

Міністерство освіти і науки України  
Одеський національний політехнічний університет  
Інститут комп'ютерних систем  
Кафедра інформаційних систем

## **КУРСОВА РОБОТА**

з дисципліни «Технології створення програмних продуктів»

за темою

«Що подивитись?»

Частина № 1

Виконав:  
студент 3-го курсу  
групи АІ-181  
Олійник Вадим Миколайович  
Перевірив:  
Блажко О. А.

Одеса-2020

## Анотація

В курсовій роботі розглядається процес створення програмного продукту «Що подивитись?». Робота виконувалась в команді з декількох учасників: Олійник В.М., Сояк А.І., Пшеничнюк А.О.

Тому в пояснювальній записці у розділах «Проектування» та «Конструювання» детальніше описано лише одну частину з урахуванням планів проведених робіт з розділу «Планування» з описом особливостей конструювання:

- структур даних моделі Django MVT в системі керування базами даних PostgreSQL;
- програмних модулів в інструментальному середовищі PyCharm з використанням фреймворку Django та мови програмування Python

Результати роботи розміщено на *github*-репозиторії за адресою: [https://github.com/VadimKukuzia/what\\_to\\_watch](https://github.com/VadimKukuzia/what_to_watch)

Робота з додатком передбачає перехід користувача на сайт <https://what-to-watch-sop.herokuapp.com>

## **Перелік скорочень**

ОС – операційна система

ІС – інформаційна система

БД – база даних

СКБД – система керування базами даних

ПЗ – програмне забезпечення

ПП– програмний продукт

UML – уніфікована мова моделювання

## ЗМІСТ

1	Вимоги до програмного продукту	6
1.1	Визначення потреб споживача	6
1.1.1	Ієрархія потреб споживача	6
1.1.2	Деталізація матеріальної потреби	7
1.2	Бізнес-вимоги до програмного продукту	8
1.2.1	Опис проблеми споживача	8
1.2.1.1	Концептуальний опис проблеми споживача	8
1.2.1.2	Метричний опис проблеми споживача	9
1.2.2	Мета створення програмного продукту	9
1.2.2.1	Проблемний аналіз існуючих програмних продуктів	9
1.2.2.2	Мета створення програмного продукту	10
1.2.3	Назва програмного продукту	10
1.2.3.1	Гасло програмного продукту	11
1.2.3.2	Логотип програмного продукту	11
1.3	Вимоги користувача до програмного продукту	11
1.3.1	Історія користувача програмного продукту	11
1.3.2	Діаграма прецедентів програмного продукту	12
1.3.3	Сценаріїв використання прецедентів програмного продукту	13
1.4	Функціональні вимоги до програмного продукту	15
1.4.1.	Багаторівнева класифікація функціональних вимог	15
1.4.2	Функціональний аналіз існуючих програмних продуктів	18
1.5	Нефункціональні вимоги до програмного продукту	19
1.5.1	Опис зовнішніх інтерфейсів	19
1.5.1.1	Опис інтерфейса користувача	19
1.5.1.1.1	Опис INPUT-інтерфейса користувача	19

					ІС КР 122 АІ181 ПЗ			
<b>Змін</b>	<b>Арк.</b>	<b>№ докум.</b>	<b>Підпис</b>	<b>Дата</b>				
Виконав		Олійник В.М.			Веб-додаток «Що подивитись?»	<b>Літ.</b>	<b>Лист</b>	<b>Листів</b>
Перев.		Блажко О. А.					3	65
Реценз.						ОНПУ, каф. ІС, АІ-181		
Н. Контр.								
Утверд.								

1.5.1.1.2	Опис OUTPUT-інтерфейса користувача	20
1.5.1.2	Опис інтерфейсу із зовнішніми пристроями	24
1.5.1.3	Опис програмних інтерфейсів	25
1.5.1.4	Опис інтерфейсів передачі інформації	25
1.5.1.5	Опис атрибутів продуктивності	25
2	Планування процесу розробки програмного продукту	27
2.1	Планування ітерацій розробки програмного продукту	27
2.2	Концептуальний опис архітектури програмного продукту	30
2.3	План розробки програмного продукту	30
2.3.1	Оцінка трудомісткості розробки програмного продукту	30
2.3.2	Визначення дерева робіт з розробки програмного продукту	33
2.3.3	Графік робіт з розробки програмного продукту	34
2.3.3.1	Таблиця з графіком робіт	34
2.3.3.2	Діаграма Ганта	35
3	Проектування програмного продукту	36
3.1	Концептуальне та логічне проектування структур даних програмного продукту	36
3.1.1	Концептуальне проектування на основі UML-діаграми концептуальних класів	36
3.1.2	Логічне проектування структур даних	37
3.2	Проектування програмних класів	37
3.3	Проектування алгоритмів роботи методів програмних класів	39
3.4	Проектування тестових наборів методів програмних класів	41
4	Конструювання програмного продукту	43
4.1	Особливості конструювання структур даних	43
4.1.1	Особливості інсталяції та роботи з СУБД	43
4.1.2	Особливості створення структур даних	43
4.2	Особливості конструювання програмних модулів	45
4.2.1	Особливості роботи з інтегрованим середовищем розробки	45

4.2.2 Особливості створення програмної структури з урахуванням спеціалізованого Фреймворку	45
4.2.3 Особливості створення програмних класів	46
4.2.4 Особливості розробки алгоритмів методів програмних класів або процедур/функцій	46
4.3 Модульне тестування програмних класів	48
5 Розгортання та валідація програмного продукту	55
5.1 Інструкція з встановлення програмного продукту	55
5.2 Інструкція з використання програмного продукту	61
5.3 Результати валідації програмного продукту	65
Висновки до курсової роботи	66

# 1 Вимоги до програмного продукту

## 1.1 Визначення потреб споживача

### 1.1.1 Ієрархія потреб споживача

Згідно А. Маслоу, людські потреби мають рівні від більш простих до більш високим, і прагнення до більш високих потреб, як правило, можливо і виникає тільки після задоволення потреб нижчого порядку, наприклад, в їжі і безпеки.

В своїй роботі «Мотивація і особистість» (1954) Маслоу припустив, що всі потреби людини вроджені, і що вони організовані в ієрархічну систему пріоритету або домінування, що складається з п'яти рівнів:

- Фізіологічні потреби (їжа, вода, сон тощо)
- Потреба в безпеці (стабільність, порядок, залежність, захист)
- Потреба в любові і приналежності (сім'я, дружба, своє коло)
- Потреба в повазі та визнання (я поважаю себе, шанують мене, я відомий і потрібен. 1: я досягаю, 2: престиж і репутація, статус, слава)
- Потреба в самоактуалізації (Самовираження) (розвиток здібностей, творчість, моральність . Людина повинна займатися тим, до чого у нього є схильності і здатності).

На рисунку 1.1.1 представлено рівень потреби споживача, який хотілося б задовольнити, використовуючи майбутній програмний продукт.

Був обраний рівень «Самовираження», тому що, використовуючи програмний продукт «Що подивитись?», споживач задовольняє такі потреби, як потреба у творчості, культурному розвитку, самоактуалізація, усе це можна об'єднати в одне словосполучення «Перегляд фільму».

					ІС КР 122 АІ181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			6



Рис. 1.1.1 – Рівень потреби споживача

### 1.1.2 Деталізація матеріальної потреби

Для деталізації матеріальної потреби була використана ментальна карта (MindMap). При створенні ментальних карт матеріальна потреба розташовується в центрі карти. Асоціативні гілки можна швидко створити, припускаючи, що в загальному вигляді з об'єктом пов'язані три потоки даних / інформації: вхідний, внутрішній, вихідний. Кожен потік - це асоціативна група, що включає можливі п'ять гілок, що відповідають на п'ять питань: Хто? Що? Де? Коли? Як?

Потреба, яка була визначена при аналізі матеріальних проблем споживача зображені на рисунку 1.1.2



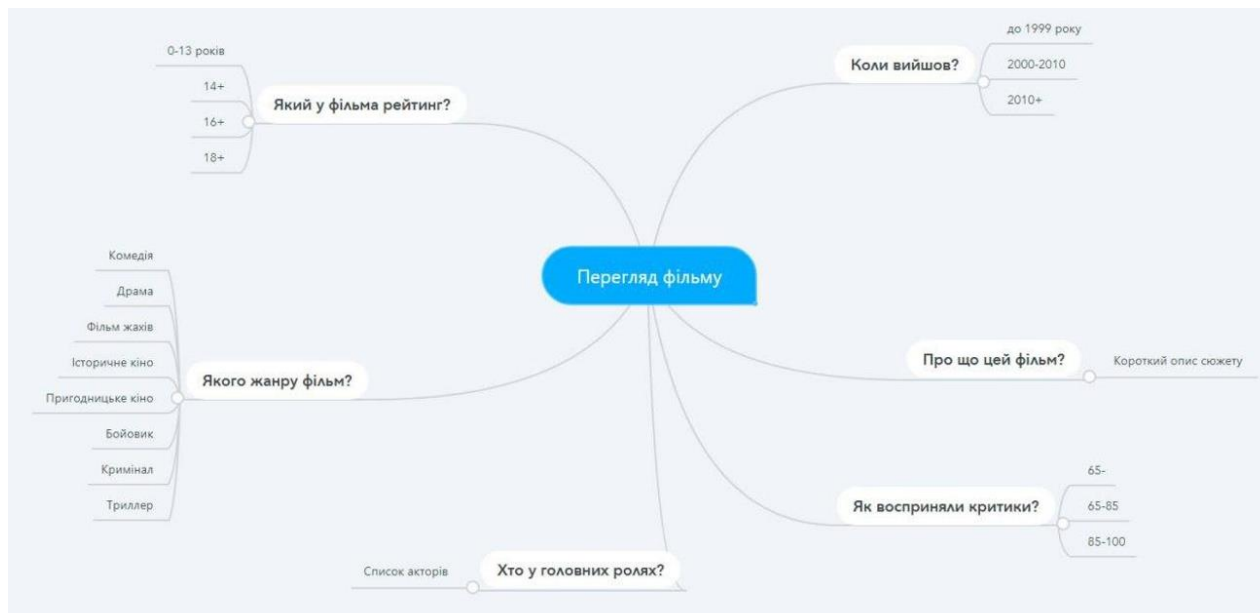


Рис. 1.1.2 – Ментальна карта деталізації матеріальної потреби

## 1.2 Бізнес-вимоги до програмного продукту

### 1.2.1 Опис проблеми споживача

#### 1.2.1.1 Концептуальний опис проблеми споживача

Для скорочення часу і коштів при задоволенні реальних потреб людині потрібна інформація, що призводить до появи інформаційної потреби.

Для аналізу проблем зі сторони споживача була обрана статистика найпопулярнішого стримінгового сервісу планети Netflix, а саме його інформація про десять фільмів та серіалів, що користувалися найбільшою популярністю у 2020 року. Після цього, ми порівняли наявність повноцінної інформації про ці фільми та серіали на сервісах іноземних (IMDB, Rotten Tomatoes) та російськомовних (KinoPoisk).

Так як розроблений ресурс орієнтується на україно- та російськомовних громадян, основною проблемою споживача є відсутність інформації про контент на рідній мові.

Як було сказано вище, за базу перевірки інформаційної потреби було взято 10 найпопулярніших тайтлів у 2020 році, а саме:

- Ink Master (Повністю відсутня локалізація на KinoPoisk);
- The Office (Локалізовано та представлено на KinoPoisk);
- Mr. Iglesias (Повністю відсутня локалізація на KinoPoisk);
- The Crown (Локалізовано та представлено на KinoPoisk);
- Cocomelon (Повністю відсутня локалізація на KinoPoisk);
- The Queen's Gambit (Повністю відсутня локалізація на KinoPoisk);
- Virgin River (Повністю відсутня локалізація на KinoPoisk);
- Manhunt: Deadly Games (Локалізовано та представлено на KinoPoisk);
- Big Mouth (Повністю відсутня локалізація на KinoPoisk);
- Selena (Локалізовано та представлено на KinoPoisk).

Можна побачити, що лише чотири з десяти найпопулярніших фільми/серіалу були коректно надані на мові користувача, отже, можемо сформулювати критерії вимоги до інформації.

