Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Інститут деревооброблювальних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри, циклової комісії)

КУРСОВИЙ ПРОЕКТ

з дисципліни «Методи та засоби ООАП»

на тему: «Розробка функціоналу реєстрації та класу домашній кінотеатр на основі шаблонів проектування Template method, Asynchronous processing та Facade»

Студента(ки) 4 курсу КН–41 групи
Напрям підготовки 6.050101
Дель Ріо Альбречет К.М.
(прізвище та ініціали)

Керівник Завідувач кафедри інформаційних технологій, проф, д.т.н.Соколовський Я.І
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала
Кількість балів: Оцінка: ECTS

Члени комісії
(підпис) Соколовський Я.І
(вчене звання, науковий ступінь, прізвище та ініціали).
(підпис) Мокрицька О.В.
(вчене звання, науковий ступінь, прізвище та ініціали)
(підпис) Яркун В.І.
(вчене звання, науковий ступінь, прізвище та ініціали)
(підпис) Нечепуренко А.В.
(вчене звання, науковий ступінь, прізвище та ініціали)

м.Львів – 2019 рік

ЗМІСТ

РЕФЕРАТ

Метою даної роботи є реалізація функціоналу реєстрації та класу домашній кінотеатр на основі шаблонів проектування Template method, Asynchronous Processing та Facade.

Програмна реалізація додатку користувача здійснена мовою Swift в середовищі XCode.

Цей курсовий проект є актуальним для набуття фахової підготовки зі спеціальності «Методи та засоби ООАП» за напрямком «Комп'ютерні науки».

Курсовий проект складається з завдання, 44 сторінок, 4 розділів, вступу, висновку, діаграм класів, містить 13 рисунків, 1 додаток, 4 джерела. У ньому розроблено програму, яка реалізує функціонал реєстрації користувача та клас домашній кінотеатр на основі запропонованих шаблонів проектування.

ABSTRACT

The aim of this project is designing registration functionality and home cinema class base on the Template method, Asynchronous Processing та Facade design patterns.

Software implementation of the application was made using the Swift programming language in the environment XCode.

This course project is important for acquiring professional training in the specialty "OOAD" in the direction of "Computer Science".

Course project consists of task, 44 pages, 4 sections, 1 addition, class diagrams and 13 pictures. There is a developed program that implements registration functionality and home cinema class based on proposed design patterns.

ВСТУП

Актуальність даної курсової роботи пояснюється тим, що саме свідоме розуміння шаблонів проектування відрізняє справжнього спеціаліста в сфері IT від аматора. Ви можете забити цвях молотком, а можете й дрилем, якщо дуже сильно постараєтесь. Але професіонал знає, що головне першочергове завдання дриля зовсім не в цьому. З чого випливає, що патерни надають нам безліч переваг, деякі з них перераховано нижче:

1. Перевірені рішення. Ви витрачаєте менше часу, використовуючи готові рішення, замість повторного винаходу велосипеда. До деяких рішень ви могли б дійти й самотужки, але багато які з них стануть для вас відкриттям.
2. Стандартизація коду. Ви робите менше прорахунків при проектуванні, використовуючи типові уніфіковані рішення, оскільки всі приховані в них проблеми вже давно знайдено.
3. Загальний словник термінів. Ви вимовляєте назву патерна, замість того, щоб годину пояснювати іншим спеціалістам, яку структуру програми ви плануєте реалізувати ти які класи для цього потрібні.
4. Перевикористання коду. Вдало написаний клас з високим рівнем абстракції може бути перевикористане не лише в межах однієї програми, а й застосований як готове рішення в майбутніх проектах.

Метою роботи являється закріплення набутих знать з дисципліни ООАП та практичне використання шаблонів проектування у реальних ситуаціях.

Результатами роботи являються повноцінно функціонуюча програма з функціоналом реєстрації та домашнього кінотеатру, які реалізовані ОО мовою.

