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 № 2

Виконав:
студент 3-го курсу
групи АІ-181
Совяк Артем Ігорович
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., Совяк 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

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

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».

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 сформувати 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**
Представленність мовою споживача	0.4

1.2.2 Мета створення програмного продукту

1.2.2.1 Проблемний аналіз існуючих програмних продуктів

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?»

1.2.3.1 Гасло програмного продукту

Думай під час перегляду, а не під час пошуку.

1.2.3.2 Логотип програмного продукту

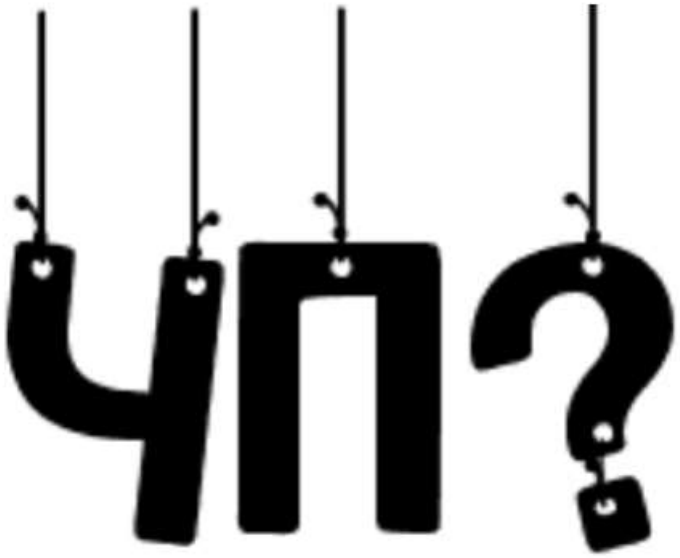


Рис. 1.2.1 – Логотип програмного продукту

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

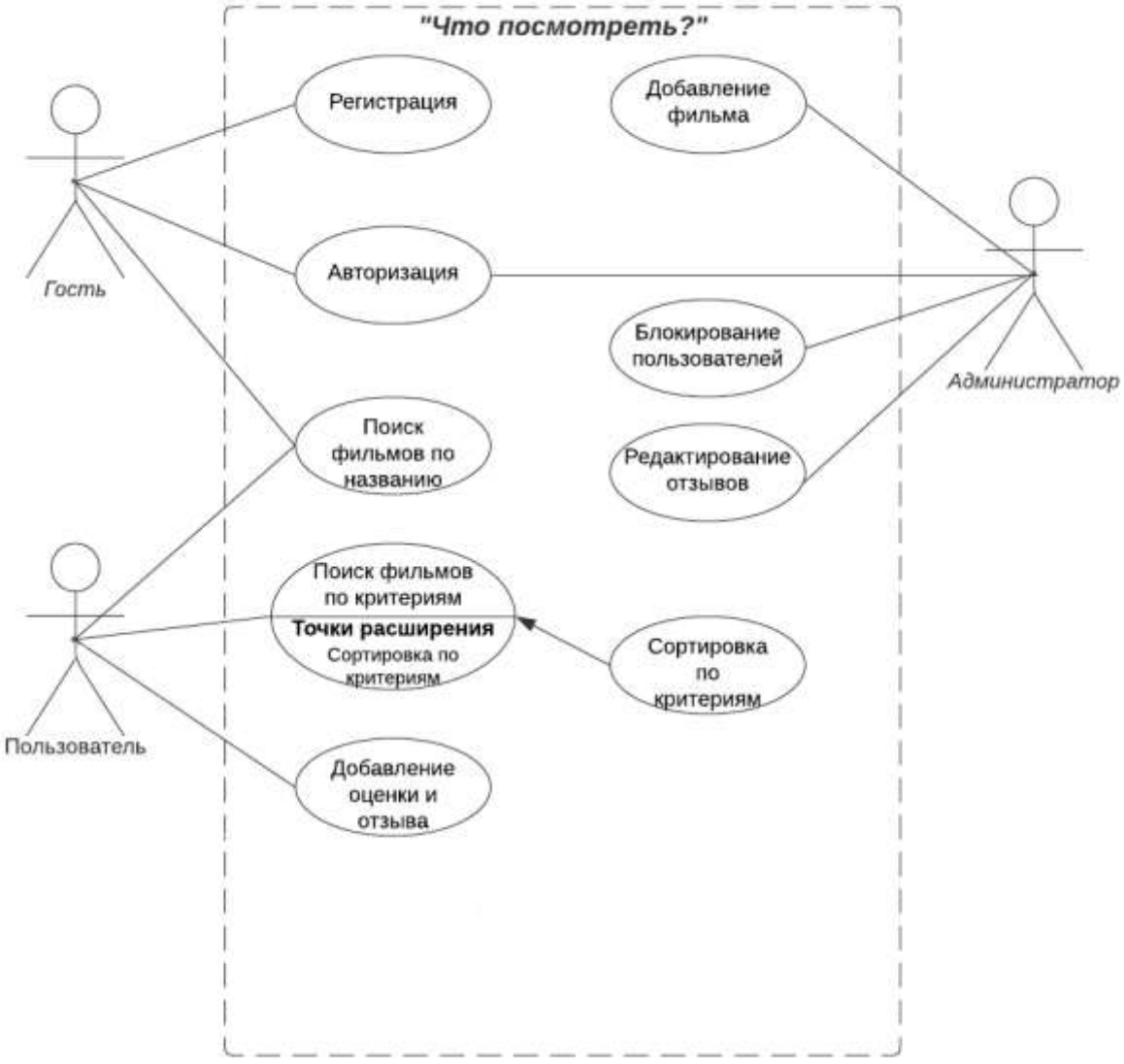


Рис. 1.3.2.1 – Діаграма прецедентів

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 ненаявністю 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
FR1	**Реєстрація користувача**

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
FR2	**Авторизація користувача**
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
FR3	**Пошук фільмів за назвою**
FR3.1	Введення користувачем назви фільму
FR3.2	Пошук фільму у БД
FR3.3	Повернення результату пошуку
FR4	**Пошук фільмів по критеріям**
FR4.1	Введення користувачем критеріїв фільму
FR4.2	Пошук фільмів у БД

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

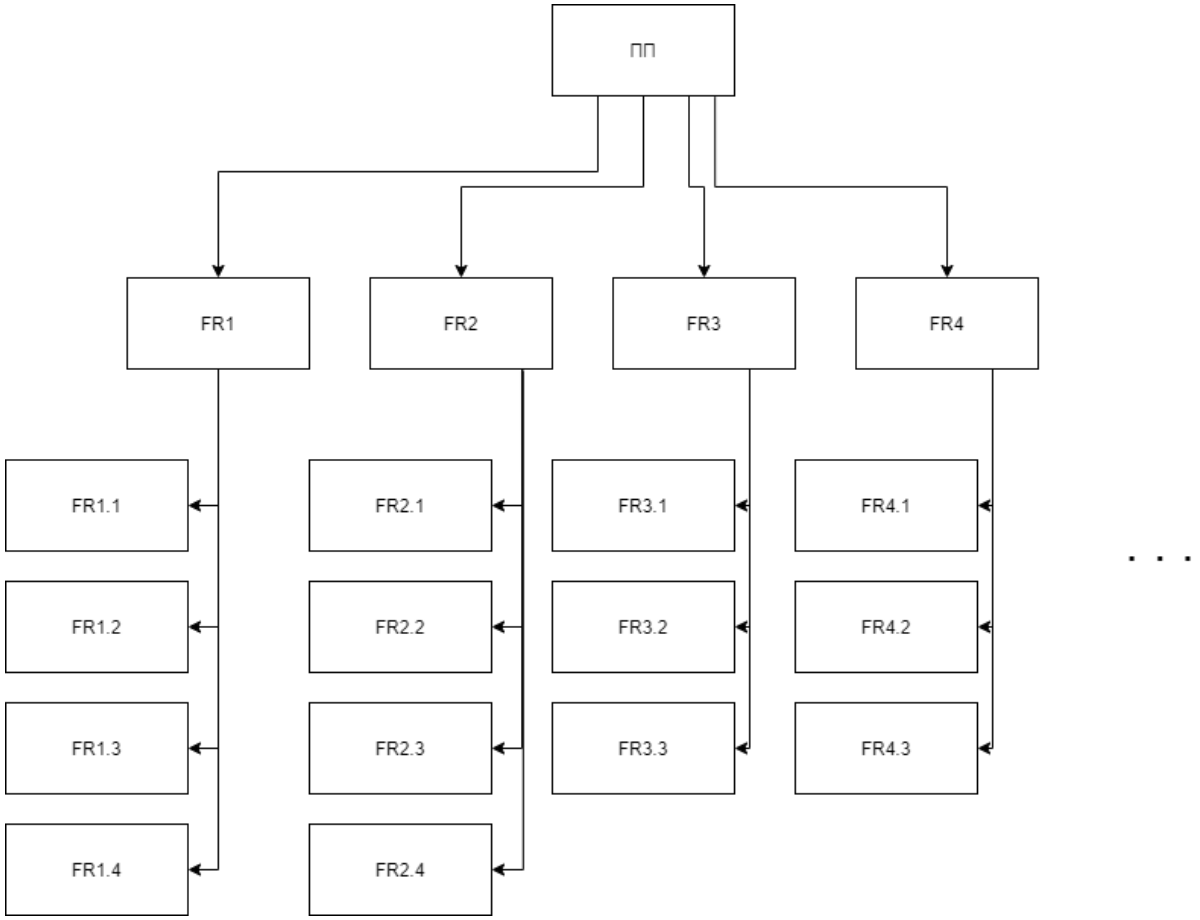


Рис. 1.4.1.1 - WBS-структура багаторівневої класифікації функціональних вимог

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-потоку**	**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 данних 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-потоку**	**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