#### 1.2.1.2 Метричний опис проблеми споживача

Метричний опис проблеми споживача наведено у таблиці 1.2.2

Таблиця 1.2.1.2 – Метричний опис проблеми споживача

Вимога	Метричний показник доступності
Представленість мовою споживача	0.4

#### 1.2.2 Мета створення програмного продукту

##### 1.2.2.1 Проблемний аналіз існуючих програмних продуктів

					ІС КР 122 АІ181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			9

Для проблемного аналізу існуючих програмних продуктів був сформований список схожих за тематикою продуктів в інтернеті та проаналізовані фактори задоволення цими продуктами вимог до інформації. Аналіз наведено у таблиці 1.2.2.1

Таблиця 1.2.2.1 – Аналіз існуючих програмних продуктів

№	Назва продукту	Вартість	Ступінь готовності	Примітка
1	RottenTomatoes	Безкоштовно	1	Неможливість дивитись без реклами
2	IMDb	Безкоштовно	1	Обмежена кількість функцій сортування
3	Kinopoisk	Безкоштовно	1	Наявність важливих функцій лише за платну підписку

#### 1.2.2.2 Мета створення програмного продукту

Мета створення програмного продукту:

Покращення рівня цінності знайденої інформації при пошуку фільму для перегляду за рахунок створення можливості отримання інформації на мові користувача.

#### 1.2.3 Назва програмного продукту

Рекомендаційна система – «Що подивитись?»

					ІС КР 122 АІ181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			10

### 1.2.3.1 Гасло програмного продукту

Думай під час перегляду, а не під час пошуку.

### 1.2.3.2 Логотип програмного продукту

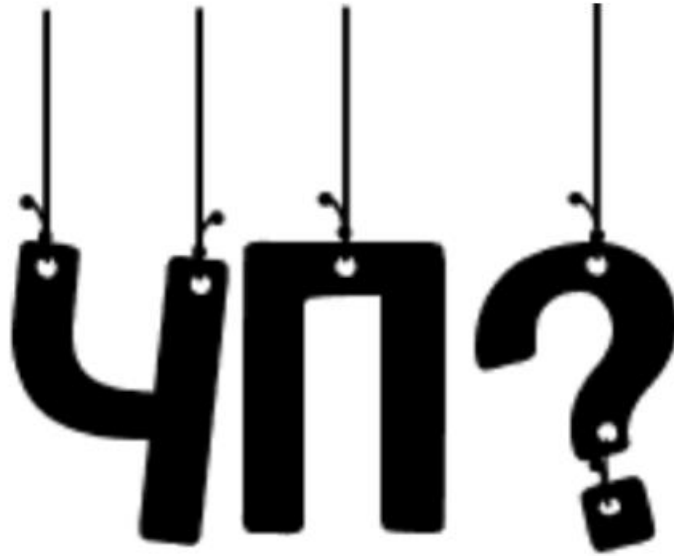


Рис. 1.2.1 – Логотип програмного продукту

## 1.3 Вимоги користувача до програмного продукту

### 1.3.1 Історія користувача програмного продукту

User-stories продукту:

- Як гість, я можу зареєструватися
- Як гість, я можу увійти до свого облікового запису користувача
- Як гість, я можу скористуватися пошуком
- Як користувач, я можу скористуватися пошуком
- Як користувач, я можу шукати фільми по заданим критеріям
- Як користувач, я можу сортувати знайдені результати
- Як користувач, я можу оцінити та залишити відгук про фільм

- Як користувач, я можу залишити запит на додавання фільму
- Як адміністратор, я можу додавати фільми в БД
- Як адміністратор, я можу редагувати/видаляти відгуки та коментарі
- Як адміністратор, я можу блокувати аккаунти користувачів

### 1.3.2 Діаграма прецедентів програмного продукту

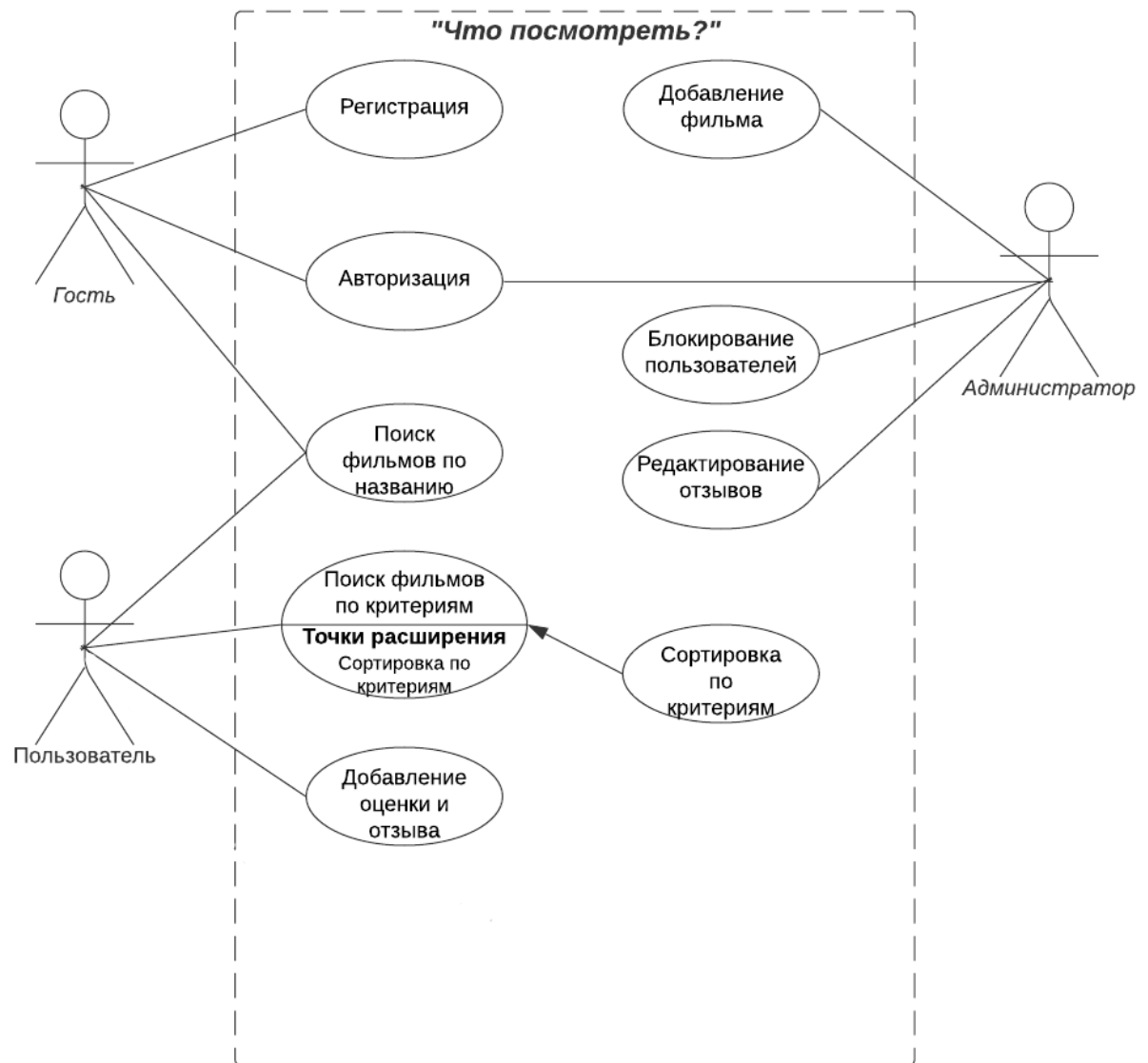


Рис. 1.3.2.1 – Діаграма прецедентів

### 1.3.3 Сценарії використання прецедентів програмного продукту

- Приклад успішного сценарію прецеденту "Реєстрація користувача":
  1. Запит на реєстрацію від гостя
  2. Запит у гостя його даних
  3. Передача даних ПП
  4. Реєстрація користувача у БД
- Альтернативний сценарій прецеденту "Реєстрація користувача":
  1. Користувач вносить некоректні дані
  2. ПП видає помилку при реєстрації і переходить до 1 кроку успішного сценарію
- Приклад успішного сценарію прецеденту "Авторизація користувача":
  1. Запит від гостя на авторизацію до системи
  2. Запит від ПП параметрів авторизації (Ідентифікаторів/Аутентифікаторів)
  3. Передача користувачем параметрів авторизації
  4. Авторизація користувача, тобто надавання доступу до інших прецедентів
- Альтернативний сценарій прецеденту "Авторизація користувача":
  1. Користувач вносить некоректні дані
  2. ПП видає помилку при авторизації і переходить до 1 кроку успішного сценарію
- Приклад успішного сценарію прецеденту "Пошук фільму за назвою":

1. Введення користувачем назви фільму.
  2. ПП проводить пошук фільма у БД.
  3. Передача даних про фільм користувачу.
- Альтернативний сценарій прецеденту "Пошук фільму за назвою":
    1. Користувач вносить некоректні дані
    2. ПП видає помилку, зв'язану з ненааявністю фільма у БД.
  - Приклад успішного сценарію прецеденту "Пошук фільму по критеріям":
    1. Користувач обирає критерії пошуку.
    2. ПП проводить пошук фільма у БД.
    3. ПП видає дані про знайдені по критеріям фільми.
  - Альтернативний сценарій прецеденту "Пошук фільму по критеріям":
    1. Користувач додатково вводить критерії сортування
    2. ПП сортує знайдені результати
  - Приклад успішного сценарію прецеденту "Додавання оцінки та відгуку":
    1. Користувач вводить оцінку/відгук.
    2. Збереження даних у БД.
    3. Оновлення сторінки відгуків, оновлення середнього рейтингу.
  - Альтернативний сценарій прецеденту "Додавання оцінки та відгуку":
    1. Користувач вносить некоректні дані.
    2. ПП видає помилку, зв'язану з неможливістю збереження відгуку у БД.
  - Приклад успішного сценарію прецеденту "Додавання фільму":

1. Адміністратор вводить дані про фільм.
  2. Дані зберігаються до БД.
  3. Рекомендації користувачів оновлюються у залежності від популярних тегів.
- Альтернативний сценарій прецеденту "Додавання фільму":
    1. Адміністратор вносить некоректні дані.
    2. ПП видає помилку, зв'язану з неможливістю збереження даних про фільм у БД.
  - Приклад успішного сценарію прецеденту "Блокування користувача":
    1. Адміністратор обирає користувача для блокування.
    2. Запит уведення причини блокування.
    3. Дані про користувача видаляються з БД.
  - Альтернативний сценарій прецеденту "Блокування користувача":
    1. Адміністратор вносить некоректні дані.
    2. ПП видає помилку, зв'язану з неможливістю блокування користувача.

#### 1.4 Функціональні вимоги до програмного продукту

##### 1.4.1. Багаторівнева класифікація функціональних вимог

Таблиця 1.4.1.1 - Багаторівнева класифікація функціональних вимог

Ідентифікатор функції	Назва функції
<b>FR1</b>	<b>Реєстрація користувача</b>



FR1.1	Створення запиту для користувача на отримання його параметрів ідентифікації та аутентифікації
FR1.2	Передача від користувача його параметрів ідентифікації та аутентифікації
FR1.3	Запис користувача до БД
<b>FR2</b>	<b>Авторизація користувача</b>
FR2.1	Створення запиту для користувача на отримання його параметрів ідентифікації та аутентифікації
FR2.2	Передача від користувача його параметрів ідентифікації та аутентифікації
FR2.3	Пошук інформації у базі даних користувачів
FR2.4	Створення сесії для користувача
<b>FR3</b>	<b>Пошук фільмів за назвою</b>
FR3.1	Введення користувачем назви фільму
FR3.2	Пошук фільму у БД
FR3.3	Повернення результату пошуку
<b>FR4</b>	<b>Пошук фільмів по критеріям</b>
FR4.1	Введення користувачем критеріїв фільму
FR4.2	Пошук фільмів у БД

FR4.3	Повернення результату пошуку
<b>FR5</b>	<b>Додавання коментаря/оцінки фільму</b>
FR5.1	Вибір фільму для додання коментаря/оцінки
FR5.2	Введення коментаря/оцінки
<b>FR6</b>	<b>Додавання фільму</b>
FR6.1	Отримання даних про бажаний фільм
FR6.2	Додавання фільму в БД
<b>FR7</b>	<b>Редагування/видалення коментарів</b>
FR7.1	Вибір коментаря для редагування/видалення
FR7.2	Редагування/цензування/видалення коментаря/оцінки
<b>FR8</b>	<b>Видалення користувача</b>
FR8.1	Вибір користувача для блокування
FR8.2	Видалення користувача