РОЗДІЛ 1. ТЕХНІЧНЕ ЗАВДАННЯ

Потрібно написати універсальний клас для реєстрації входу користувачів в додаток. Крім можливості багаторазового використання і зручності застосування, до класу ставляться такі вимоги:

1. Надати можливість для користувача ввести свої ім'я та пароль;
2. Аутентифікувати ID користувача і пароль, результатом чого має бути об'єкт. Якщо при аутентифікації створюється деяка інформація, яка в подальшому знадобиться як підтвердження аутентифікації, то одержуваний в ході аутентифікації об'єкт повинен інкапсулювати цю інформацію;
3. в процесі виконання аутентифікації користувач повинен бачити прогрес логування (можливо анімаційне) зображення, яке інформуватиме про виконання аутентифікації і про те, що все йде нормально;
4. Після завершення логування, зробити об'єкт, створюваний в ході операції авторизації та повідомити іншу частину програми про те, що користувача залоговано.

Розробити домашній кінотеатр з можливістю переглядати фільм слухати радіо чи музику: керуючись наступними вимогами:

1. Надавати можливість користувачу фільм, пісню чи радіо-станцію;
2. Створити можливість користуватись машиною для попкорну, яка знаходиться в домашньому кінотеатрі;
3. Додати можливість зберігати поточну стану фільму або пісні для продовження перегляду в майбутньому;
4. Мати легкий механізм для увімкнення та вимкнення фільму.

РОЗДІЛ 2. МЕТОДИ ТА ЗАСОБИ РОЗВ'ЯЗАННЯ ЗАДАЧІ

Хто вигадав патерни? Це хороше, але не зовсім коректне запитання, бо патерни не вигадують, а радше «відкривають». Це не якісь супер-оригінальні рішення, а, навпаки, типові способи вирішення однієї і тієї ж проблеми, що часто повторюються з невеликими варіаціями.

Концепцію патернів вперше описав Крістофер Александер у книзі «Мова шаблонів. Міста. Будівлі. Будівництво». У книзі описано «мову» для проектування навколишнього середовища, одиниці якого — шаблони (або патерни, що ближче до оригінального терміна patterns) — відповідають на архітектурні запитання: якої висоти потрібно зробити вікна, скільки поверхів має бути в будівлі, яку площу в мікрорайоні відвести для дерев та газонів.

Ідея видалася привабливою четвірці авторів: Еріху Гаммі, Річарду Хелму, Ральфу Джонсону, Джону Вліссідесу. У 1995 році вони написали книгу «Патерни проектування: повторно використовувані елементи архітектури об'єктно-орієнтованого програмного забезпечення», до якої увійшли 23 патерни, що вирішують різні проблеми об'єктно-орієнтованого дизайну. Назва книги була занадто довгою, щоб хтось зміг її запам'ятати. Тому незабаром усі стали називати її «book by the gang of four», тобто «книга від банди чотирьох», а потім і зовсім «GOF book».

З того часу було знайдено десятки інших об'єктних патернів. «Патерновий» підхід став популярним і в інших галузях програмування, тому зараз можна зустріти різноманітні патерни також за межами об'єктного проектування.

2.1. Загальна інформація та структура шаблонів проектування

Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.

На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми.

Патерни часто плутають з алгоритмами, адже обидва поняття описують типові рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то патерн — це високорівневий опис рішення, реалізація якого може відрізнятися у двох різних програмах.

Якщо провести аналогії, то алгоритм — це кулінарний рецепт з чіткими кроками, а патерн — інженерне креслення, на якому намальовано рішення без конкретних кроків його отримання.

Описи патернів зазвичай дуже формальні й найчастіше складаються з таких пунктів:

- проблема, яку вирішує патерн;
- мотивація щодо вирішення проблеми способом, який пропонує патерн;
- структура класів, складових рішення;
- приклад однією з мов програмування;
- особливості реалізації в різних контекстах;
- зв’язки з іншими патернами.

Такий формалізм опису дозволив зібрати великий каталог патернів, додатково перевіривши кожен патерн на дієвість.