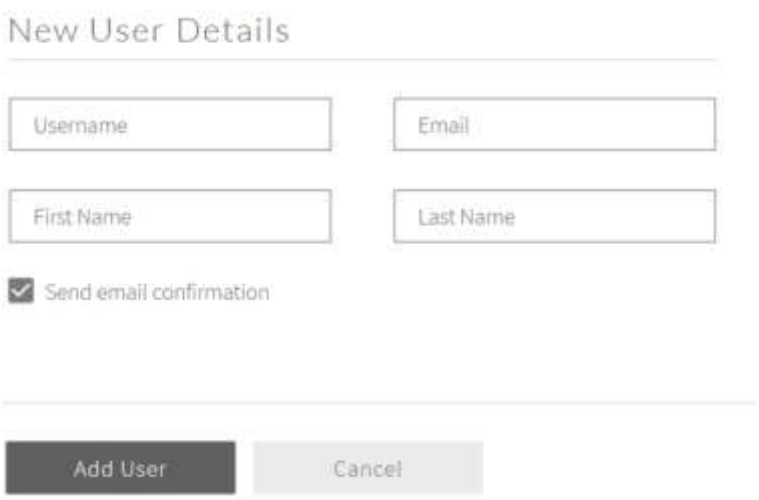


Рис. 1.5.1 – Mockup функції FR1

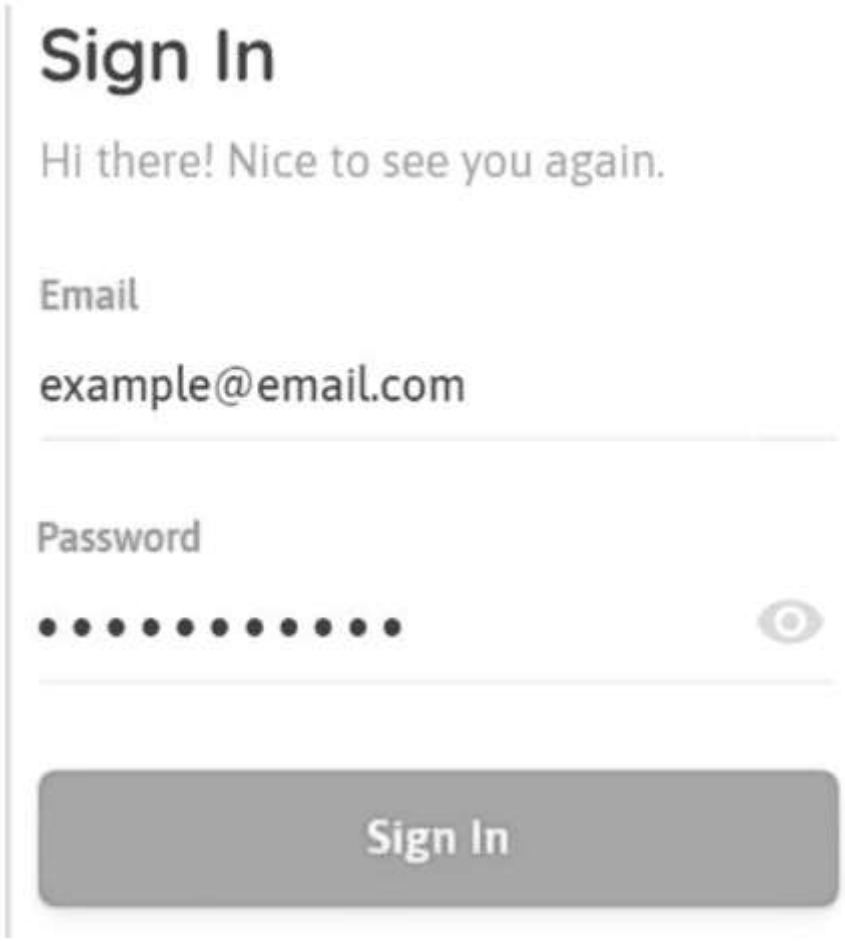


Рис. 1.5.2 – Mockup функції FR2

Рис. 1.5.3 – Mockup функції FR3

Рис. 1.5.4 – Mockup функції FR4

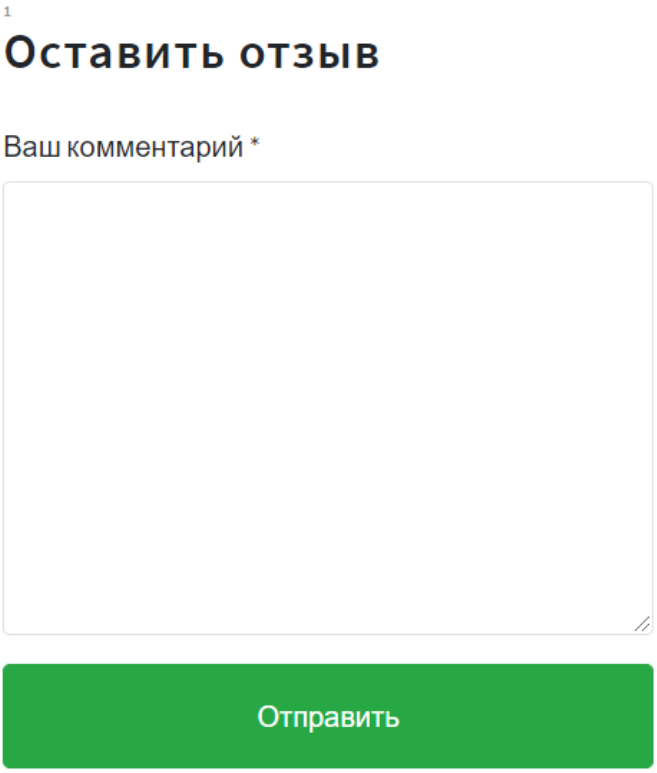


Рис. 1.5.5 – Mockup функції FR5

Рис. 1.5.6 – Mockup функції FR6

Рис. 1.5.7 – Mockup функції FR7

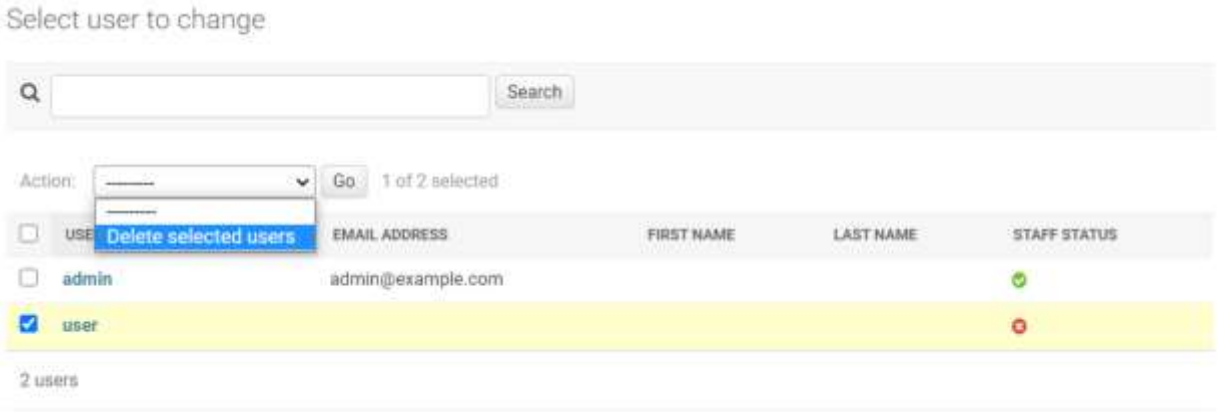


Рис. 1.5.8 – Mockup функції FR8

1.5.1.2 Опис інтерфейсу із зовнішніми пристроями

Ідентифікатор функції	**Зовнішній пристрій**

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 реалізованої у перших ітераціях, якщо це взагалі можливо;

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

− W (Want) – функція може бути реалізованої 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**
FR1	**-**	**20**	**M**
FR1.1	-	-	-
FR1.2	-	-	-
FR1.3	-	-	-
FR2	**FR1.1**	**20**	**M**
FR2.1	FR1.1	-	-
FR2.2	FR1.1	-	-
FR2.3	FR1.1	-	-
FR2.4	FR1.1	-	-
FR3	**FR2**	**15**	**S**
FR3.1	FR2	-	-
FR3.2	FR2	-	-
FR3.3	FR2	-	-

FR4	**FR2**	**15**	**S**
FR4.1	FR2	-	-
FR4.2	FR2	-	-
FR4.3	FR2	-	-
FR5	**FR2**	**0**	**W**
FR5.1	FR2	-	-
FR5.2	FR2	-	-
FR6	**FR2**	**20**	**M**
FR6.1	FR2	-	-
FR6.2	FR2	-	-
FR7	**FR2**	**5**	**C**
FR7.1	FR2	-	-
FR7.2	FR2	-	-
FR8	**FR2**	**5**	**C**
FR8.1	FR2	-	-
FR8.2	FR2	-	-