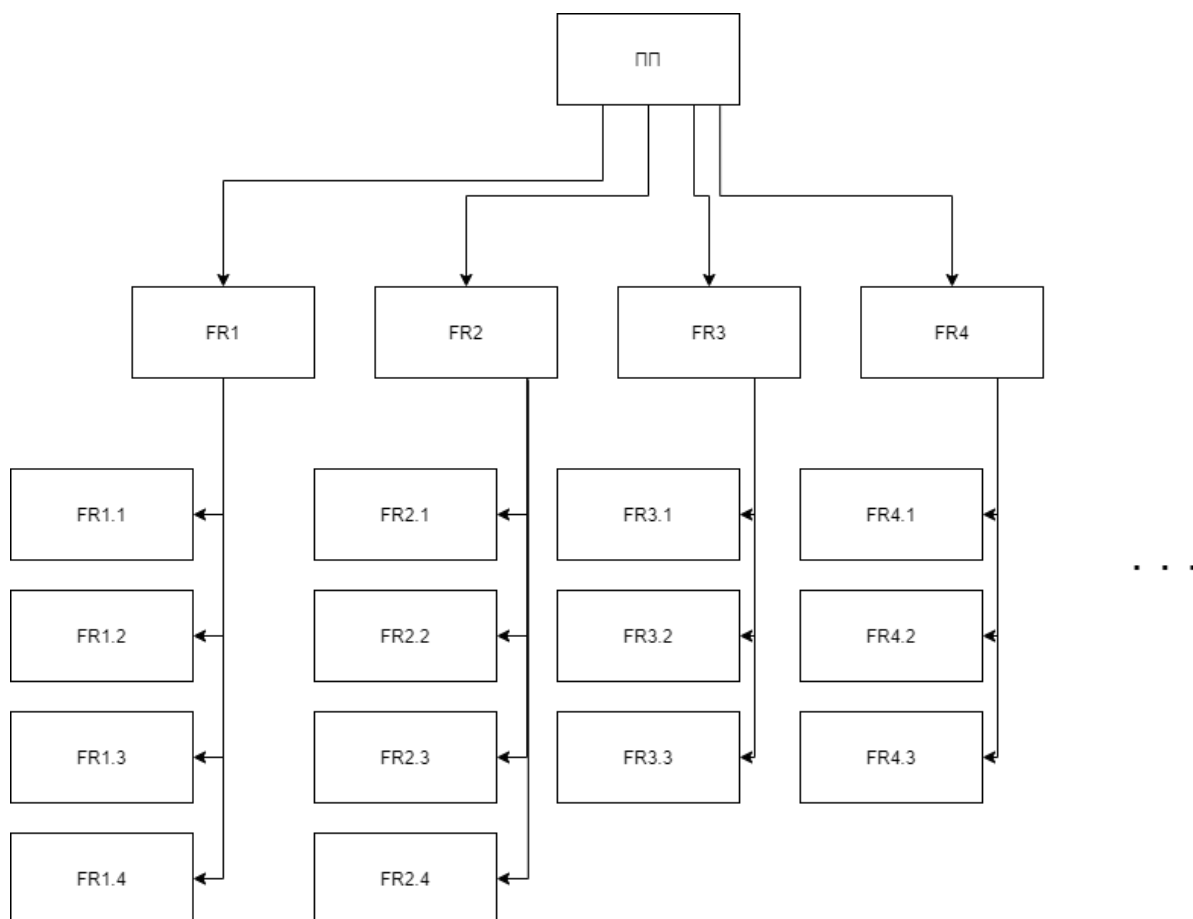


Рис. 1.4.1.1 - WBS-структура багаторівневої класифікації функціональних  
ВИМОГ

#### 1.4.2 Функціональний аналіз існуючих програмних продуктів

Таблиця 1.4.2.1 – Функціональний аналіз існуючих програмних продуктів

Ідентифікатор функції	IMDb	Kinopoisk	RottenTomatoes
FR1	+	+	+
FR2	+	+	+
FR3	+	+	+
FR4	+	+	-
FR5	+	+	+

FR6	-	-	-
FR7	+	+	+
FR8	+	-	-

## 1.5 Нефункціональні вимоги до програмного продукту

### 1.5.1 Опис зовнішніх інтерфейсів

#### 1.5.1.1 Опис інтерфейса користувача

##### 1.5.1.1.1 Опис INPUT-інтерфейса користувача

Таблиця 1.5.1.1.1 – Опис INPUT-інтерфейса користувача

Ідентифікатор функції	Засіб INPUT-потoku	Особливості використання
FR1	Маніпулятор типу миша, клавіатура	Використання лівої кнопки миші для завершення процесу вводу даних
FR2	Маніпулятор типу миша, клавіатура	Використання лівої кнопки миші для завершення процесу вводу даних
FR3	Маніпулятор типу миша, клавіатура	Введення назви за допомогою клавіатури, початок пошуку через ліву кнопку миші

FR4	2/3-кнопочний маніпулятор типу "миша"	Використання лівої кнопки миші для вибору критеріїв
FR5	Маніпулятор типу миша, клавіатура	Написання та вибір оцінки, підтвердження вводу
FR6	Маніпулятор типу миша, клавіатура	Введення даних про фільм та підтвердження вводу
FR7	Маніпулятор типу миша, клавіатура	Вибір лівою кнопкою миші
FR8	2/3-кнопочний маніпулятор типу "миша"	Вибір лівою кнопкою миші

#### 1.5.1.1.2 Опис OUTPUT-інтерфейса користувача

Результат аналізу засобів OUTPUT-потоків представлені у таблиці 1.5.1.1.2

Таблиця 1.5.1.1.2

Ідентифікатор функції	Засіб OUTPUT- потоків	Особливості використання
FR1	Графічний інтерфейс	Рис. 1.5.1
FR2	Графічний інтерфейс	Рис. 1.5.2
FR3	Графічний інтерфейс	Рис. 1.5.3

FR4	Графічний інтерфейс	Рис. 1.5.4
FR5	Графічний інтерфейс	Рис. 1.5.5
FR6	Графічний інтерфейс	Рис. 1.5.6
FR7	Графічний інтерфейс	Рис. 1.5.7
FR8	Графічний інтерфейс	Рис. 1.5.8

### New User Details

---

<input type="text" value="Username"/>	<input type="text" value="Email"/>
<input type="text" value="First Name"/>	<input type="text" value="Last Name"/>

☒ Send email confirmation

---

<input type="button" value="Add User"/>	<input type="button" value="Cancel"/>
---	---------------------------------------

Рис. 1.5.1 – Мокери функції FR1

The mockup shows a 'Sign In' form with a title 'Sign In' in bold. Below it is a greeting 'Hi there! Nice to see you again.' in a lighter font. There are two input fields: 'Email' containing 'example@email.com' and 'Password' which is masked with dots. To the right of the password field is an eye icon for toggling visibility. At the bottom is a large grey button labeled 'Sign In'.

Рис. 1.5.2 – Москир функції FR2

## ПОИСК ФИЛЬМА

The mockup shows a search form with the title 'ПОИСК ФИЛЬМА' in bold. Below the title is a text input field with the placeholder 'Введите название...' and a dark button with a red magnifying glass icon.

Рис. 1.5.3 – Москир функції FR3

## Жанры

- ☐ Аніме
- ☐ Драма
- ☐ Вестерн
- ☐ Боевики
- ☐ Комедии
- ☐ Ужасы

Рис. 1.5.4 – Москир функції FR4

### <sup>1</sup> ОСТАВИТЬ ОТЗЫВ

Ваш комментарий \*

Отправить

Рис. 1.5.5 – Москир функції FR5



Add film

Name:

Description:

Actors:

Date published:

Date:

Today

Time:

Now

Note: You are 3 hours ahead of server time.

Save and add another

Save and continue editing

SAVE

Рис. 1.5.6 – Москир функції FR6

Select comment to change

ADD COMMENT +

Action:

-----

Go

0 of 1 selected

☐

COMMENT

☐

Comment object (1)

1 comment

Рис. 1.5.7 – Москир функції FR7

Select user to change

Q

Search

Action:

-----

Go

1 of 2 selected

☐

USE

Delete selected users

☐

admin

admin@example.com

✓

☒

user

✗

2 users

Рис. 1.5.8 – Москир функції FR8

### 1.5.1.2 Опис інтерфейсу із зовнішніми пристроями

Ідентифікатор функції	Зовнішній пристрій
-----------------------	--------------------

					ІС КР 122 АІ181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			24

FR1	Desktop, Notebook
FR2	Desktop, Notebook
FR3	Desktop, Notebook
FR4	Desktop, Notebook
FR5	Desktop, Notebook
FR6	Desktop, Notebook
FR7	Desktop, Notebook
FR8	Desktop, Notebook

#### 1.5.1.3 Опис програмних інтерфейсів

Для доступу до сервісу, так як це буде веб-додаток, достатнім буде пристрій з наявністю стабільної ОС(Windows, Linux, MacOS) з доступом до мережі-інтернет.

#### 1.5.1.4 Опис інтерфейсів передачі інформації

Дротові інтерфейси:

- Ethernet

Бездротові інтерфейси:

- Wi-Fi

#### 1.5.1.5 Опис атрибутів продуктивності

Ідентифікатор функції	Максимальний час реакції ПП на дії користувачів, секунди
FR1	1

FR2	1
FR3	3
FR4	3
FR5	2
FR6	2
FR7	1
FR8	1

## 2 Планування процесу розробки програмного продукту

### 2.1 Планування ітерацій розробки програмного продукту

З метою забезпечення для вимог таких рекомендацій IEEE-стандарту, як необхідність, корисність при експлуатації, здійсненність функціональних вимог до ПП, були визначені функціональні пріоритети, які будуть використані при плануванні ітерацій розробки ПП. При створенні пріоритетів були враховані:

- сценарні залежності між прецедентами, до яких належать функції, на основі аналізу пунктів передумов початку роботи прецедентів, вказаних в описі сценаріїв роботи прецедентів;

- вплив роботи прецеденту, до якого належить функція, на досягнення мети ПП у відсотках, на основі аналізу пунктів гарантій успіху, вказаних в описі сценаріїв роботи прецедентів.

Сценарні залежності будуть перетворені у відповідні функціональні залежності.

Вплив роботи прецеденту буде поширено на всі підлеглі функції ієрархії. При визначенні пріоритетів рекомендується використовувати наступні позначки:

- M (Must) – функція повинна бути реалізованою у перших ітераціях за будь-яких обставин;

- S (Should) – функція повинна бути реалізованою у перших ітераціях, якщо це взагалі можливо;

- C (Could) – функція може бути реалізованою, якщо це не вплине негативно на строки розробки;

- W (Want) – функція може бути реалізованою у наступних ітераціях.

Опис представлено в таблиці 2.1.

Таблиця 2.1 – Опис функціональних пріоритетів

Ідентифікатор функції	Функціональні залежності	Вплив на досягнення мети, %	Пріоритет функції
<b>FR1</b>	-	<b>20</b>	<b>M</b>
FR1.1	-	-	-
FR1.2	-	-	-
FR1.3	-	-	-
<b>FR2</b>	<b>FR1.1</b>	<b>20</b>	<b>M</b>
FR2.1	FR1.1	-	-
FR2.2	FR1.1	-	-
FR2.3	FR1.1	-	-
FR2.4	FR1.1	-	-
<b>FR3</b>	<b>FR2</b>	<b>15</b>	<b>S</b>
FR3.1	FR2	-	-
FR3.2	FR2	-	-
FR3.3	FR2	-	-

<b>FR4</b>	<b>FR2</b>	<b>15</b>	<b>S</b>
FR4.1	FR2	-	-
FR4.2	FR2	-	-
FR4.3	FR2	-	-
<b>FR5</b>	<b>FR2</b>	<b>0</b>	<b>W</b>
FR5.1	FR2	-	-
FR5.2	FR2	-	-
<b>FR6</b>	<b>FR2</b>	<b>20</b>	<b>M</b>
FR6.1	FR2	-	-
FR6.2	FR2	-	-
<b>FR7</b>	<b>FR2</b>	<b>5</b>	<b>C</b>
FR7.1	FR2	-	-
FR7.2	FR2	-	-
<b>FR8</b>	<b>FR2</b>	<b>5</b>	<b>C</b>
FR8.1	FR2	-	-
FR8.2	FR2	-	-

## 2.2 Концептуальний опис архітектури програмного продукту

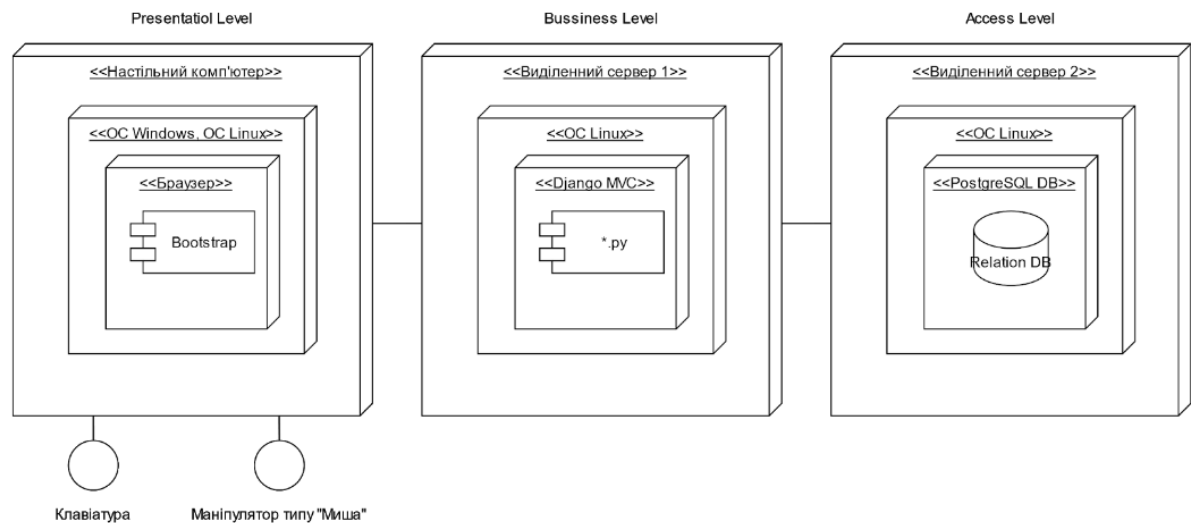


Рис. 2.2.1 – Концептуальний опис архітектури програмного продукту

## 2.3 План розробки програмного продукту

### 2.3.1 Оцінка трудомісткості розробки програмного продукту

Для оцінки трудомісткості продукту була обрана методика Use Case Point, яка має наступні кроки.