2.2. Критика шаблонів проектування

Патерни були описані більше 20-ти років тому, тому тільки ледачий не встиг кинути в них камінь. Давайте розглянемо найпопулярнішу критику.

Тому одними з проблем використання являються розпорки для слабкої мови програмування. Потреба в патернах з’являється тоді, коли люди вибирають для свого проекту мову програмування з недостатнім рівнем абстракції. В цьому випадку, патерни — це розпорки, які надають цій мові супер-здібності. Наприклад, патерн Стратегія в сучасних мовах можна реалізувати простою анонімною (лямбда) функцією.

Наступним недоліком являються неефективні рішення. Патерни намагаються стандартизувати підходи, які й так вже широко використовуються. Така стандартизація здається деяким людям догмою і вони починають усюди реалізовувати патерни «як в книжці», не пристосовуючи їх до реалій проекту.
Невиправдане застосування

«Якщо у тебе в руках молоток, то всі предмети навколо починають нагадувати цвяхи», - така проблема виникає у новачків, які тільки-но познайомилися з патернами. Вникнувши в патерни, людина намагається застосувати свої знання всюди. Навіть там, де можна було б обійтися більш простим кодом.

2.3. Класифікація шаблонів проектування

Патерни відрізняються за рівнем складності, деталізації та охоплення проектованої системи. Проводячи аналогію з будівництвом, ви можете підвищити безпеку на перехресті, встановивши світлофор, а можете замінити перехрестя цілою автомобільною розв’язкою з підземними переходами.

Найбільш низькорівневі та прості патерни — *ідіоми*. Вони не дуже універсальні, позаяк мають сенс лише в рамках однієї мови програмування.

Найбільш універсальні — *архітектурні патерни*, які можна реалізувати практично будь-якою мовою. Вони потрібні для проектування всієї програми, а не окремих її елементів.

Крім цього, патерни відрізняються і за призначенням. У цій книзі буде розглянуто три основні групи патернів:

- Породжуючі патерни піклуються про гнучке створення об’єктів без внесення в програму зайвих залежностей. До них відносяться: Фабричний метод, Будівельник, Абстрактна фабрика, Одинак, Прототип.
- Структурні патерни показують різні способи побудови зв’язків між об’єктами. Перелік: Адаптер, Компонувальник, Міст, Декоратор, Фасад, Легковаговик, Замісник.
- Поведінкові патерни піклуються про ефективну комунікацію між об’єктами. Перелік: Ланцюжок відавідальностей, Знімок, Команда, Стратегія, Ітератор, Посередник, Стан, Спостерігач, Відвідувач, Шаблонний метод.

2.4. Template Method

2.4.1. Суть патерна

Шаблонний метод — це поведінковий патерн проектування, який визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

Уявимо, ви пишете програму для дата-майнінгу в офісних документах. Користувачі завантажуватимуть до неї документи різних форматів (PDF, DOC, CSV), а програма повинна видобути з них корисну інформацію.

У першій версії ви обмежилися обробкою тільки DOC файлів. У наступній версії додали підтримку CSV. А через місяць «прикрутили» роботу з PDF документами.