2.2 Концептуальний опис архітектури програмного продукту

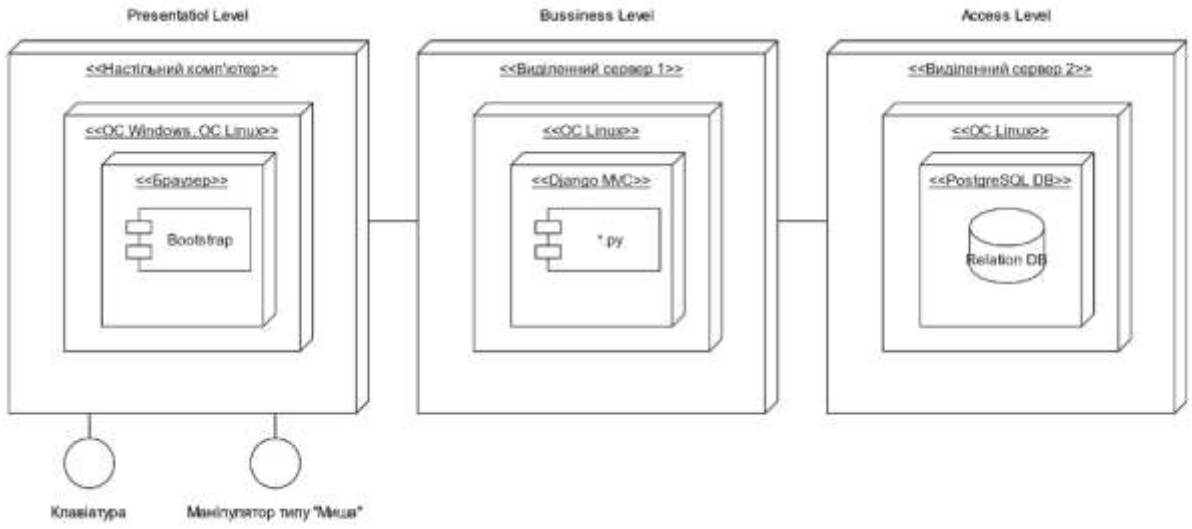


Рис. 2.2.1 – Концептуальний опис архітектури програмного продукту

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

Таблиця 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 * (STi * Вага_i)) = 0,6 + (0,01 * 27,5) = 0,87

3. Визначення рівня кваліфікації розробників

Таблиця 2.3.1.4 – Визначення рівня кваліфікації розробніків

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 * (SFi * Вагаі)) = 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).

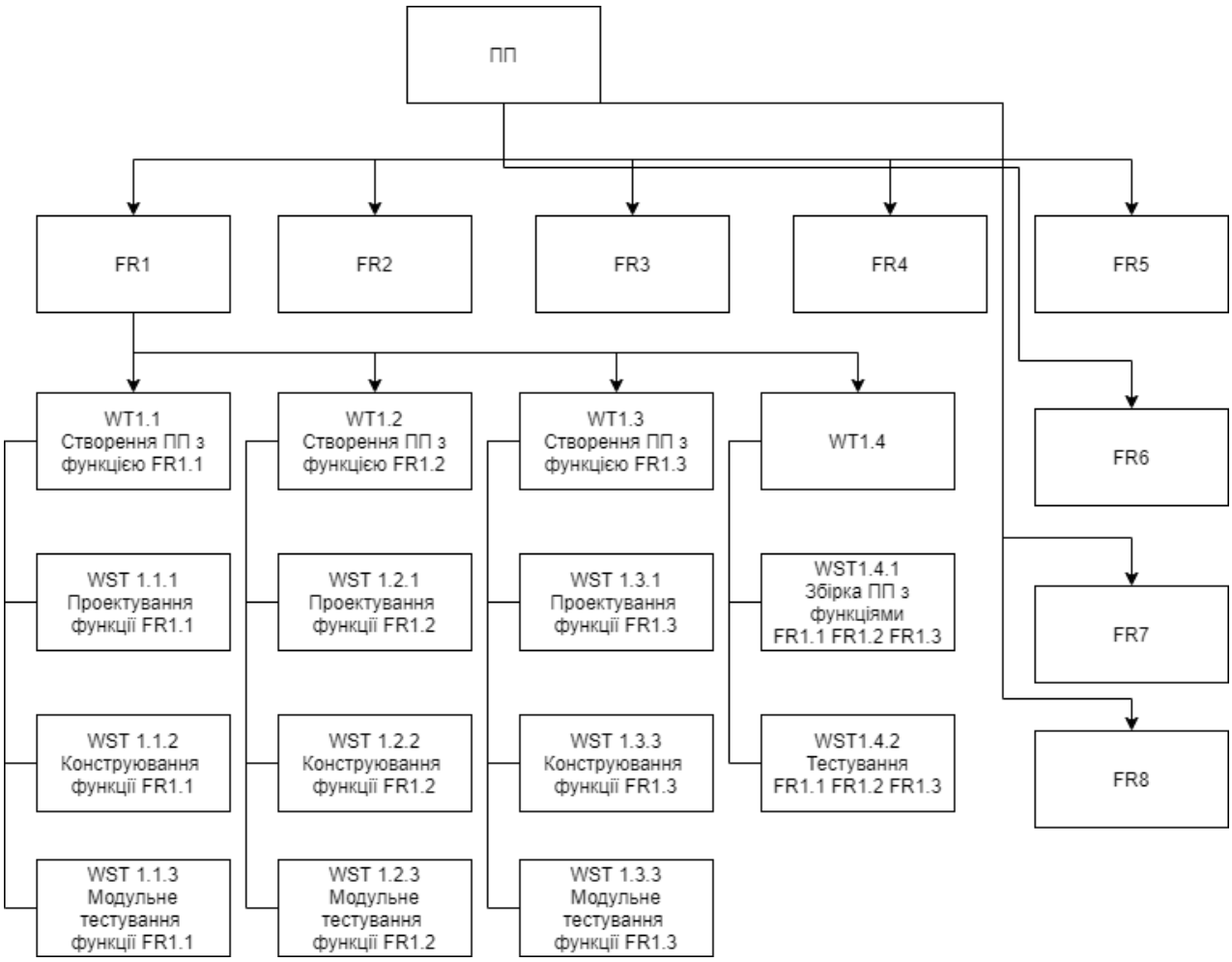


Рис. 2.3.2.1 – Дерево робіт

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	Пшеничнюк А.О.
WST7.1.2	17.10.2020	2	18.10.2020	Пшеничнюк А.О.
WST7.1.3	19.10.2020	2	20.10.2020	Пшеничнюк А.О.
WST8.1.1	15.10.2020	1	15.10.2020	Пшеничнюк А.О.
WST8.1.2	16.10.2020	1	16.10.2020	Пшеничнюк А.О.
WST8.1.3	17.10.2020	1	17.10.2020	Пшеничнюк А.О.

2.3.3.2 Діаграма Ганта

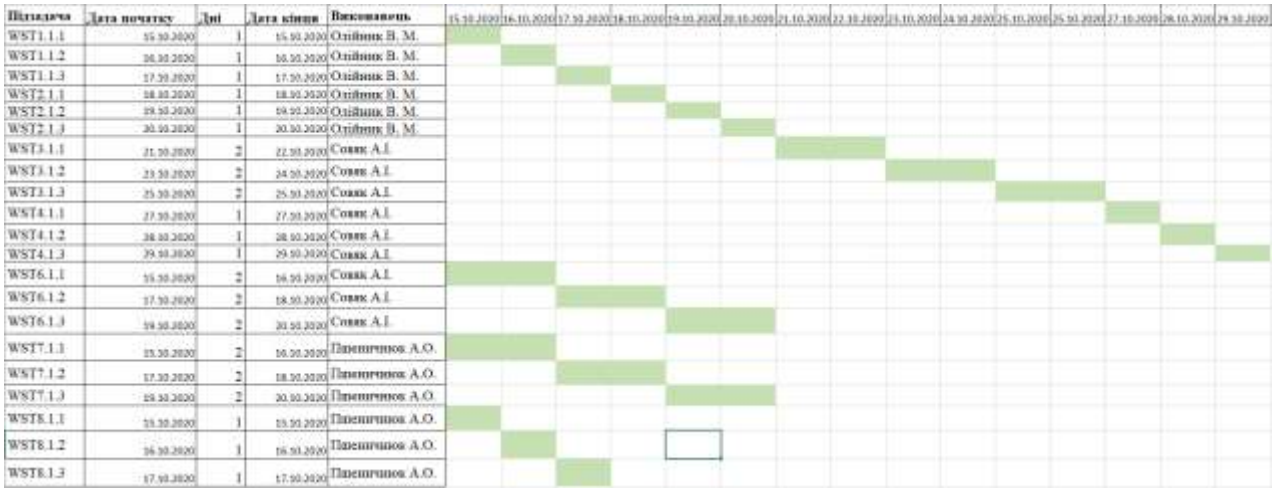


Рис. 2.3.3.2 – Діаграма Ганта

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).