1. Визначення нескорегованого показника UUCP (Unadjusted Use Case Points)

Таблиця 2.3.1.1 – «Вагові коефіцієнти акторів»

Тип Актора	Ваговий коефіцієнт
Простий – Гість	1
Середній – Авторизований користувач	2
Складний - Адмін	3

Таблиця 2.3.1.2 – «Вагові коефіцієнти прецедентів»

Тип прецедента	Кількість кроків сценарію	Ваговий коефіцієнт
Простий	1-2	5
Середній	3	10
Складний	4	15

$$UUCP = A + UC = 6 + 70 = 76$$

## 2. Визначення технічної складності проекту

Таблица 2.3.1.3 – «Технічна складність проекту»

Показник	Опис показника	Вага	Присвоєне значення
T1	Распределенная система	2	2
T2	Высокая производительность (пропускная способность)	1	2
T3	Работа конечных пользователей в режиме он-лайн	1	1
T4	Сложная обработка данных	1	2
T5	Повторное использование кода	1	1
T6	Простота установки	0,5	2
T7	Простота использования	0,5	1
T8	Переносимость	1	2
T9	Простота внесения изменений	1	2



T10	Параллелизм	1	1
T11	Специальные требования к безопасности	1	3
T12	Непосредственный доступ к системе со стороны внешних пользователей	1	1
T13	Специальные требования к обучению пользователей	1	1

$$TCF = 0,6 + (0,01 * (ST_i * Вага_i)) = 0,6 + (0,01 * 27,5) = 0,87$$

### 3. Визначення рівня кваліфікації розробників

Таблица 2.3.1.4 – Визначення рівня кваліфікації розробників

Показник	Опис показника	Вага	Присвоєне значення
F1	Знакомство с технологией	1,5	3
F2	Опыт разработки приложений	0,5	1
F3	Опыт использования объектно-ориентированного подхода	1	3
F4	Наличие ведущего аналитика	0,5	0
F5	Мотивация	1	4
F6	Стабильность требований	2	4
F7	Частичная занятость	-1	2
F8	Сложные языки программирования	-1	3

$$EF = 1,4 + (-0,03 * (SF_i * Barai)) = 1.4 + (-0.03 * 15) = 0.95$$

#### 4. Остаточне значення UCP (Use Case Points)

$$UCP = UUCP * TCF * EF = 76 + 0,87 + 0,95 = 77,82$$

#### 5. Оцінка трудомісткості проекту

Показників F1 - F6, які мають значення менше 3 – 2

Показників F7 - F8, які мають значення більше 3 – 0

Отже слід використовувати 20 люд.-год на одну UCP

#### 2.3.2 Визначення дерева робіт з розробки програмного продукту

При створенні дерева робіт (Work BreakDown Structure- WBS) використовується дерево функцій, яке було створено раніше.

Кожна функція 1-го рівня ієрархії перетворюється в Work Package (WP).

Кожна функція 2-го рівня ієрархії перетворюється в Work Task (WT).

Для кожної задачі визначаються підзадачі - Work SubTask (WST) з урахуванням базових процесів розробки програмних модулів: проектування, конструювання, модульне тестування, збірка та системне тестування (Рис. 2.3.2.1).

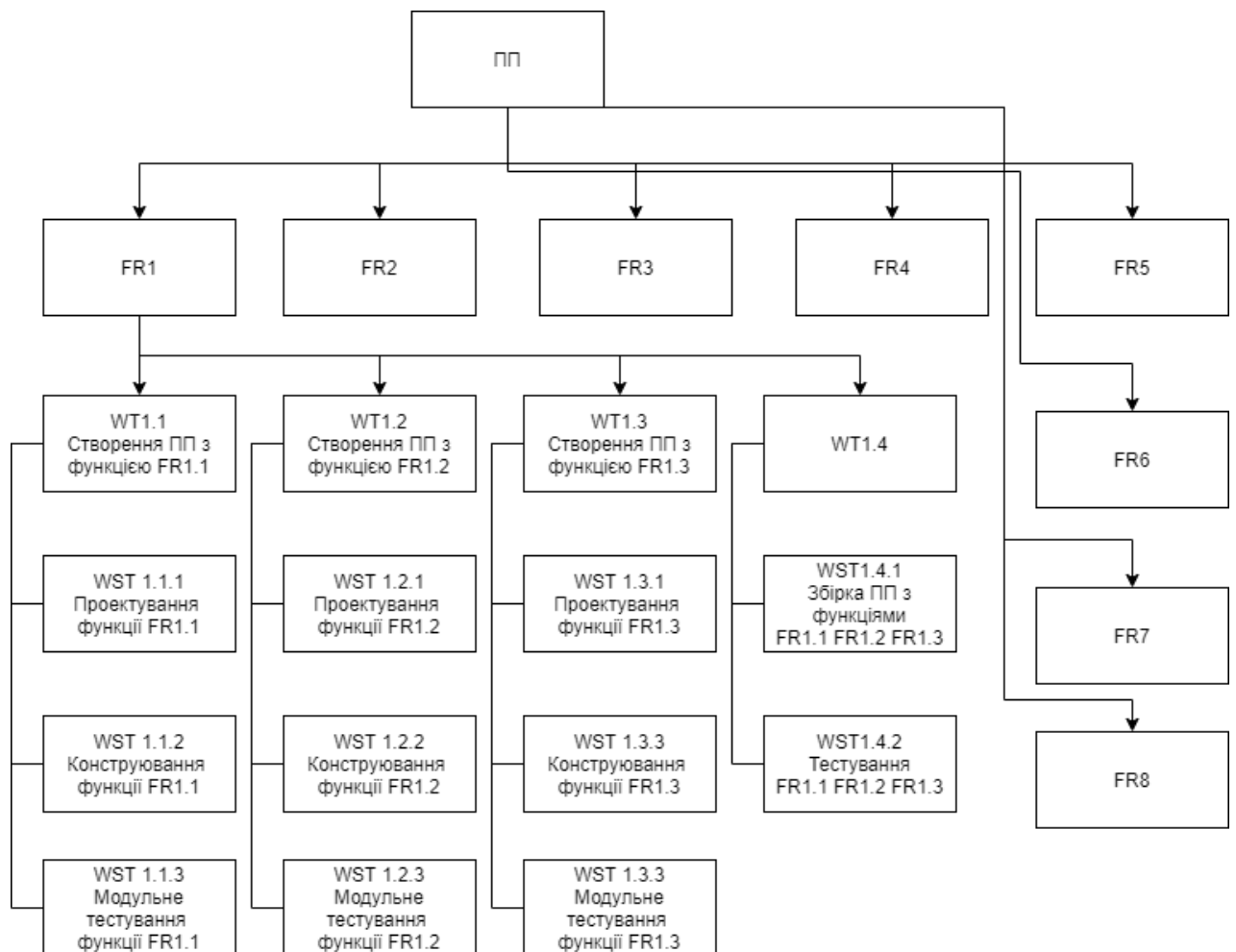


Рис. 2.3.2.1 – Дерево робіт

### 2.3.3 Графік робіт з розробки програмного продукту

#### 2.3.3.1 Таблиця з графіком робіт

Таблиця 2.3.3.1 - Таблиця з графіком робіт

Підзадача	Дата початку	Дні	Дата кінця	Виконавець
WST1.1.1	15.10.2020	1	15.10.2020	Олійник В. М.
WST1.1.2	16.10.2020	1	16.10.2020	Олійник В. М.
WST1.1.3	17.10.2020	1	17.10.2020	Олійник В. М.
WST2.1.1	18.10.2020	1	18.10.2020	Олійник В. М.
WST2.1.2	19.10.2020	1	19.10.2020	Олійник В. М.
WST2.1.3	20.10.2020	1	20.10.2020	Олійник В. М.

WST3.1.1	21.10.2020	2	22.10.2020	Совяк А.І.
WST3.1.2	23.10.2020	2	24.10.2020	Совяк А.І.
WST3.1.3	25.10.2020	2	25.10.2020	Совяк А.І.
WST4.1.1	27.10.2020	1	27.10.2020	Совяк А.І.
WST4.1.2	28.10.2020	1	28.10.2020	Совяк А.І.
WST4.1.3	29.10.2020	1	29.10.2020	Совяк А.І.
WST6.1.1	15.10.2020	2	16.10.2020	Совяк А.І.
WST6.1.2	17.10.2020	2	18.10.2020	Совяк А.І.
WST6.1.3	19.10.2020	2	20.10.2020	Совяк А.І.
WST7.1.1	15.10.2020	2	16.10.2020	Пшеничний А.О.
WST7.1.2	17.10.2020	2	18.10.2020	Пшеничний А.О.
WST7.1.3	19.10.2020	2	20.10.2020	Пшеничний А.О.
WST8.1.1	15.10.2020	1	15.10.2020	Пшеничний А.О.
WST8.1.2	16.10.2020	1	16.10.2020	Пшеничний А.О.
WST8.1.3	17.10.2020	1	17.10.2020	Пшеничний А.О.

### 2.3.3.2 Діаграма Ганта

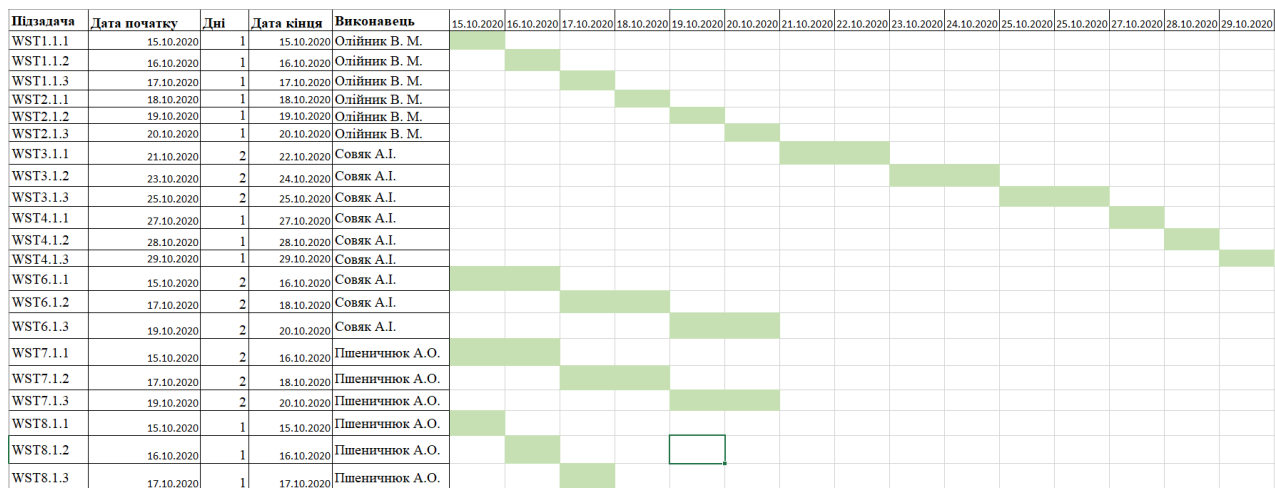


Рис. 2.3.3.2 – Діаграма Ганта

### 3 Проектування програмного продукту

#### 3.1 Концептуальне та логічне проектування структур даних програмного продукту

##### 3.1.1 Концептуальне проектування на основі UML-діаграми концептуальних класів

Використовуючи кроки основного успішного та альтернативного сценаріїв роботи прецедентів ПП, було спроектовано UML-діаграми концептуальних класів (рис. 3.1.1).