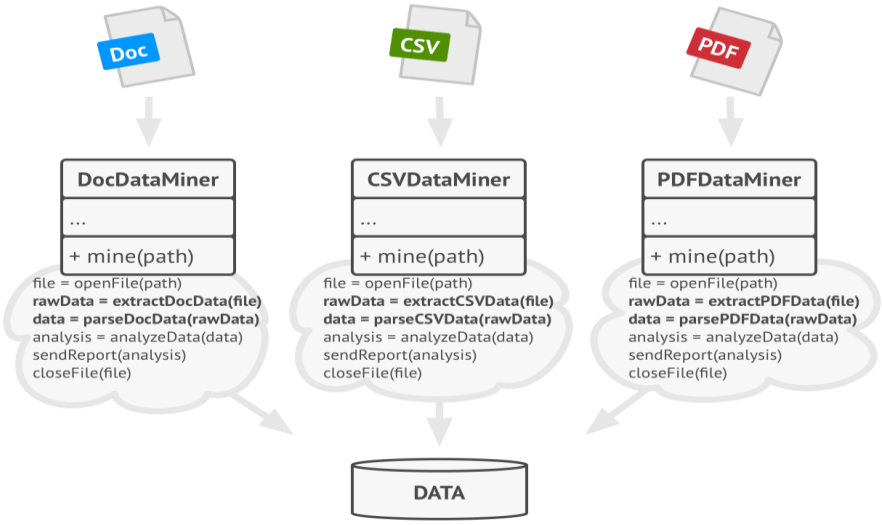


Рис 1. Класи дата-майнінгу

В якийсь момент ви помітили, що код усіх трьох класів обробки документів хоч і відрізняється в частині роботи з файлами, але містить досить багато спільного в частині самого видобування даних. Було б добре позбавитися від повторної реалізації алгоритму видобування даних у кожному з класів.

До того ж інший код, який працює з об’єктами цих класів, наповнений умовами, що перевіряють тип обробника перед початком роботи. Весь цей код можна спростити, якщо злити всі три класи в одне ціле або звести їх до загального інтерфейсу.

Патерн Шаблонний метод пропонує розбити алгоритм на послідовність кроків, описати ці кроки в окремих методах і викликати їх в одному *шаблонному* методі один за одним.

Це дозволить підкласам перевизначити деякі кроки алгоритму, залишаючи без змін його структуру та інші кроки, які для цього підкласу не є важливими.

У нашому прикладі з дата-майнінгом ми можемо створити загальний базовий клас для всіх трьох алгоритмів. Цей клас складатиметься з шаблонного методу, який послідовно викликає кроки розбору документів.

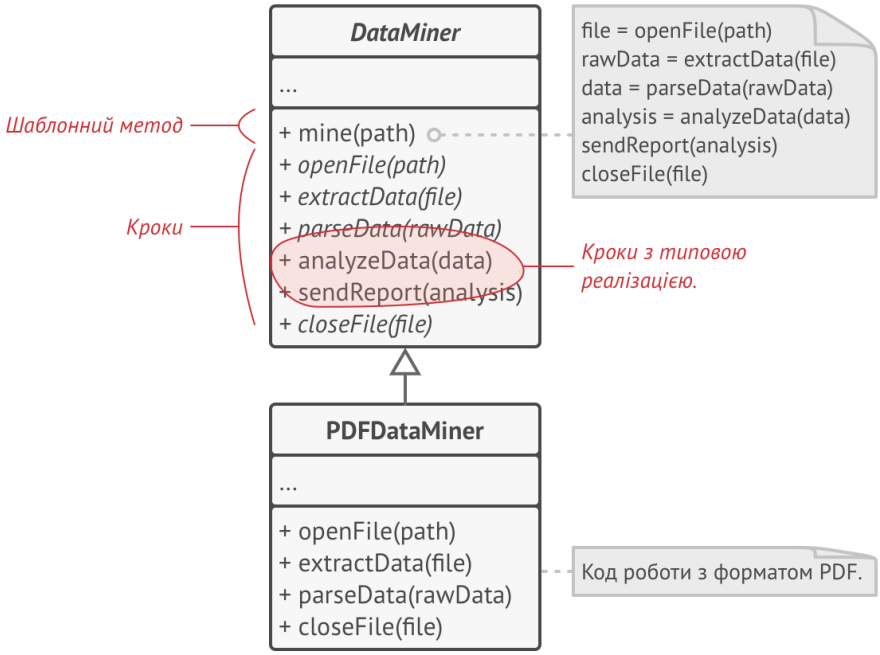


Рис 2. Діаграма дата-майнінгу

Для початку кроки шаблонного методу можна зробити абстрактними. З цієї причини усі підкласи повинні будуть реалізувати кожен з кроків по-своєму. В нашому випадку всі підкласи вже містять реалізацію кожного з кроків, тому додатково нічого робити не потрібно.