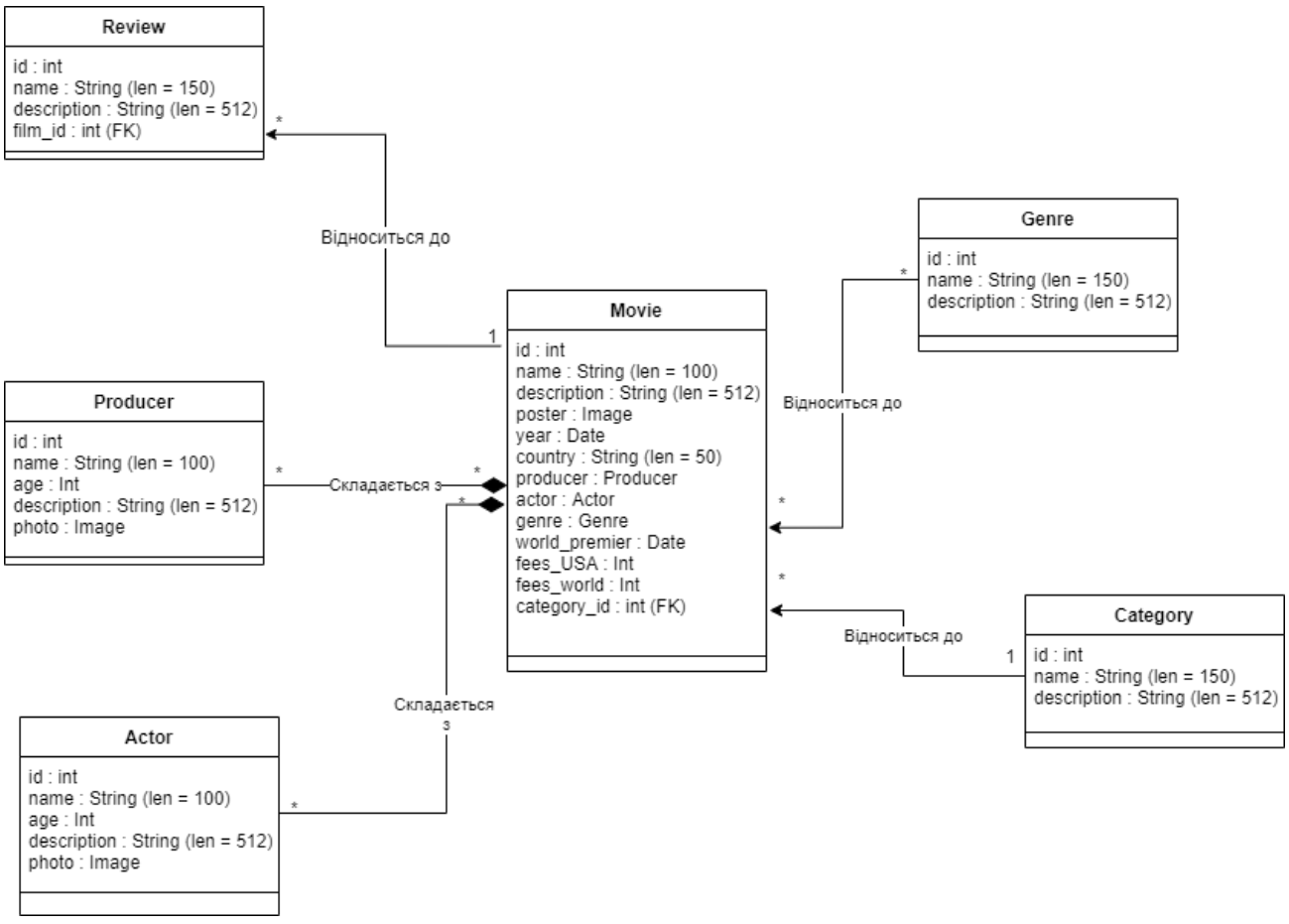


Рис. 3.1.1

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).