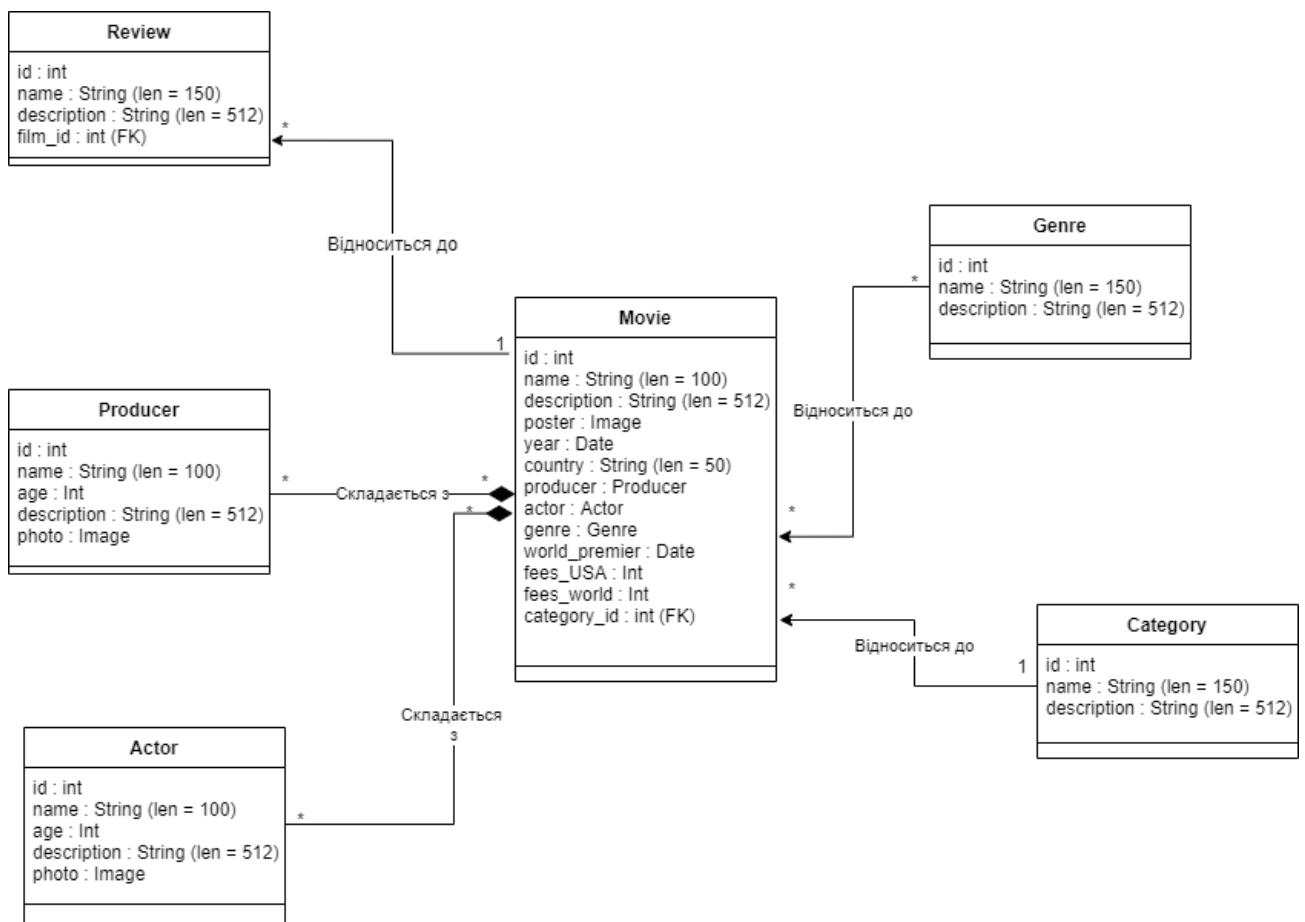


Рис. 3.1.1

### 3.1.2 Логічне проектування структур даних

UML-діаграма концептуальних класів була перетворена в опис структур даних з використанням моделі, яка була обрана в концептуальному описі архітектури ПП (рис. 3.1.2).

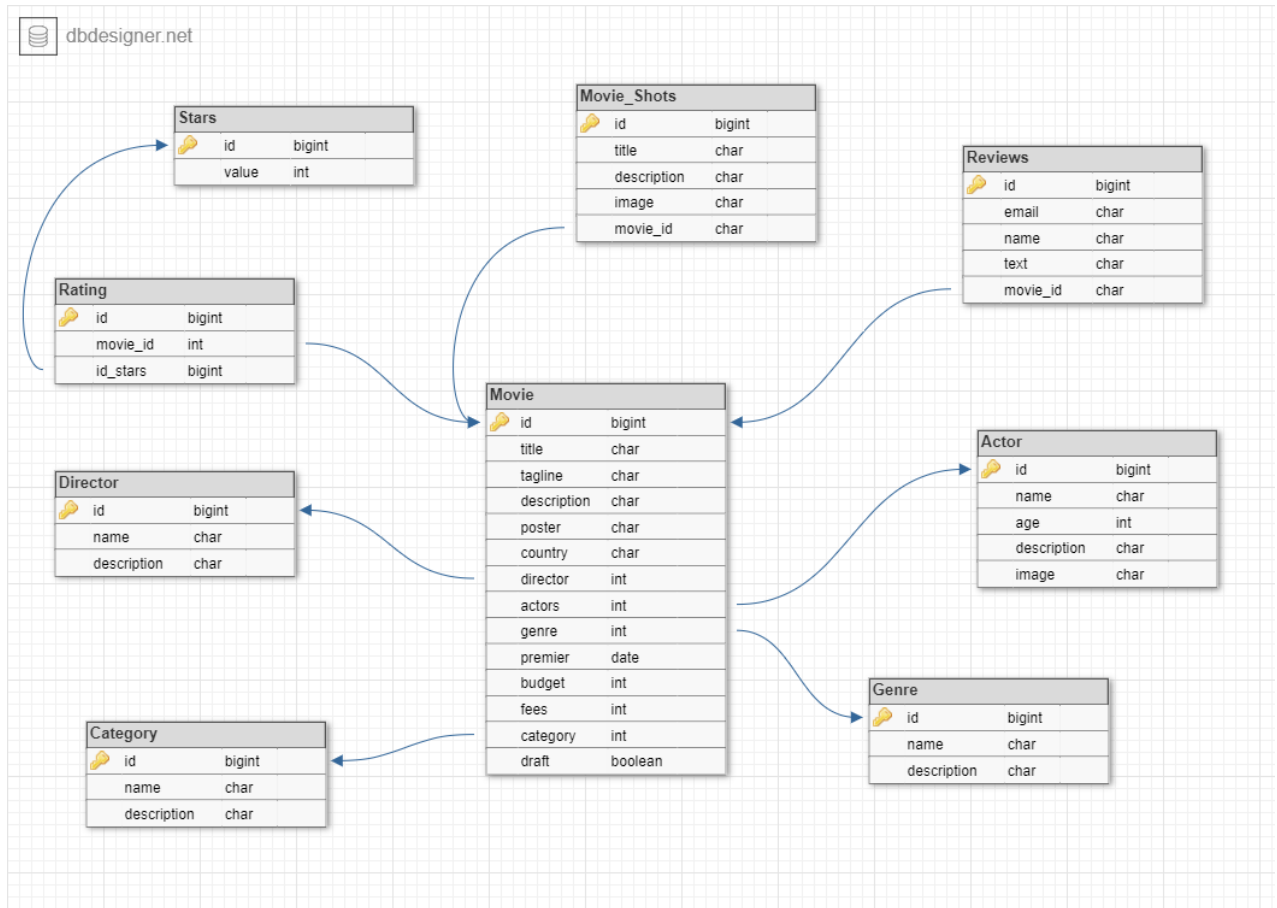


Рис. 3.1.2 – Схема БД

### 3.2 Проектування програмних класів

На основі UML-діаграми концептуальних класів були спроектовані програмні класи:

- англійські або транслітерацію україномовних назви класів та їх атрибутів;
- абстрактні класи, їх класи-нащадки та інші класи;

- зв'язки між класами (наслідування, іменована асоціація, агрегатна асоціація, або агрегація, композитна асоціація або композиція) та їх кратності;
- атрибути класів с типами даних (цілий, дійсний, логічний, перелічуваний, символьний з урахуванням розміру), та типом видимості (публічний, захищений, приватний);
- методи-конструктори ініціалізації екземплярів об'єктів класу, set методи та get-методи для доступу до атрибутів класу

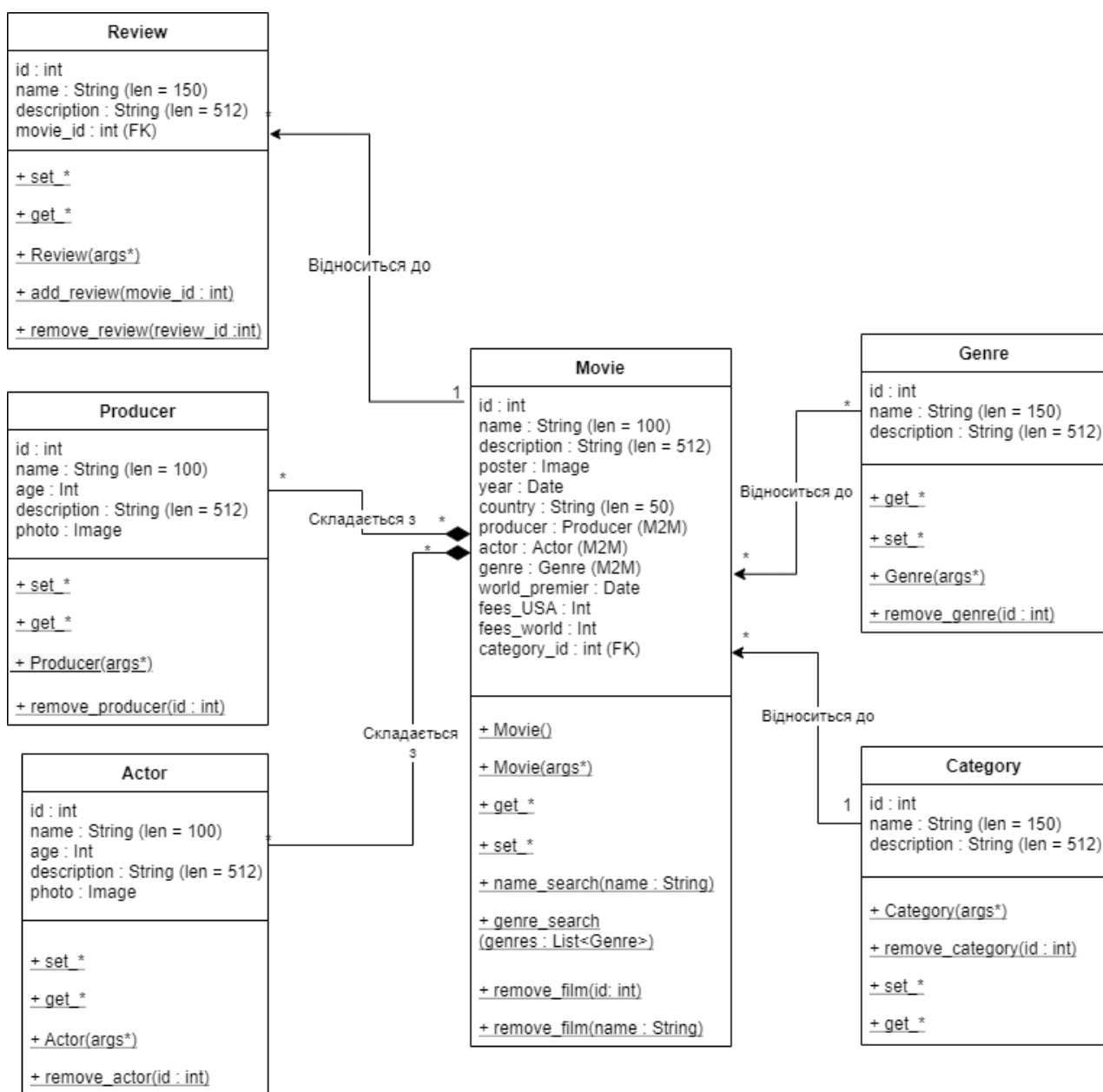


Рис. 3.2.1 – UML-діаграма класів

### 3.3 Проектування алгоритмів роботи методів програмних класів

#### UML-код реєстрації користувача

@startuml

title User.Registration(username, email, password, confirmpassword)

start

repeat

:Вывод экранной формы для регистрации пользователя;

note right

Мокап экранной формы FR1 представлен

в разделе "Описание OUTPUT-интерфейса пользователя"

end note

:Ввод пользователем собственных данных;

:Проверка корректности введенных данных;

if (Некорректное введение данных) then (да);

:Информирование о введении некорректных данных;

else (нет)

:Сохранение данных в БД;

note right

INSERT INTO users

VALUES(username, email, password)

end note

:Авторизация пользователя;