Справді важливим є наступний етап. Тепер ми можемо визначити спільну поведінку для всіх трьох класів і винести її до суперкласу. У нашому прикладі кроки відкривання та закривання документів відрізнятимуться для всіх підкласів, тому залишаться абстрактними. З іншого боку, код обробки даних, однаковий для всіх типів документів, переїде до базового класу.

Як бачите, у нас з'явилося два типа кроків: *абстрактні*, що кожен підклас обов'язково має реалізувати, а також кроки *з типовою реалізацією*, які можна перевизначити в підкласах, але це не обов'язково.

Але є ще й третій тип кроків — *хуки*. Це опціональні кроки, які виглядають як звичайні методи, але взагалі не містять коду. Шаблонний метод залишиться робочим, навіть якщо жоден підклас не перевизначить такий хук. Підсумовуючи сказане, хук дає підкласам додаткові точки «вклинювання» в хід шаблонного мету.

2.4.2. Структура

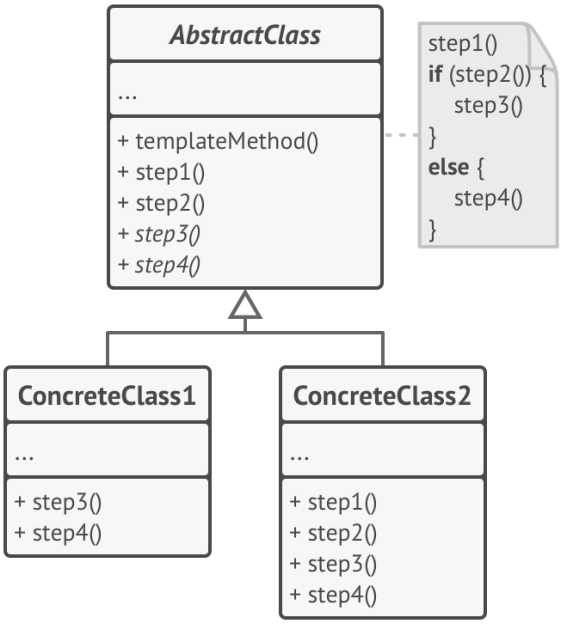


Рис 3. Структура патерну Шаблонний метод

Абстрактний клас визначає кроки алгоритму й містить шаблонний метод, що складається з викликів цих кроків. Кроки можуть бути як абстрактними, так і містити реалізацію за замовчуванням.

Конкретний клас перевизначає деякі або всі кроки алгоритму. Конкретні класи не перевизначають сам шаблонний метод.

Якщо підкласи повинні розширювати базовий алгоритм, не змінюючи його структури.

Шаблонний метод дозволяє підкласами розширювати певні кроки алгоритму через спадкування, не змінюючи при цьому структуру алгоритмів, оголошену в базовому класі.

Якщо у вас є кілька класів, які роблять одне й те саме з незначними відмінностями. Якщо ви редагуєте один клас, тоді доводиться вносити такі ж виправлення до інших класів.

Патерн шаблонний метод пропонує створити для схожих класів спільний суперклас та оформити в ньому головний алгоритм у вигляді кроків. Кроки, які відрізняються, можна перевизначити у підкласах.

Це дозволить прибрати дублювання коду в кількох класах, які відрізняються деталями, але мають схожу поведінку.

