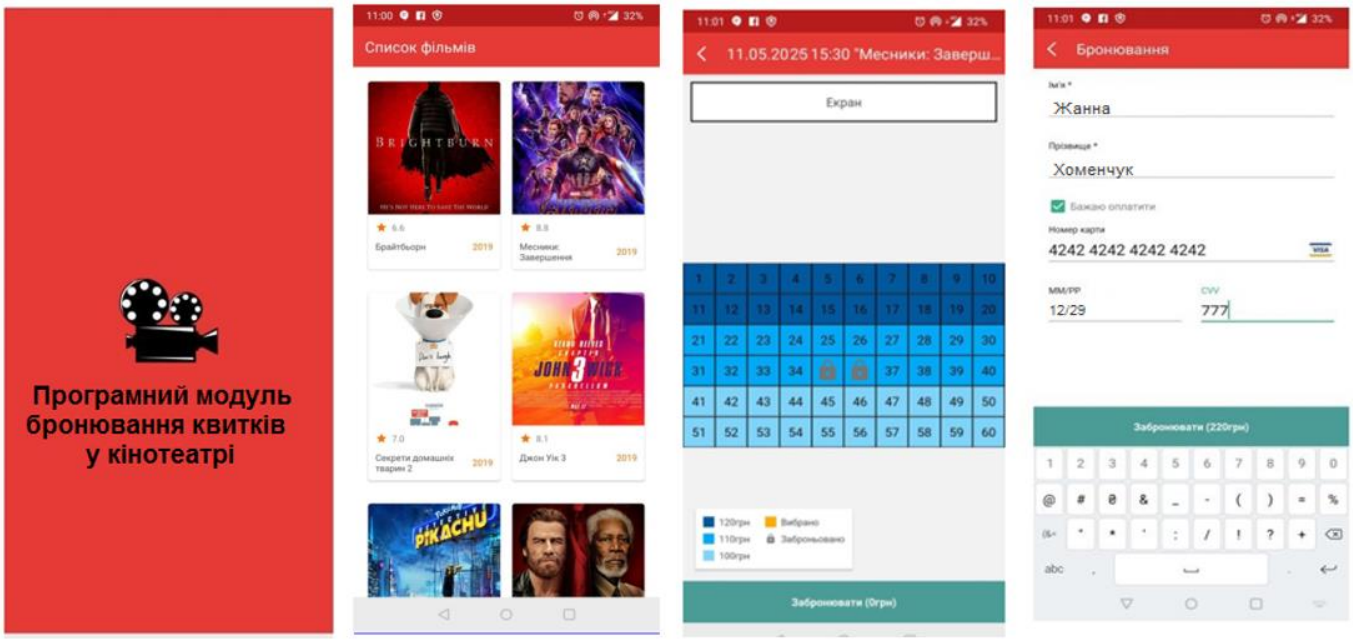

Рисунок Г.7 – Графічний інтерфейс програмного модуля бронювання квитків у кінотеатрі

ДОДАТОК Д (довідниковий)

Довідка про впровадження

МАЛЕ НАУКОВО-ВИРОБНИЧЕ ПІДПРИЄМСТВО "ТОВ "ІТІ"
Україна, 21021, м.Вінниця, вул. Келецька, 56

№ 52 від "6" 06 2025р.

ДОВІДКА

Дана Хоменчук Жанні Станіславівні в тому, що результати, одержані нею в процесі виконання бакалаврської кваліфікаційної роботи, а саме алгоритм та програмне забезпечення бронювання квитків у кінотеатрі, планується використати в розробках ТОВ «ІТІ».

Зам. директора ТОВ «ІТІ» Бодяк В.М.