:Перенаправление на начальную страницу;

stop

@enduml



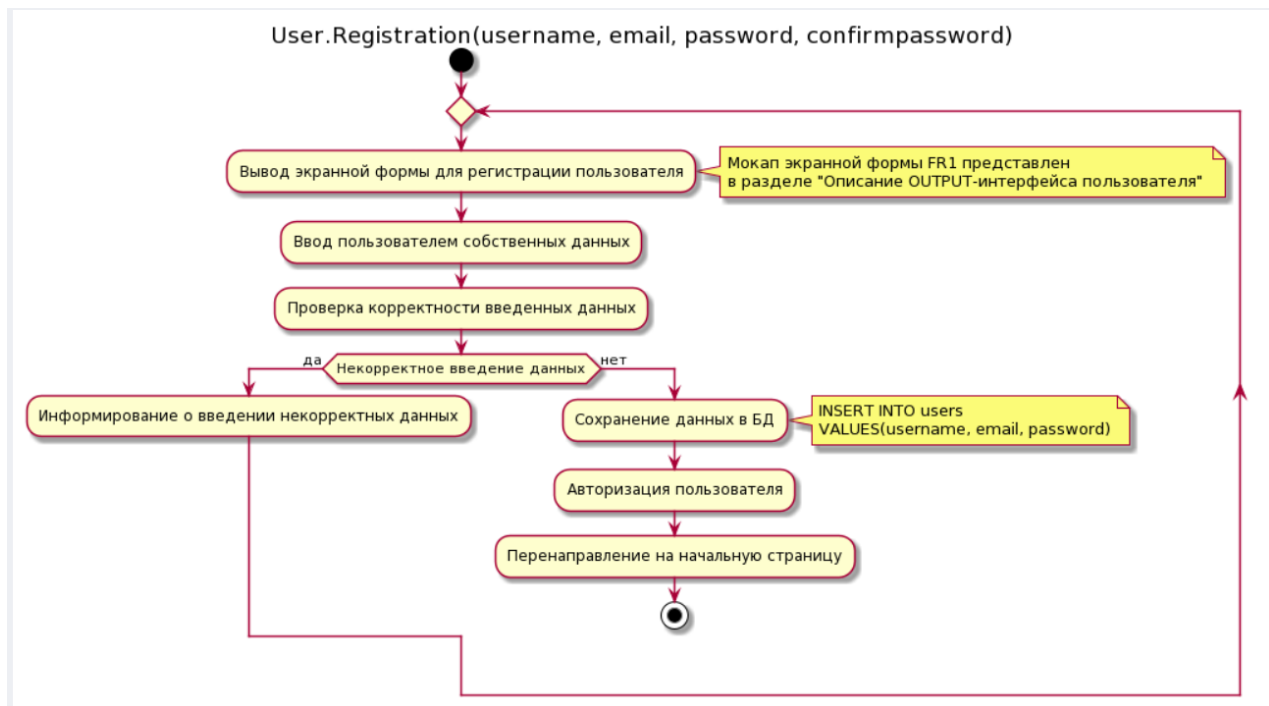


Рис. 3.3.5 - Диаграмма методу User.Registration()

### UML-код авторизації користувача

@startuml

title User.Login(username, password)

start

repeat

:Вывод экранной формы для авторизации пользователя;

note right

Мокап экранной формы FR2 представлен

в разделе "Описание OUTPUT-интерфейса пользователя"

end note

:Ввод пользователем данных авторизации;

:Проверка корректности введенных данных;

if (Некорректное введение данных) then (да);

:Информирование о введении некорректных данных;

else (нет)

:Авторизация пользователя;

:Перенаправление на начальную страницу;

stop

@endum1

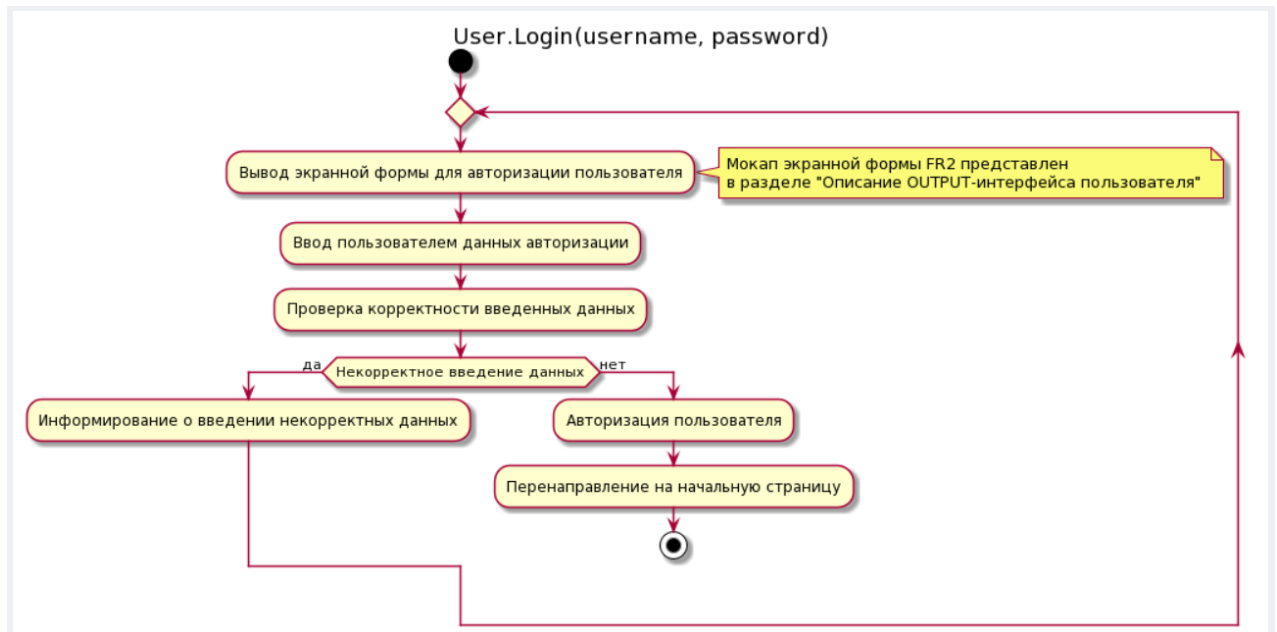


Рис. 3.3.6 – Диаграмма метода User.Login()

### 3.4 Проектування тестових наборів методів програмних класів

Назва функції	№ тесту	Опис значень вхідних даних	Опис очікуваних значень результату
User.registration(nickname, email, password, confirmpassword)	1	Коректний список вхідних даних	Збереження користувача до бази даних
	2	Nickname, nickname>150	Помилка вхідних даних
	3	Email, невірного формату	Помилка вхідних даних

	4	Password, password>50	Помилка вхідних даних
	5	Password, password != confirmpassword	Помилка вхідних даних
User.login(username, password)	1	Коректна комбінація логіну та паролю	Аутентифікація користувача
	2	Відсутність користувача з відповідною комбінацією логіну та паролю	Помилка вхідних даних

## 4 Конструювання програмного продукту

### 4.1 Особливості конструювання структур даних

#### 4.1.1 Особливості інсталяції та роботи з СУБД

Для розробки проекту була використана реляційна БД – PostgreSQL 13 версії, яка була встановлена на локальні комп'ютери. Для деплою була використана хмарна СУБД Heroku PostgreSQL, з якою робота відбувається лише через графічний інтерфейс у браузері.

#### 4.1.2 Особливості створення структур даних

Після налаштування підключення до БД, запити створення таблиць формуються автоматично за допомогою Django на основі написаних класів та за необхідності таблиці оновлюються при зміні програмного коду – також автоматично.

```
class Movie(models.Model):
    """Фільм"""
    title = models.CharField("Название", max_length=100)
    tagline = models.CharField("Слоган", max_length=100, default='')
    description = models.TextField("Описание")
    poster = models.ImageField("Постер", upload_to="movies/")
    year = models.PositiveSmallIntegerField('Дата выхода', default=2020)
    country = models.CharField("Страна", max_length=30)
    directors = models.ManyToManyField(Actor, verbose_name="режиссер", related_name='film_director')
    actors = models.ManyToManyField(Actor, verbose_name="актеры", related_name='film_actor')
    genres = models.ManyToManyField(Genre, verbose_name="Жанры")
    world_premiere = models.DateField("Премьера в мире", default=date.today)
    budget = models.PositiveIntegerField("Бюджет", default=0, help_text='указывать сумму в долларах')
    fees_in_usa = models.PositiveIntegerField(
        "Сборы в США", default=0, help_text='указывать сумму в долларах'
    )
    fees_in_world = models.PositiveIntegerField(
        "Сборы в мире", default=0, help_text='указывать сумму в долларах'
    )
    category = models.ForeignKey(
        Category, verbose_name="Категория", on_delete=models.SET_NULL, null=True
    )
    url = models.SlugField(max_length=160, unique=True)
    draft = models.BooleanField("Черновик", default=False)
```

Рис. 4.1 – Клас “Movie”

Створення таблиці на основі класу Movie (рис. 1) буде відбуватися за допомогою спеціальних атрибутів, які сформують поля БД автоматично на основі описаних типів.

Наприклад:

- Атрибут `models.CharField` сформує звичайне поле типу `Char` з указаним розміром
- Атрибут `models.TextField` формують текстове поле
- Атрибут `models.ImageField` формують поле типу `Char` з прописаною адресою до теки з зображенням (в нашому випадку: `movies/`)
- Атрибут `models.PositiveSmallInteger` сформує поле типу `Int` з обмеженням на лише невід'ємність значення цього поля. Діапазон значень - з 0 до 32767
- Атрибут `models.ManyToManyField` формують зв'язок багато-до-багатьох з сутністю, указаною в цьому атрибуті, наприклад `actors = models.ManyToManyField(ACTOR)` сформує зв'язок з сутністю актора
- Атрибут `models.DateField` сформує поле типу дата
- Атрибут `models.PositiveIntegerField` сформує поле типу `Int` з обмеженням на лише невід'ємність значення цього поля
- Атрибут `models.ForeignKey(<сутність>)` сформує поле типу `int` з зовнішнім ключом - зв'язком з сутністю, вказаною у параметрах
- Атрибут `models.BooleanField` сформує поле логічного типу даних, яке може приймати лише `True` або `False`
- Атрибут `models.Slug` - формують поле текстового типу в якому можуть бути лише букви цифри і особливі символи, які сформують адресу екземпляру класу у браузері.

Під'єднання до бд відбувається у файлі налаштування проекту `settings.py` у виді

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'name',
```

```

        'USER': 'django',
        'PASSWORD': 'pass',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

```

Після описання програмних класів створюються так звані міграції командою `python manage.py makemigrations`, що сформує запити створення усіх описаних програмних класів, після внесення усіх бажаних змін, якщо вони є, виконується команда `python manage.py migrate`, що виконає усі запити "міграцій".

## 4.2 Особливості конструювання програмних модулів

### 4.2.1 Особливості роботи з інтегрованим середовищем розробки

Використовувалося середовище програмування PyCharm; інсталяція проводилася з офіційного сайту ([jetbrains.com](http://jetbrains.com)), ліцензія – студентська.

Включає зручні інструменти розробки. Використовувалися фреймворки Django, Bootstrap, psycorg2 – для значного пришвидшення написання програми.

### 4.2.2 Особливості створення програмної структури з урахуванням спеціалізованого Фреймворку

Django створює головні конфігураційні файли, та дозволяє створювати так звані «застосунки» за допомогою команди `python manage.py startapp <appname>`, що створить у проекті нову папку з моделями та налаштуваннями для `<appname>`, потім цей застосунок допишеться до встановлених у головному файлі налаштувань проекту `settings.py`

### 4.2.3 Особливості створення програмних класів

Python у комбінації з Django дозволяє робити моделі зручними за допомогою наслідування класами вбудованої моделі. Розробнику не потрібно писати спеціальні поля типу id. Атрибути класів задаються без обмежень на публічність, та за допомогою models.<тип-атрибуту>.

Django має у своїй бібліотеці багато вбудованих та зручних типів даних, тож додавання типу ImageField для атрибуту «постер» через крапку створить автоматично зручне для взаємодії поле для постера у БД.

### 4.2.4 Особливості розробки алгоритмів методів програмних класів або процедур/функцій

Методи реєстрації та авторизації у Django оперують 2 головними для них фрагментами: Формою (рис. 4.2.2) та методом обробки (рис. 4.2.3 – 4.2.4)

```
class UserRegistrationForm(UserCreationForm):
    email = forms.EmailField(max_length=254, help_text='Обязательно к вводу, введите действующую электронную почту',
                             required=True)

    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')

    def clean_email(self):
        email = self.cleaned_data.get('email')
        user_count = User.objects.filter(email=email).count()

        if not any(i in email for i in ('.ru', '.com', '.ua', '.net')):
            raise forms.ValidationError('Введите корректный адрес')

        if user_count > 0:
            raise forms.ValidationError('Такая электронная почта уже есть в базе данных')
        return email
```

Рис. 4.2.2 – Форма реєстрації

Форма авторизації прописується як окремий клас, який наслідує базову форму авторизації, їй приписуються атрибути – поля, які користувачу потім потрібно буде заповняти. Наприклад, поля username, email, password1, password2 – задаються автоматично, але в цьому випадку ми перевизначили поле email, зробивши його обов’язковим до введення.

Також, ми задали метод класу `clean_mail()`, який викликається одразу після заповнення користувачем інформації. Він перевіряє правопис закінчення електронної пошти та наявність такої пошти уже в БД.