2.4.3 Кроки реалізації

1. Вивчіть алгоритм і подумайте, чи можна його розбити на кроки. Вирішіть, які кроки будуть стандартними для всіх варіацій алгоритму, а які можуть бути змінюваними.
2. Створіть абстрактний базовий клас. Визначте в ньому шаблонний метод. Цей метод повинен складатися з викликів кроків алгоритму. Є сенс у тому, щоб зробити шаблонний метод фінальним, аби підкласи не могли перевизначити його (якщо ваша мова програмування це дозволяє).
3. Додайте до абстрактного класу методи для кожного з кроків алгоритму. Ви можете зробити ці методи абстрактними або додати якусь типову реалізацію. У першому випадку всі підкласи *повинні* будуть реалізувати ці методи, а в другому — тільки якщо реалізація кроку в підкласі відрізняється від стандартної версії.
4. Подумайте про введення хуків в алгоритм. Найчастіше хуки розташовують між основними кроками алгоритму, а також до та після всіх кроків.
5. Створіть конкретні класи, успадкувавши їх від абстрактного класу. Реалізуйте в них всі кроки та хуки, яких не вистачає.

2.4.4. Переваги та недоліки

\+ Полегшує повторне використання коду.

\- Ви жорстко обмежені скелетом існуючого алгоритму.

\- Ви можете порушити *принцип підстановки Барбари Лісков*, змінюючи базову поведінку одного з кроків алгоритму через підклас.

\- У міру зростання кількості кроків шаблонний метод стає занадто складно підтримувати.

2.4.5. Відносини з іншими патернами

Фабричний метод можна розглядати як окремий випадок Шаблонного методу. Крім того, Фабричний метод нерідко буває частиною великого класу з Шаблонними методами.

Шаблонний метод використовує спадкування, щоб розширювати частини алгоритму. Стратегія використовує делегування, щоб змінювати «на льоту» алгоритми, що виконуються. Шаблонний метод працює на рівні класів. Стратегія дозволяє змінювати логіку окремих об’єктів.

2.5. Asynchronous processing

2.5.1. Суть патерна

Шаблон Asynchronous Processing не дозволяє асинхронно обробляти процеси. Замість цього вони ставляться в чергу і потім обробляються асинхронно.

2.5.2. Структура та кроки реалізації

При проектуванні і реалізації шаблону Asynchronous Processing слід враховувати, що запити будуть керуватися обробником і розподілятися по обробних потоках, а також інші об'єкти повинні знати результат обробки.

Існує безліч способів, які дозволяють керувати запитами і розбивати діючі потоки, які їх обробляють. Під управлінням запитів тут приймається визначення того, коли запит не може бути опрацьований. Можливі наступні варіанти:

- запит негайно може бути направлений на обробку;
- обробка запиту відкладається на деякий певний час;
- обробка відкладається до тих пір, поки не буде виконана деяка умова.

Найпростіший метод синхронізації потоків заключається в тому, щоб запускати новий потік кожного разу, коли повинен бути опрацьований запит. Основна перевага такого підходу - його простота. Недоліком є те, що цей спосіб

забезпечує мінімальний контроль над управлінням запитами і виділенням потоків. Такий підхід дозволяє відкидати запит, але не здатний відкласти його обробку. Він не забезпечує ніякого реального механіму розподілу потоків або обмеження кількості потоків.

Виникне проблема, якщо численні сповіщення про події надійдуть приблизно в один і той же час. Якщо занадто велика кількість подій обробляється одночасно, то загальна кількість потоків може призвести до уповільнення обробки запитів або перервати її за браком ресурсів.

Проста політика розподілу потоків полягає в тому, щоб мати постійну кількість потоків, придатних для обробки подій, і довільним чином призначає події потокам, у міру того як один з них стає доступним. Це, по суті, є застосуванням шаблону Producer-Consumer. Потік, відповідальний за отримання запитів і приміщення їх в чергу, є виробником, а потоки, обробні запити, є споживачами.

Складніша політика розподілу потоків може бути реалізована за допомогою шаблону Thread Pool. Передача запитів пулу потоків дозволяє реалізувати більш складну політику розподілу потоків. У відповідь на деякий вимога можна змінювати кількість використовуваних потоків. Крім того, в залежності від суті запиту може вимірюватися пріоритет потоків.