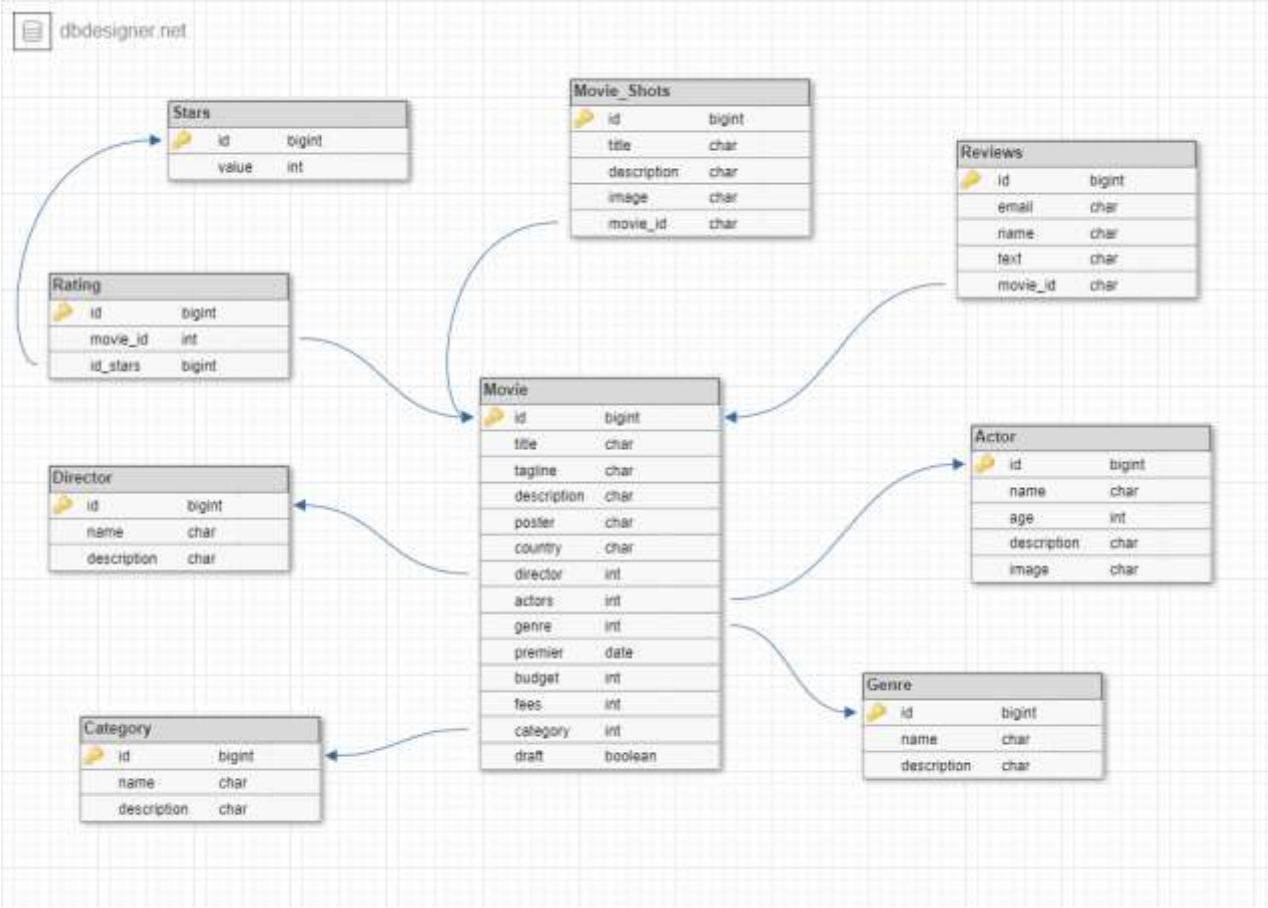


Рис. 3.1.2 – Схема БД

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

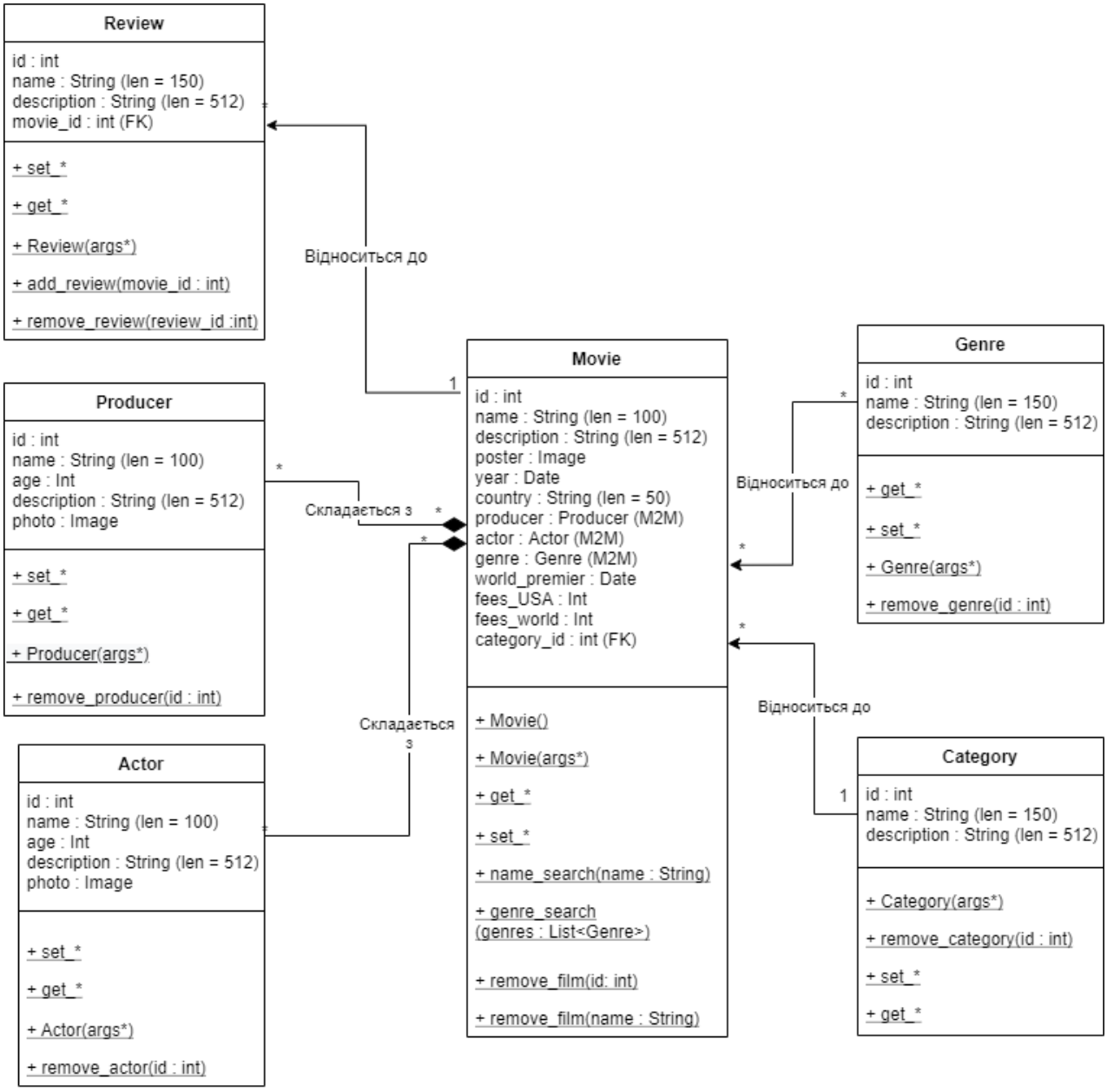


Рис. 3.2.1 – UML-діаграма класів

3.3 Проектування алгоритмів роботи методів програмних класів

За допомогою UML були описані такі методи: AddMovie, get_movies, get_movie_by_name, get_movies_by_genre

UML-код алгоритму додавання фільму

```
@startuml
title Movie.addMovie(name, desctiption, poster, year, country,
producer, actor, genre, premier, fees, category)
start
repeat
    :Вывод экранной формы для добавления фильма;
        note right
            Мокап экранной формы FR7.1 представлен
            в разделе "Описание OUTPUT-интерфейса пользователя"
        end note
    :Ввод администратором данных о фильме;
    :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 movie
                VALUES(name, desctiption, poster, year, country,
producer, actor, genre, premier, fees, category)
            end note
    :Информирование об успешном завершении операции;
stop
@enduml
```

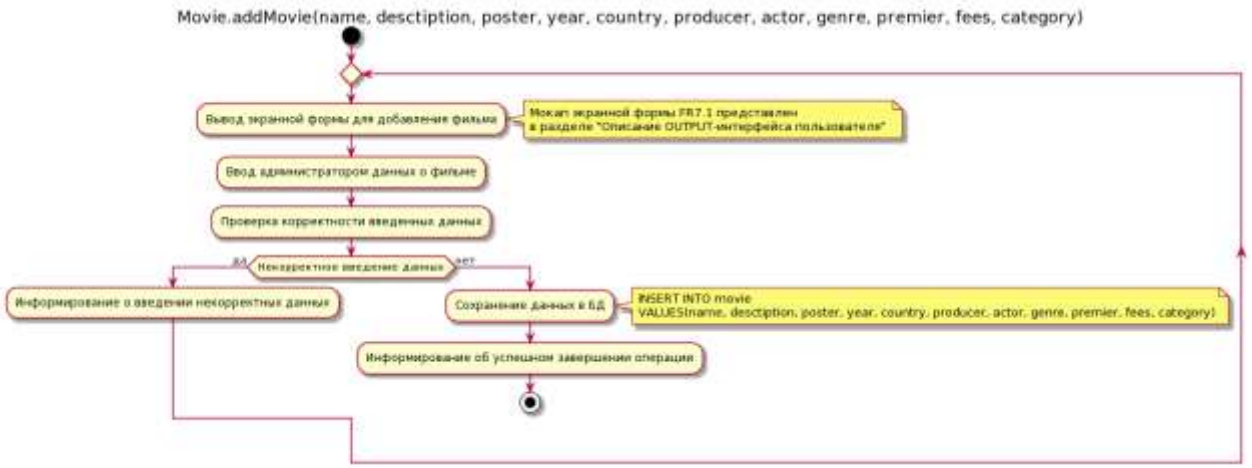


Рис. 3.3.1 – Діаграма методу Movie.addMovie()

UML-код перегляду списку усіх фільмів

```
@startuml
title Movie.get_movies()
start
:Запрос от пользователя на список всех фильмов;
:Получение списка фильмов;
    note left
        SELECT * FROM movies
    end note
if (Список фильмов пуст) then (нет)
    :Вывод списка фильмов;
else (да)
    :Уведомление пользователя об отсутствии фильмов;
endif
stop
@enduml
```

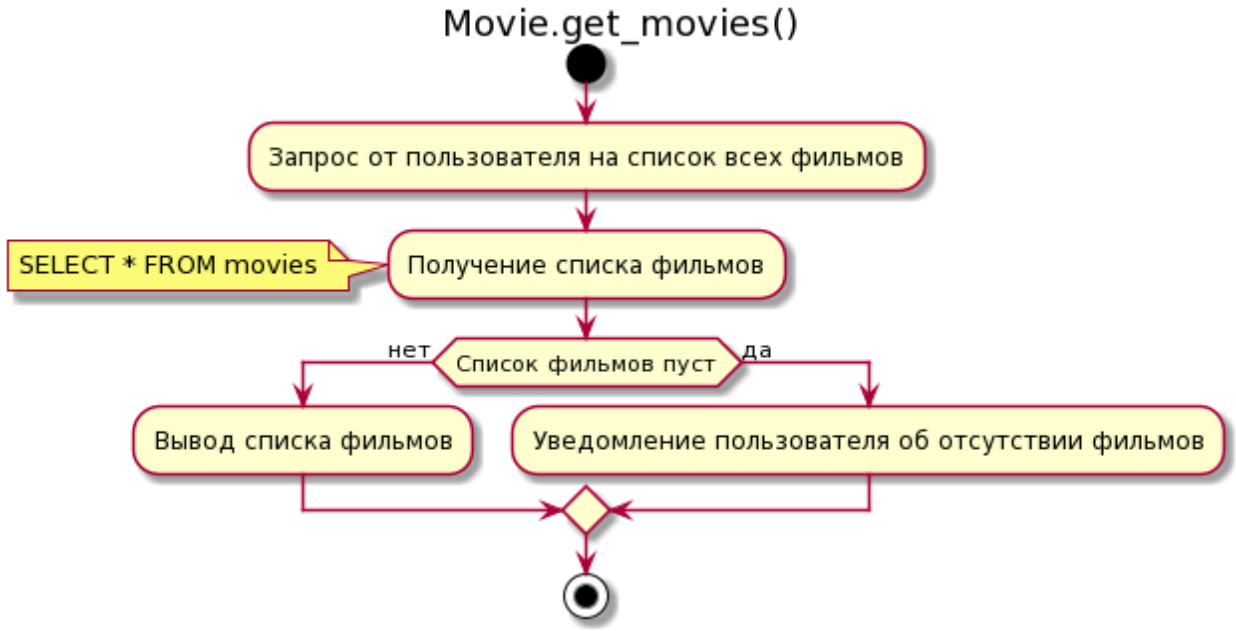


Рис. 3.3.2– Діаграма методу Movie.get_movies()

UML-код пошуку фільмів за жанрами

```
@startuml
title Movie.get_movies_by_genre(genres : List)
start
:Вывод экранной формы жанров фильмов;
    note right
        Мокап экранной формы FR4.1 представлен
        в разделе "Описание OUTPUT-интерфейса пользователя"
    end note
:Ввод пользователем желаемых жанров для поиска;
:Получение списка фильмов по жанрам;
    note left
```

```
        SELECT * FROM movies WHERE genres IN (*Выбранные пользователем
жанры*)
    end note
if (Список фильмов пуст) then (нет)
    :Вывод списка фильмов;
else (да)
    :Уведомление пользователя об отсутствии фильмов;
endif
stop
@enduml
```

Рис. 3.3.3– Діаграма методу Movie.get_movies_by_genre()

UML-код пошуку фільму за назвою

```
@startuml
title Movie.get_movie_by_name(movie_name)
start
:Вывод экранной формы поиска фильма по названию;
    note right
        Мокап экранной формы FR3.1 представлен
        в разделе "Описание OUTPUT-интерфейса пользователя"
    end note
:Ввод пользователем названия фильма для поиска;
:Получение данных об искомом фильме;
    note left
        SELECT * FROM movies WHERE name =  "movie_name"
    end note
if (Фильм есть) then (да)
    :Вывод искомого фильма;
else (нет)
    :Уведомление пользователя об отсутствии данного фильма;
    :Вывод окна создания запроса на добавление фильма;
endif
stop
@enduml
```

Рис. 3.3.4 – Діаграма методу Movie.get_movies_by_name()

3.4 Проектування тестових наборів методів програмних класів

Назва функції	№ тесту	Опис значень вхідних даних	Опис очікуваних значень результату
`Movie.addMovie(name, desctiption, poster, year, country, producer, actor, genre, premier, fees, category)`	1	Коректний список вхідних аргументів	Збереження фільму до БД
	2	name:<String>, len > 100	Помилка вхідних даних
	3	description:<String>, len > 512	Помилка вхідних даних