```
def sign_up(request):
    """Метод регистрации.

    Используется, когда пользователь нажимает на кнопку регистрации, в этом случае возвращает форму для
    заполнения. Далее, после введения пользователем желаемых данных, проверяем их на валидность, и, в случае
    корректного введения, сохраняем, создаем сессию и авторизуем пользователя, возвращая его на начальную страницу.
    """
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('index')
        else:
            form = UserRegistrationForm()
    return render(request, 'pages/sign-up.html', {'form': form})
```

Рис. 4.2.3 – Метод обробки процесу реєстрації

На Get запит від користувача він генерує форму для заповнення за допомогою виклику форми напряму :

```
form = UserRegistrationForm()
```

, та повертає сторінку реєстрації. Після введення користувачем на сервер посиляється запит Post, тож метод обробляє форму, яка прийшла за допомогою цього методу, потім перевіряє її на валідність введених даних, якщо дані введено коректно – користувача реєструє у БД та авторизує, перенаправляючи на головну сторінку сайту.

Форма авторизації є влаштованою у фреймворк Django MVC та лише викликається у методі обробки (рис. 4.2.4)



```
def sign_in(request):
    """Метод авторизации в аккаунт.

    Используется, когда пользователь нажимает на кнопку входа, в этом случае вызывается POST запрос
    Метод с помощью определения пользователя по запросу и введенных пользователем в форме авторизации данных,
    проверяет наличие пользователя в базе данных, в случае успеха - создаёт сессию и авторизует его.
    """
    if request.method == 'POST':
        form = AuthenticationForm(request.POST, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect('index')
        else:
            messages.info(request, 'Имя пользователя или пароль не совпадают')
            return render(request, 'pages/sign_in.html', {'form': form})
    else:
        form = AuthenticationForm()

    return render(request, 'pages/sign_in.html', {'form': form})
```

Рис. 4.2.4 – Метод обробки процесу авторизації

На Get запит від користувача він генерує форму для заповнення за допомогою виклику форми напряму :

```
form = AuthenticationForm()
```

, та повертає сторінку авторизації. Після введення користувачем на сервер посилається запит Post, тож метод обробляє форму, яка прийшла за допомогою цього методу, потім перевіряє її на валідність введених даних, якщо дані введено некоректно – користувачу передається повідомлення про некоректне введення даних, в інакшому випадку – авторизує, перенаправляючи на головну сторінку сайту.

#### 4.2.5 Особливості використання спеціалізованих бібліотек та API

Для спрощення розробки програмного продукту було використано готове API взаємодії з БД - psycopg2. API повністю автоматизований, тож для роботи з БД потрібно було лише його встановити

### 4.3 Модульне тестування програмних класів

#### 4.3.1 User.Registration()

User.Registration() являє собою метод, що перевіряє введення користувачем даних та реєструє нового користувача. Метод складається з 2 підметодів: створення форми та маршрутизація запиту. Метод виглядає наступним образом:

```
class UserRegistrationForm(UserCreationForm):
    email = forms.EmailField(max_length=254, help_text='Обязательно к вводу,
введите действующую электронную почту',
                             required=True)

    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')

    def clean_email(self):
        email = self.cleaned_data.get('email')
        user_count = User.objects.filter(email=email).count()

        if not any(i in email for i in ('.ru', '.com', '.ua', '.net')):
            raise forms.ValidationError('Введите корректный адрес')

        if user_count > 0:
            raise forms.ValidationError('Такая электронная почта уже есть в
базе данных')
        return email
```

Клас UserRegistrationForm відповідає за те, які поля будуть у користувача та перевіряє правильність введення та кількість таких електронних в БД.

```
def sign_up(request):
    """Метод регистрации.

    Используется, когда пользователь нажимает на кнопку регистрации, в этом
    случае возвращает форму для
    заполнения. Далее, после введения пользователем желаемых данных,
    проверяя их на валидность, и, в случае
    корректного введения, сохраняет, создает сессию и авторизует
    пользователя, возвращая его на начальную страницу.
    """
    if request.method == 'POST':
        form = UserRegistrationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('index')
    else:
        form = UserRegistrationForm()

    return render(request, 'pages/sign_up.html', {'form': form})
```

Метод для маршрутизації та відстеження запиту користувача на реєстрацію, у випадку запиту GET відправляє користувачу форму для заповнення (рис. 4.3)

Имя пользователя:

Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./+/\_/.

Email:

Обязательно к вводу, введите действующую электронную почту

Пароль:

- Ваш пароль должен содержать как минимум 8 символов.
- Пароль не может состоять только из цифр.

Подтверждение пароля:

Для подтверждения введите, пожалуйста, пароль ещё раз.

[Зарегистрироваться](#)

[Уже есть аккаунт? Войти](#)

Рис. 4.3 – Вікно реєстрації користувача

#### Тест 1 – Користувач не вводить дані

Имя пользователя:

150 символов. Только @/./+/\_/.

! Заполните это поле.

Email:

Рис. 4.4 – Перевірка введення

У випадку, якщо користувач не вводить дані, окремі поля один за одним будуть повідомляти про це користувача.

#### Тест 2 – Користувач вводить не коректні дані:

– Email

					IC KP 122 AI181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			50

Email:


 Недопустимое положение символа "." в адресе "gmail".

Рис. 4.5 – Перевірка введення пошти

Або ж, якщо користувач з такою поштою уже є в бд, він отримає таке повідомлення (рис. 4.6)

Е-mail.

Обязательно к вводу, введите действующую электронную почту

Такая электронная почта уже есть в базе данных

Рис. 4.6 – Перевірка введення даних

#### – Пароль

При введенні некоректного паролю користувач отримає такі повідомлення (рис. 4.7 – 4.8)

Пароль:

- Ваш пароль должен содержать как минимум 8 символов.
- Пароль не может состоять только из цифр.

Подтверждение пароля:

Для подтверждения введите, пожалуйста, пароль ещё раз.

Введенные пароли не совпадают.

Рис. 4.7

Пароль:

- Ваш пароль должен содержать как минимум 8 символов.
- Пароль не может состоять только из цифр.

Подтверждение пароля:

Для подтверждения введите, пожалуйста, пароль ещё раз.

Введённый пароль слишком короткий. Он должен содержать как минимум 8 символов.

Введённый пароль состоит только из цифр.

Зарегистрироваться

Уже есть аккаунт? [Войти](#)

Рис. 4.8

У випадку, коли помилок немає, користувача зареєструє, авторизує, та перенаправляє на головну сторінку

#### 4.3.2 User.Login()

Являє собою метод, що перевіряє введення користувачем даних та авторизує його. Метод складається з 2 підметодів: створення форми та маршрутизація запиту. Метод виглядає наступним образом: Форма, яка була використана з фреймворку Django, та клас UserRegistrationForm відповідає за те, які поля будуть у користувача та перевіряє правильність введення та кількість таких електронних в БД.

```
def sign_in(request):  
    """Метод авторизации в аккаунт.  
  
    Используется, когда пользователь нажимает на кнопку входа, в этом  
    случае вызывается POST запрос  
    Метод с помощью определения пользователя по запросу и введенных  
    пользователем в форме авторизации данных,  
    проверяет наличие пользователя в базе данных, в случае успеха - создаёт  
    сессию и авторизует его.  
    """  
    if request.method == 'POST':
```

					IC KP 122 AI181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			52

```

form = AuthenticationForm(request.POST, data=request.POST)
if form.is_valid():
    user = form.get_user()
    login(request, user)
    return redirect('index')
else:
    messages.info(request, 'Имя пользователя или пароль не совпадают')
    return render(request, 'pages/sign_in.html', {'form': form})
else:
    form = AuthenticationForm()

return render(request, 'pages/sign_in.html', {'form': form})

```

Метод для маршрутизації та відстеження запиту користувача на реєстрацію, у випадку запиту GET відправляє користувачу форму для заповнення (рис. 4.8)

Имя пользователя:

Пароль:

Войти

Ещё нет аккаунта?  
[Зарегистрироваться](#)

Рис. 4.8 – Вікно входу користувача

Имя пользователя:

Заполните это поле.

Войти

Ещё нет аккаунта?  
[Зарегистрироваться](#)

Рис. 4.9 – Помилки, якщо користувач не вводить дані

Пароль:

Имя пользователя или  
пароль не совпадают

Войти

Рис. 4.10 – Помилка у випадку ненаявності такого користувача в БД

## 5 Розгортання та валідація програмного продукту

### 5.1 Інструкція з встановлення програмного продукту

В підрозділі «2.2 Концептуальний опис архітектури програмного продукту» було представлено UML-діаграму розгортання ПП на трьох рівнях (PL,BL,AL)

В якості презентаційного рівня використовується будь-який браузер користувача із доступом до мережі інтернет, користувачу лише необхідно перейти за адресою: <https://what-to-watch-sop.herokuapp.com>

В якості другого бізнес-рівня була обрана платформа Heroku (рис. 5.1.1), яка дозволяє безкоштовно хостити власні продукти.

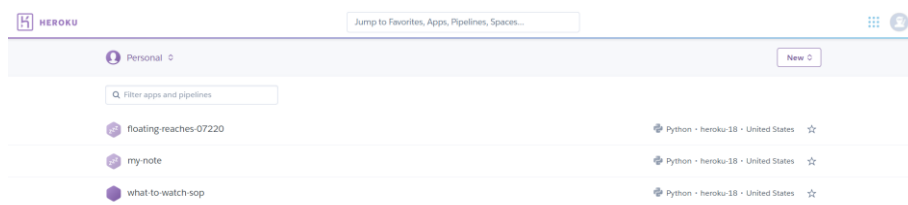


Рис. 5.1.1 – Хостинг ПП на Heroku

Heroku надає безкоштовний тариф на хостинг з обмеженнями на об'єм хостингового ПП, а точніше – 512мб, та режим «Завжди увімкнено» - у безкоштовному тарифі ПП вимикається через пів години, якщо ним не користуватись.

Для розгортання Python Django веб-додатку необхідно:

1. Встановити додаткові бібліотеки:

- gunicorn (HTTP шлюзовий інтерфейс для Python)
- dj-database-url (Бібліотека автоматичного підключення до БД, яка бере налаштування з змінних оточення, які налаштовуються або окремо, або, як у випадку з БД Postgres – автоматично)



- boto3 (для налаштування мосту між файловим сервером та сервером з ПП)
- django-storages (для конфігурації підключення ПП до файлового серверу)
- whitenoise (для налаштування static-файлів для деплою)

## 2. Дописати налаштування для деплою

У головний конфігураційний файл settings.py дописуються строки:

До проміжного програмного забезпечення дописується:

```
MIDDLEWARE = [
    'whitenoise.middleware.WhiteNoiseMiddleware',
]
```

Що включає роботу whitenoise

```
db_from_env = dj_database_url.config()
DATABASES['default'].update(db_from_env)
```

Що дозволяє ПП автоматично під'єднатись до БД Heroku Postgres.

```
DEBUG = False
```

Що вмикає режим налагодження.

```
AWS_ACCESS_KEY_ID = os.environ.get('AWS_ACCESS_KEY_ID')
AWS_SECRET_ACCESS_KEY = os.environ.get('AWS_SECRET_ACCESS_KEY')
AWS_S3_ADDRESSING_STYLE = os.environ.get('AWS_S3_ADDRESSING_STYLE')
AWS_STORAGE_BUCKET_NAME = os.environ.get('AWS_STORAGE_BUCKET_NAME')
AWS_S3_FILE_OVERWRITE = False
AWS_DEFAULT_ACL = None
DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'
AWS_S3_REGION_NAME = 'eu-central-1'
AWS_S3_SIGNATURE_VERSION = 's3v4'
```

Що додає налаштування на підключення ПП до хмарного файлового сервісу AWS Bucket.

```
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

Що прописує шлях до статичних файлів, які будуть використовуватись Heroku за допомогою whitenoise

## 3. Створити конфігураційні файли

Створюється файл Procfile, який містить у собі :

					ІС КР 122 АІІ81 ПЗ	56
Зм.	Арк.	№ докум.	Підп.			

```
web: gunicorn tsppfinal.wsgi --log-file -
```

Що відображає список процесів, які будуть виконані для старту веб-додатки, в нашому випадку, лише наш проект.