Управління запитами може бути реалізовано за допомогою шаблону Scheduler. Сенс об'єкта-планувальника в тому, що він забороняє виконання потоку до тих пір, поки не буде виконана деяка умова. Застосування об'єкта-планувальника дозволяє відкласти обробку запиту на підставі практичні но будь-якого критерію.

Об'єкт Scheduler є активним, має свій власний потік. Він здатний розпізнавати, коли запит готовий до обробки, і передає його пулу по потоків незалежно від тих дій, які виконують будь-які інші потоки.

Якщо потрібно зберегти потоки, можна реалізувати об'єкт Scheduler як пасивний об'єкт, який не має власного потоку. Тоді кожен раз при передачі запиту об'єкту Scheduler цей об'єкт буде перевіряти, чи є у нього які-небудь запити, готові до передачі в пул потоків. Коли один з потоків пулу завершує обробку деякого запиту, він може (але не зобов'язаний) викликати метод об'єкта Scheduler, який перевірить наявність запитів, готових до обробки.

2.5.3. Відносини з іншими патернами

Шаблон Thread Pool, Producer-Consumer, Scheduler можуть використовуватись в реалізації Asynchronous Processing.

Класи обробники запитів часто виступають в ролі Фасаду, який приховує деталі від клієнтських класів.

Об’єкт, який віддає запит має мати змогу сповістити про виконання операції інші об’єкти в асинхронному порядку, тому даний об’єкт може використовувати шаблон Future для синхронізації інших операцій, які виконує даний клас.

Шаблон Active Object, описує спосіб об’єднання щаблона Future с щаблоном Asynchronous Processing.

2.6. Façade

2.6.1. Суть патерна

Фасад — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.

Якщо вашому коду доводиться працювати з великою кількістю об’єктів певної складної бібліотеки чи фреймворка. Ви повинні самостійно ініціалізувати ці об’єкти, стежити за правильним порядком залежностей тощо.

В результаті бізнес-логіка ваших класів тісно переплітається з деталями реалізації сторонніх класів. Такий код досить складно розуміти та підтримувати.

Фасад — це простий інтерфейс для роботи зі складною підсистемою, яка містить безліч класів. Фасад може бути спрощеним відображенням системи, що не має 100% тієї функціональності, якої можна було б досягти, використовуючи складну підсистему безпосередньо. Разом з тим, він надає саме ті «фічі», які потрібні клієнтові, і приховує все інше.

Фасад корисний у тому випадку, якщо ви використовуєте якусь складну бібліотеку з безліччю рухомих частин, з яких вам потрібна тільки частина.

Наприклад, програма, що заливає в соціальні мережі відео з кошенятками, може використовувати професійну бібліотеку для стискання відео, але все, що потрібно клієнтському коду цієї програми, — це простий метод encode(filename, format). Створивши клас з таким методом, ви реалізуєте свій перший фасад.