	4	year, форматів, окрім <dd-mm-yyyy>	Помилка вхідних даних
	5	country, len > 50	Помилка вхідних даних
	6	premier, форматів, окрім <yyyy>	Помилка вхідних даних
	7	fees, fees < 0	Помилка вхідних даних
`Movie.get_movies()`	1	-, Якщо фільми є в БД	Повертає список усіх знайдених фільмів
	2	-, Якщо фільмів немає в БД	Повідомлення про відсутність фільмів
`Movie.get_movie_by_name(name)`	1	name=<String>, len < 100 назва фільму, який є в БД	Повернення об’єкту Movie з назвою name
	2	name=<String>, назва фільму, якого нема в БД	Повернення повідомленн

			я про відсутність фільму
	3	name=<String>, len > 100	Помилка вхідних даних
`Movie.get_movies_by_genre(genres : List)`	1	genres:<List>	Повернення списку усіх фільмів заданих жанрів
	2	genres:<будь-який тип, окрім List>	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.

```python
class Movie(models.Model):
    """Фильм"""
    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 формує текстове поле
- Атрибут models.ImageField формує 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 формує 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.Boolean сформує поле логічного типу даних, яке може приймати лише True або False
- Атрибут models.Slug - формує 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), ліцензія – студентська.

Включає зручні інструменти розробки. Використовувалися фреймворки Django, Bootstrap, psycopg2 – 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

FilterMovies

```
class FilterMoviesView(GenreYear, ListView):
    paginate_by = 6

    def get_queryset(self):
        queryset = Movie.objects.filter(
            Q(year__in=self.request.GET.getlist("year")) |
            Q(genres__in=self.request.GET.getlist("genre"))
        ).distinct()
        return queryset

    def get_context_data(self, *args, **kwargs):
        context = super().get_context_data(*args,**kwargs)
        context["year"] = ''.join([f"year={x}&" for x in self.request.GET.getlist("year")])
        context["genre"] = ''.join([f"genre={x}&" for x in self.request.GET.getlist("genre")])
        return context
```

Рис. 4.2.4.1 – Метод FilterMovies

Розглянемо реалізацію метода get_movie_by_genre(movie_genre). Цей метод був об’єднаним з методом get_movie_by_year(movie_year) у більш загальний метод FilterMoviesView, що має можливість сортувати фільми водночас як за роком випуску, так і за жанрами для надання більш повної можливості вибору.

Завдяки елементу Q із queryset, метод має доступ до логічної операції «або», що дозволяє, наприклад, шукати водночас фільми різноманітних несумісних жанрів, або різних років виходу.

Також цей метод займається тим, що перебудовує url-адресу таким чином, щоб вона явно демонструвала те, що саме шукає користувач у даний момент часу.

У методу також наявний елемент пагінації – кожні 6 фільмів буде створена наступна сторінка пошуку для того, щоб не забруднювати екран і не робити потенційно нескінчених списків.

Movie.get_movies

```
class Search(GenreYear, ListView):
    """Поиск фильмов"""
    paginate_by = 3

    def get_queryset(self):
        return Movie.objects.filter(title__icontains=self.request.GET.get("q"))