Також створюється файл runtime.txt з наступним змістом:

```
python-3.8.6
```

Вказує на версію мови програмування Python, яка повинна використовуватись для ПП

За допомогою команди `python pip freeze > requirements.txt` з переліком абсолютно усіх пакетів, які використовує ПП. Зміст:

```
appdirs==1.4.4
asgiref==3.3.1
boto==2.49.0
boto3==1.16.34
botocore==1.19.34
certifi==2020.11.8
cffi==1.14.4
chardet==3.0.4
cryptography==3.2.1
cyclr==0.10.0
defusedxml==0.6.0
distlib==0.3.1
dj-database-url==0.5.0
Django==3.1.3
django-allauth==0.44.0
django-ckeditor==6.0.0
django-js-asset==1.2.2
django-recaptcha3==0.4.0
django-storages==1.10.1
```

					IC KP 122 AI181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			57

```

filelock==3.0.12
gunicorn==20.0.4
idna==2.10
jmespath==0.10.0
joblib==0.17.0
kiwisolver==1.3.1
lxml==4.6.2
oauthlib==3.1.0
Pillow==8.0.1
psycpg2==2.8.6
pysparser==2.20
PyJWT==1.7.1
pyparsing==2.4.7
python-dateutil==2.8.1
python3-openid==3.2.0
pytz==2020.4
requests==2.25.0
requests-oauthlib==1.3.0
s3transfer==0.3.3
six==1.15.0
sqlparse==0.4.1
threadpoolctl==2.1.0
urllib3==1.26.2
virtualenv==20.2.0
whitenoise==5.2.0

```

Він використовується системою Heroku під час деплою на сервіс, щоб сервер знав, які компоненти йому необхідно встановити і якої версії для того, щоб ПП коректно працював.

4. Зібрати усі стилі, застосовані в графічному інтерфейсі користувача

					ІС КР 122 АІ181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			58

Процес відбувається за допомогою `python manage.py collectstatic`, що збирає усі статичні стилі/фото та формує папку `staticfiles`, яка використовується Heroku.

5. Зареєструватися/увійти до акаунту Heroku та створити додаток

У терміналі в репозиторії проекту:

```
heroku login  
heroku create <my-app>
```

6. Налаштувати змінні оточення, якщо такі є

```
heroku config:set AWS_ACCESS_KEY_ID=<your_key_id>  
heroku config:set AWS_SECRET_ACCESS_KEY =<your_key>  
heroku config:set AWS_S3_ADDRESSING_STYLE =<your_style>  
heroku config:set AWS_STORAGE_BUCKET_NAME =<your_bucket_name>
```

7. Задеплоїти ПП

У терміналі в репозиторії проекту:

```
git add .  
git commit -m "Deploy"  
git push heroku master(main)
```

«Пушить» проект на хероку

```
heroku run python manage.py migrate
```

Запускає міграції – створює таблиці БД

```
heroku run python manage.py createsuperuser
```

Створює адміністратора/модератора ПП

В якості третього рівня доступу в якості «додатку» до хостингового продукту була використана БД Heroku Postgres (рис. 5.1.2)

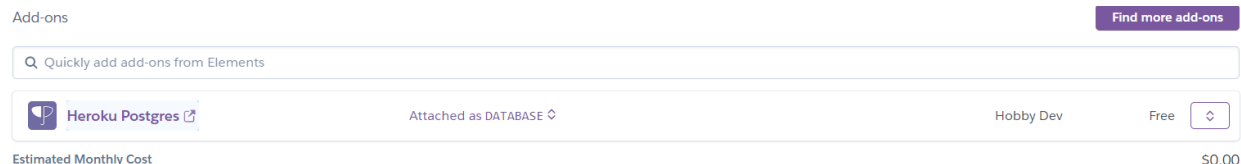


Рис. 5.1.2 – Додаток Heroku Postgres до застосунку

В результаті деплою було виявлено, що для зберігання файлів недостатньо лише БД, тож був під'єднаний ще 1 сервер до рівня доступу, який дозволяв зв'язати дані з БД з файлами – AWS Bucket, тож після цього діаграма розгортання ППІ стала такою, як показано на рисунку 5.1.3

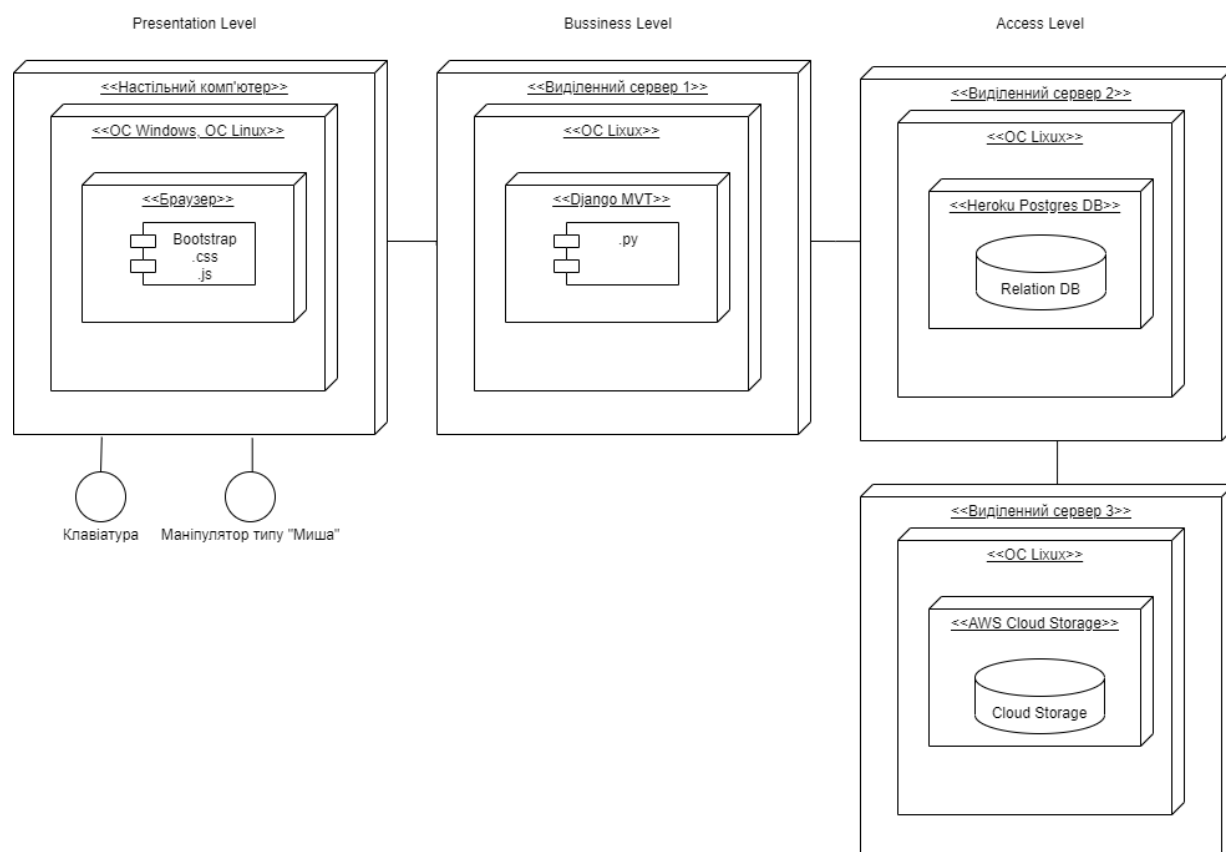


Рис. 5.1.3 – UML-діаграма розгортання ППІ

Розроблене програмне забезпечення підтримується усіма веб-браузерами, усіма версіями як на ОС Windows, Mac OS, так и на Linux ОС. Здійснювати дії на веб-сервісі та користуватися ним користувач може за допомогою маніпулятора «миша» та клавіатури. За допомогою маніпулятора «миша»

користувач може натиснути на кнопку/текст, а за допомогою клавіатури – вводити дані у різні поля та форми.

Система також має підтримуватися у всіх веб-браузерах мобільних пристроїв та усі дії будуть реалізовані користувачем за допомогою сенсора. Але, на жаль, адаптування мобільної версії ще не розроблено та вона виглядає так же само, як і на десктопній версії, що є незручним до користувача. Але у подальшому адаптація для мобільних пристроїв теж буде розроблена.

## 5.2 Інструкція з використання програмного продукту

5.2.1 Реєстрація користувача ПП надає можливість користувачу «гість» увести параметри реєстрації (прізвище, ім'я, електронна пошта, телефон, пароль), як показано на рисунках 5.2.1.1 - 5.2.1.2

Имя пользователя:

Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./+/-/\_.

Email:

Обязательно к вводу, введите действующую электронную почту

Пароль:

- Ваш пароль должен содержать как минимум 8 символов.
- Пароль не может состоять только из цифр.

Подтверждение пароля:

Для подтверждения введите, пожалуйста, пароль ещё раз.

[Уже есть аккаунт? Войти](#)

Рис. 5.2.1.1 – Вікно реєстрації

Имя пользователя:

Veseleil

Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @./+/-/\_.

Email:

vadimol081@gmail.com

Обязательно к вводу, введите действующую электронную почту

Пароль:

.....

- Ваш пароль должен содержать как минимум 8 символов.
- Пароль не может состоять только из цифр.

Подтверждение пароля:

.....

Для подтверждения введите, пожалуйста, пароль ещё раз.

Зарегистрироваться

Уже есть аккаунт? [Войти](#)

Рис. 5.2.1.2 – Приклад заповнення інформації

Після реєстрації користувач потрапляє на головну сторінку (рис. 5.2.1.3), Там він може передивлятись список усіх фільмів. Також користувач може шукати фільми за назвою та сортувати їх (рис. 5.2.1.4 -5.2.1.5)



Рис. 5.2.1.3

## ПОИСК ФИЛЬМА




Рис. 5.2.1.4

## Жанры

- ☐ Аниме
- ☐ Драма
- ☐ Вестерн
- ☐ Боевики
- ☐ Комедии
- ☐ Ужасы

Рис. 5.2.1.5

### 5.2.2 Авторизация користувача

Авторизация користувача ПП надає можливість користувачу «гість» увійти до свого облікового запису (рис 5.2.2.1)



Имя пользователя:

Пароль:

Войти

Ещё нет аккаунта?  
[Зарегистрироваться](#)

Рис. 5.2.2.1 – Вікно авторизації користувача

Що дає можливість користувачу ввести свої дані (рис. 5.2.2.2) та увійти до облікового запису, після чого його перенаправить на головну сторінку.

Имя пользователя:

Пароль:

Войти

Ещё нет аккаунта?  
[Зарегистрироваться](#)

Рис. 5.2.2.2 – Приклад введення даних

### 5.3 Результати валідації програмного продукту

Метою програмного продукту було підвищення рівня доступності до інформації про різноманітні фільми, серіали та інший медіа-контент на підставі створення веб-сайту для об'єднання необхідної інформації.

В даний період часу проект знаходиться на ранній стадії свого розвитку, але внаслідок буде розвинений більше. У розробленому ПП доступна інформація про увесь наявний медіа-контент, а також є алгоритми, що допомагають користувачам обирати наступний контент більш якісно.

Можна побачити, що метричний показник, що знаходився на позиції 0.4 перейшов до стану 1.0, однак, це не зовсім повно описує метрику вирішення проблеми через те, що вибірка контенту для метрики є не зовсім репрезентативною.

Однак, можна зі впевненістю говорити, що так як основною метою веб-додатку є саме надання користувачам інформації про актуальний контент, метричний показник доступності програми є близьким до 1.0.

## Висновки

В результаті створення програмного продукту була досягнута наступна мета його споживача: «Покращення рівня цінності знайденої інформації при пошуку фільму для перегляду та створення можливості отримання інформації на мові користувача.».

Доказом цього є наступні факти. Програмний продукт «Що подивитись?» виконує функції бази даних для фільмів, акторів і режисерів, однак містить алгоритмічні можливості, що дозволяють обирати фільми на основі особистих вподобань користувача.

«Що подивитись?» задовольняє такі потреби споживача:

1. Пошук контенту.
2. Швидке сортування необхідної інформації.
3. Можливість провести час з користю.
4. Інформаційну потребу.

В процесі створення програмного продукту виникли такі труднощі

- 1) організаційні труднощі роботи у команді;
- 2) брак часу;
- 3) відсутність досвіду у front-end розробці;
- 4) відсутність досвіду в розгортанні продуктів.

Через вищеописані непередбачені труднощі, а також через обмежений час на створення програмного продукту, залишилися нереалізованими такі прецеденти або їх окремі кроки роботи:

- Налаштування алгоритмічного пошуку для більшої його точності.
- Додавання оцінки виду «Кількість зірок»

Зазначені недоробки планується реалізувати в майбутніх курсових роботах з урахуванням тем дисциплін наступних семестрів.

					ІС КР 122 АІ181 ПЗ	
Зм.	Арк.	№ докум.	Підп.			67