    def get_context_data(self, *args, **kwargs):
        context = super().get_context_data(*args, **kwargs)
        context["q"] = f'q={self.request.GET.get("q")}&'
        return context
```

Рис. 4.2.4.2 – Метод Search

Розглянемо реалізацію методу Movie.get_movies(). У програмному коді він представлений у якості метода Search, якому передається у якості вхідних аргументів список усіх фільмів для того, щоб шукати по назві серед них.

Після запиту до бази даних, генерується url-строка, за якою і буде знаходитися відповідь з бази на предмет того, чи є фільми зі схожою назвою у ній.

У методу також наявний елемент пагінації – кожні 3 фільми буде створена наступна сторінка пошуку для того, щоб не забруднювати екран і не робити потенційно нескінчених списків.

Movie.addMovie

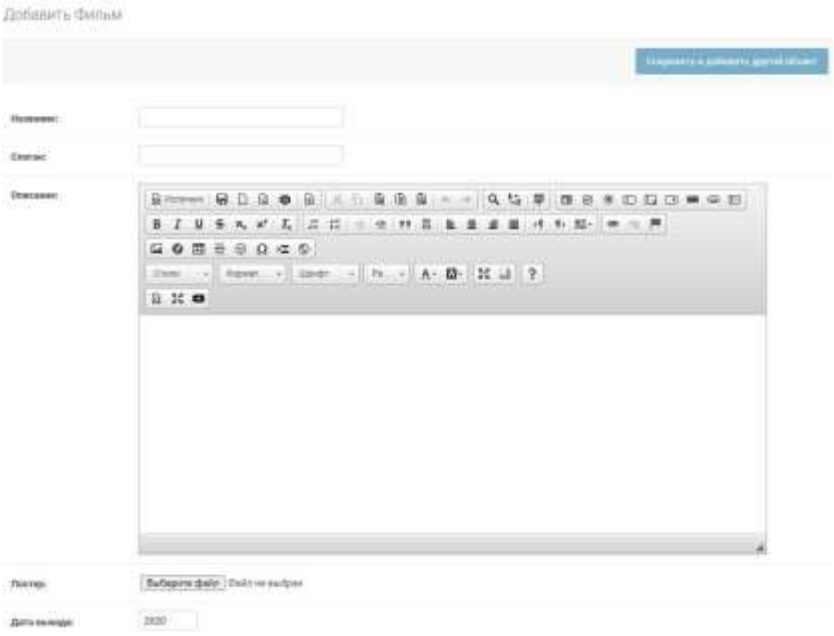


Рис. 4.2.4.3 – Панель додавання фільмів

Завдяки особливостям Django MVC, нові фільми додаються не через введення коду до командних строк, а через адміністративну панель, що має зручні та інтуітивні поля для налаштування контенту фільму.

Після створення фільму через панель, він з’являється:

- У базі даних;
- На головній сторінці;
- У списку останніх фільмів у лівій частині екрану.

Для більш зручного налаштування контенту фільмів була додана бібліотека CKEditor, що надає можливість редагувати зміст фільму, додавати посилання на трейлери та інше.

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

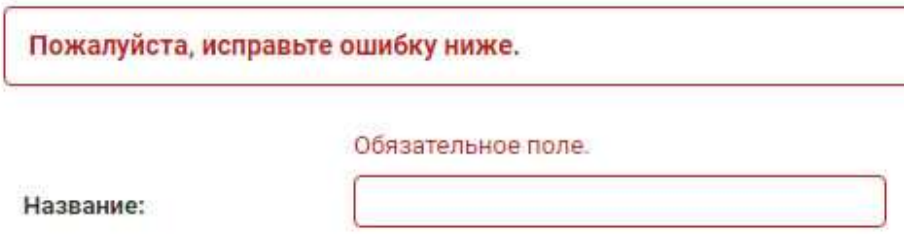
4.3 Модульне тестування програмних класів

4.3.1 Movie.addMovie(`name, desctiption, poster, year, country, producer, actor, genre, premier, fees, category`)

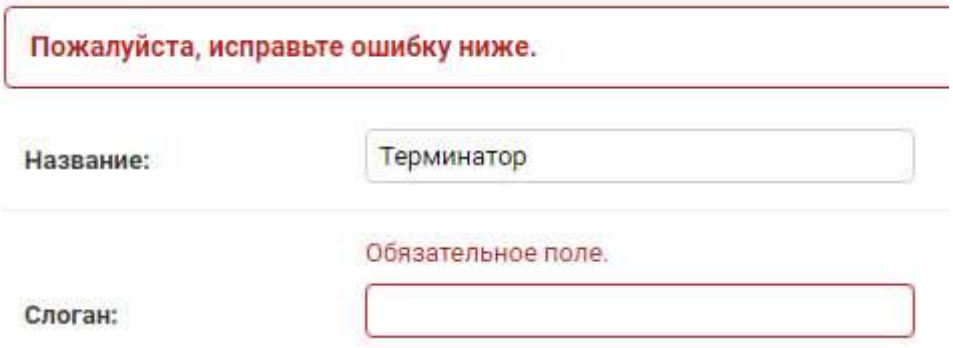
Movie.addMovie являє собою метод, що перевіряє введення користувачем даних та реєструє новий фільм. Метод реалізований з урахуванням наявного інтерфейсу адміністративної панелі, і викликається безпосередньо у ній при натисканні на кнопку «Сохранить» (далі на рис.)

При додаванні фільму до бази даних, маємо пам’ятати, що у фільму є як обов’язкові поля(назва, списки режисерів та акторів, тощо), так і необов’язкові(список коментарів, кадри з фільму, тощо). При тестуванні особливу роль грає саме випробування обов’язкових полей.

Тест 1 – Адміністратор не вводить назву фільму:

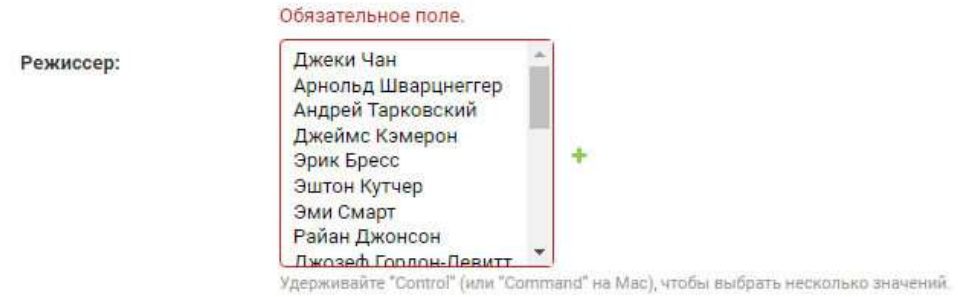


Тест 2 – Адміністратор не вводить слоган фільму:

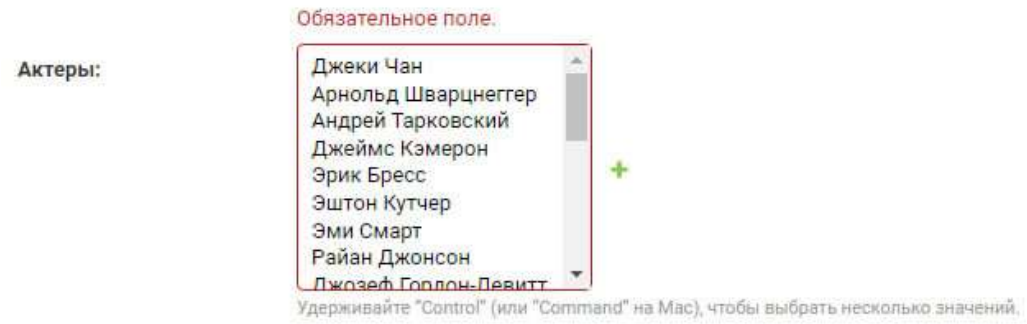


Тест 3 – Адміністратор не вводить рік виходу фільму:

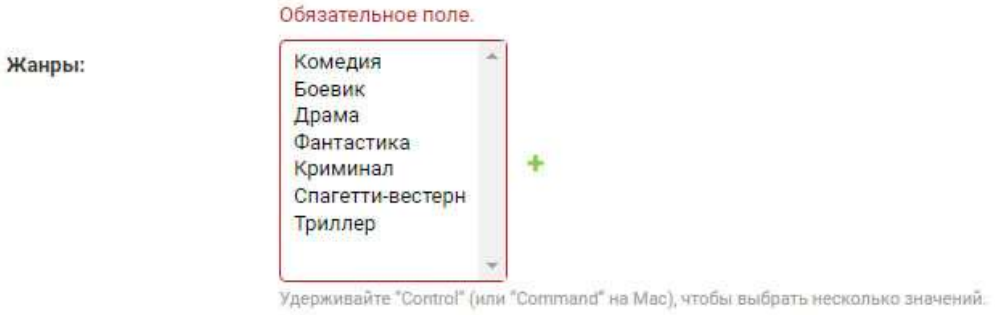
Тест 4 – Адміністратор не додає фільму список режисерів:



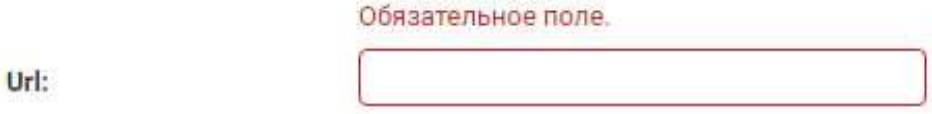
Тест 5 – Адміністратор не додає фільму список акторів:



Тест 6 – Адміністратор не додає фільму список жанрів:



Тест 7 – Адміністратор не вводить URL, за яким фільм і будуть шукати у базі даних:



У випадку успішного проходження тестів, фільм буде успішно створено, на сторінку з ним можна буде перейти як через головну сторінку сайту, так і через пошук.

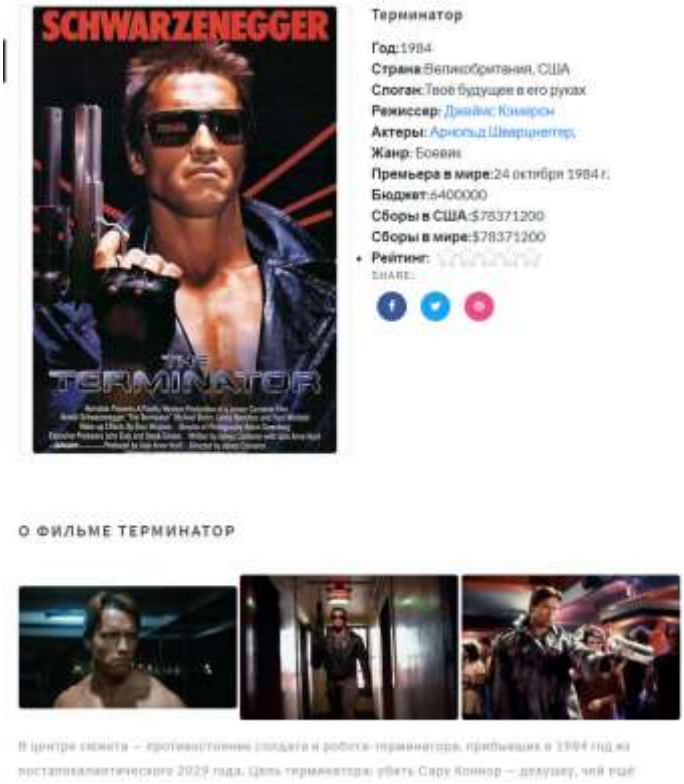


Рис. 4.3.1 – Приклад сторінки створеного фільму

4.3.2 – Movie.get_movies()

Даний метод не є необхідним для тестування – у випадку відсутності фільмів метод поверне повідомлення про відсутність фільму, однак така відповідь методу не є ошибкою. Метод визивається автоматично кожен раз, коли користувач заходить на головну сторінку сайту.

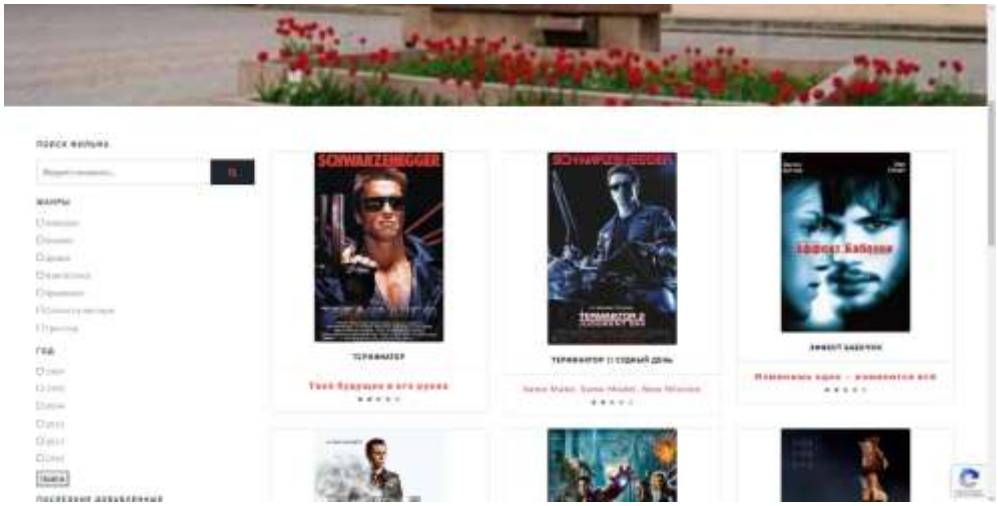


Рис. 4.3.2.1 – Приклад роботи методу на вже наявній базі даних

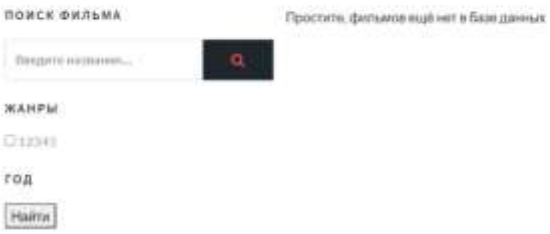


Рис. 4.3.2.2 – Приклад роботи методу на пустій базі даних

4.3.3 – Movie.get_movie_by_name(name)

Метод Movie.get_movie_by_name є методом, що взаємодіє безпосередньо з базами даних, і виводить окремою url-адресою список усіх фільмів, що підпадають під дану назву. Так як метод може у теорії і не знайти фільм по назві через відсутність такого фільму у базі даних, помилку цей метод не може викликати ніяк.

Метод має наступний код:

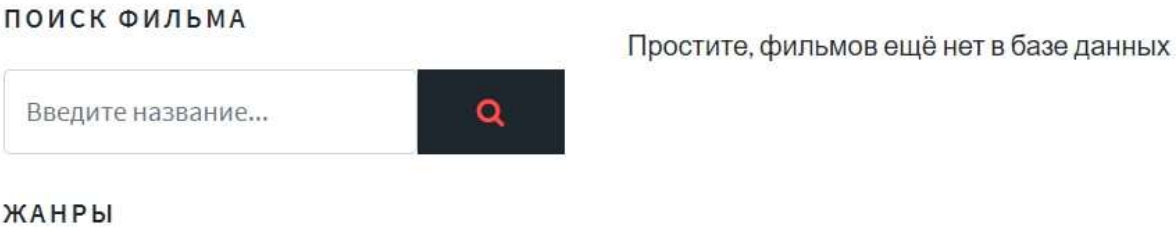
```
class Search(GenreYear, ListView):
    """Поиск фильмов"""
    paginate_by = 3

    def get_queryset(self):
        return Movie.objects.filter(title__icontains=self.request.GET.get("q"))

    def get_context_data(self, *args, **kwargs):
        context = super().get_context_data(*args, **kwargs)
        context["q"] = f'q={self.request.GET.get("q")}&'
        return context
```

Тест 1 – Пошук фільму, що є у базі даних.

Тест 2 – Пошук фільму, котрого нема у базі даних



4.3.4 - Movie.get_movie_by_genres(genres : <List>)

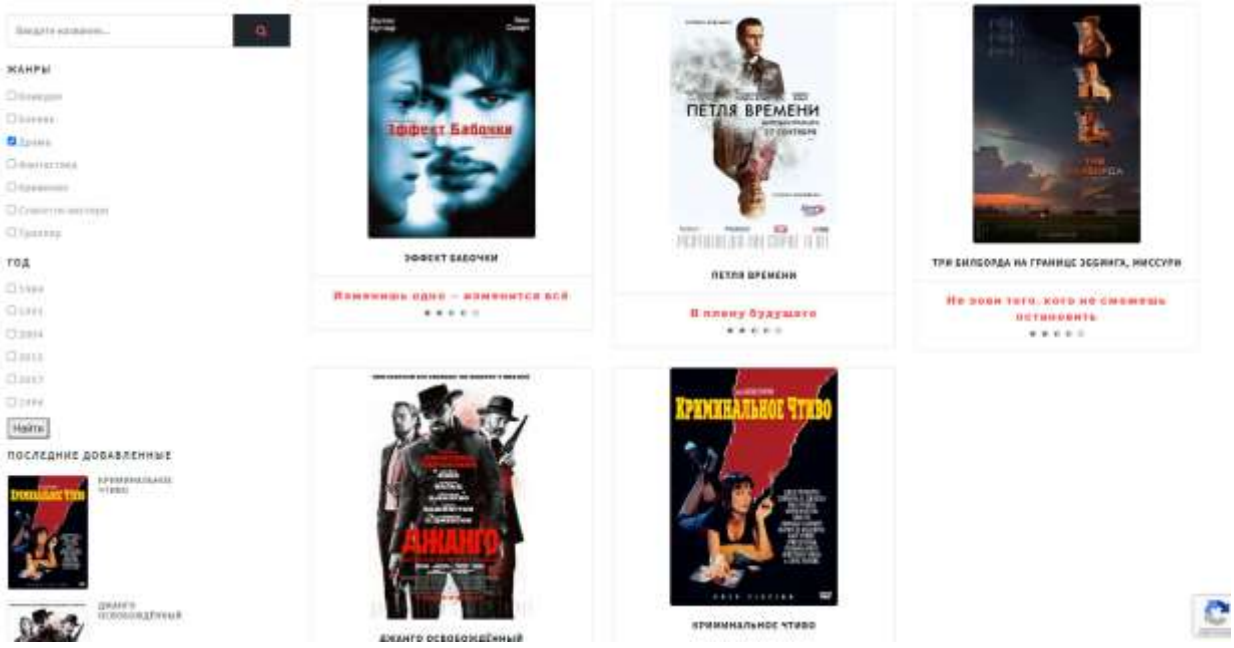
Даний метод надає користувачу можливість шукати фільми за жанрами, при чому метод працює як логічна операція «або», тобто якщо користувач забажає шукати фільми відразу декількох жанрів, метод не буде шукати лише ті фільми, де наявні всі жанри списку.

Метод має наступний код:

```
class JsonFilterMoviesView(ListView):
    """Фильтр фильмов в json"""
    def get_queryset(self):
        queryset = Movie.objects.filter(
            Q(year__in=self.request.GET.getlist("year")) |
            Q(genres__in=self.request.GET.getlist("genre"))
        ).distinct().values("title", "tagline", "url", "poster")
        return queryset
```

Так як додавання жанрів відбувається безпосередньо при створенні фільмів, ситуація, в якій користувач буде шукати фільм за неіснуючим жанром, ніколи не виникне.

Тест 1 – Пошук фільму за жанром:



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 забеспечення 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 вимикає 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 :

```
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

cycler==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

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

psycopg2==2.8.6

pycparser==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

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 ПП стала такою, як показано на рисунку 5.1.3

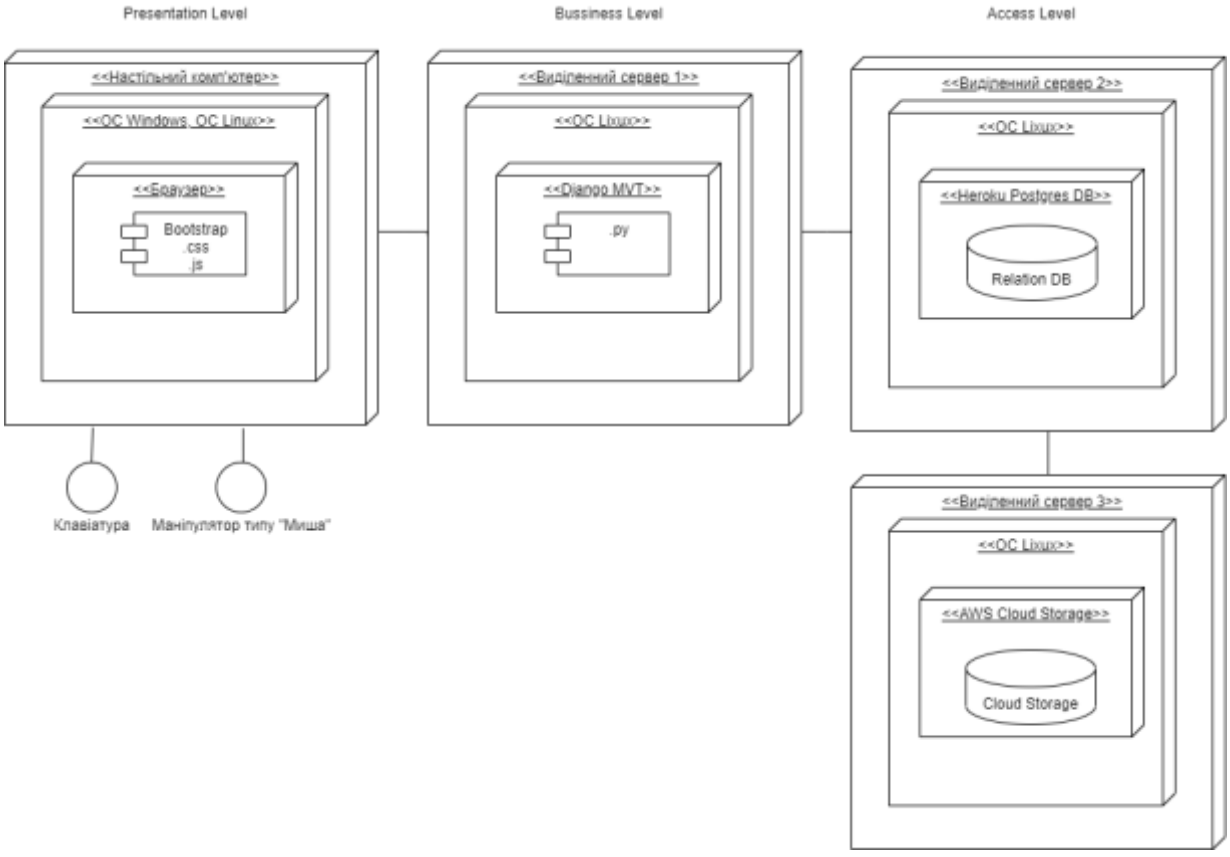


Рис. 5.1.3 – UML-діаграма розгортання ПП

Розроблене програмне забезпечення підтримується усіма веб-браузерами, усіма версіями як на ОС Windows, Mac ОС, 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 нажати на кнопку/текст, а за допомогою клавіатури – вводити дані у різні поля та форми.

Система також має підтрімуватися 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 декстопній версії, що є незручним до користучача. Але у подальшому адаптація для мобільних пристроїв теж буде розроблена.

5.2 Інструкція з використання програмного продукту

5.2.1 Пошук фільму надає користувачеві можливість шукати фільми по назві, по жанрам та по року виходу.

Після того, як користувач заходить на сайт, йому доступна панель налаштування, яка знаходиться по лівій стороні екрану:

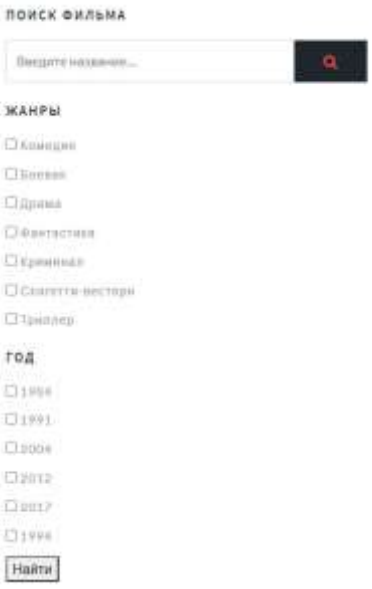


Рис. 5.2.1.1 – Вид панелі налаштування пошуку

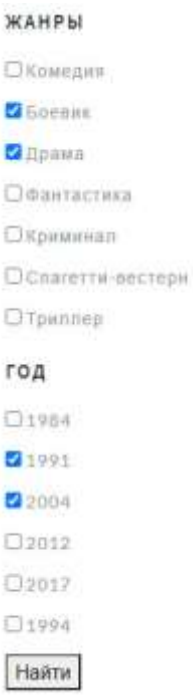


Рис. 5.2.1.2 – Приклад введення даних

У полі «Пошук фільму» можна шукати фільми за назвою, назву можна вводити як на російській та українській, так і на англійській та інших мовах.

У checkbox-полях «Жанри» та «Год» можна налаштовувати те, які жанри та годи випуску фільму користувач шукатиме при відправці запиту. Як роки, так і жанри реалізують функцію «або», що надає можливість більш гнучко налаштовувати результати пошуку та більш суттєво покривати області баз даних, до яких буде виконано запит.

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.