2.6.2. Структура

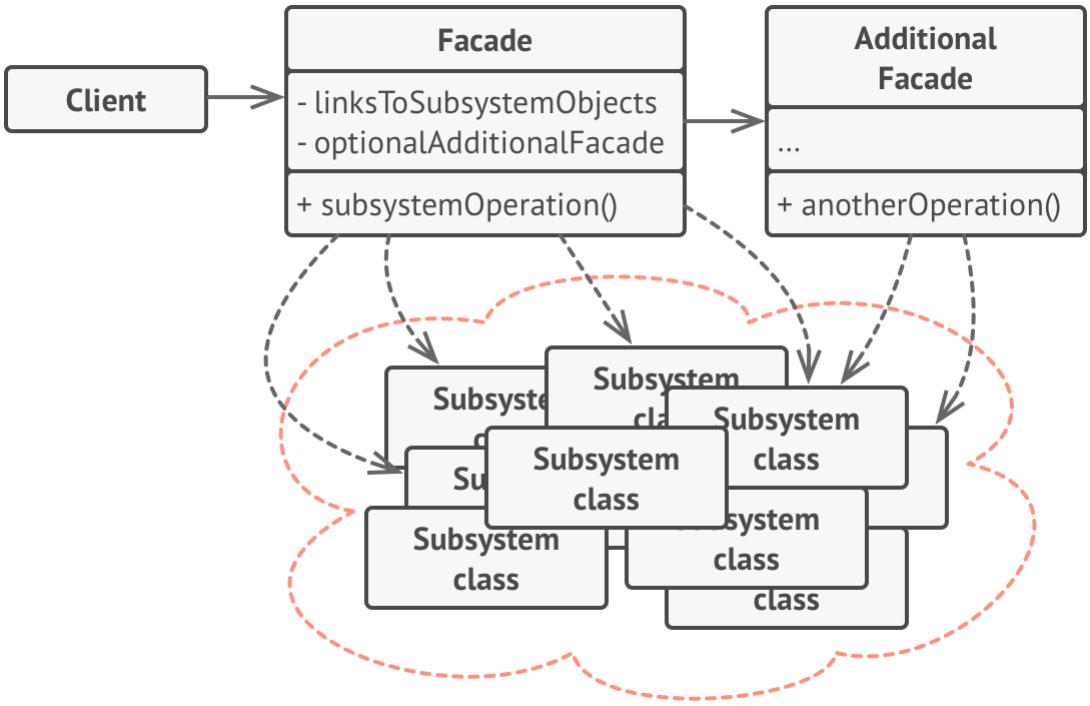


Рис 4. Структура патерну Фасад

Фасад надає швидкий доступ до певної функціональності підсистеми. Він «знає», яким класам потрібно переадресувати запит, і які дані для цього потрібні.

1. Додатковий фасад можна ввести, щоб не захаращувати єдиний фасад різнорідною функціональністю. Він може використовуватися як клієнтом, так й іншими фасадами.
2. Складна підсистема має безліч різноманітних класів. Для того, щоб примусити усіх їх щось робити, потрібно знати подробиці влаштування підсистеми, порядок ініціалізації об’єктів та інші деталі.

   Класи підсистеми не знають про існування фасаду і працюють один з одним безпосередньо.

3. Клієнт використовує фасад замість безпосередньої роботи з об’єктами складної підсистеми.

2.6.3. Застосування

Якщо вам потрібно надати простий або урізаний інтерфейс до складної підсистеми.

Часто підсистеми ускладнюються в міру розвитку програми. Застосування більшості патернів призводить до появи менших класів, але у великій кількості. Таку підсистему простіше використовувати повторно, налаштовуючи її кожен раз під конкретні потреби, але, разом з тим, використовувати таку підсистему без налаштовування важче. Фасад пропонує певний вид системи за замовчуванням, який влаштовує більшість клієнтів.

Якщо ви хочете розкласти підсистему на окремі рівні.

Використовуйте фасади для визначення точок входу на кожен рівень підсистеми. Якщо підсистеми залежать одна від одної, тоді залежність можна

спростити, дозволивши підсистемам обмінюватися інформацією тільки через фасади.

Наприклад, візьмемо ту ж саму складну систему конвертації відео. Ви хочете розбити її на окремі шари для роботи з аудіо й відео. Можна спробувати створити фасад для кожної з цих частин і примусити класи аудіо та відео обробки спілкуватися один з одним через ці фасади, а не безпосередньо.

2.6.4. Кроки реалізації

1. Визначте, чи можна створити більш простий інтерфейс, ніж той, який надає складна підсистема. Ви на правильному шляху, якщо цей інтерфейс позбавить клієнта від необхідності знати подробиці підсистеми.
2. Створіть клас фасаду, що реалізує цей інтерфейс. Він повинен переадресовувати виклики клієнта потрібним об’єктам підсистеми. Фасад повинен буде подбати про те, щоб правильно ініціалізувати об’єкти підсистеми.
3. Ви отримаєте максимум користі, якщо клієнт працюватиме тільки з фасадом. В такому випадку зміни в підсистемі стосуватимуться тільки коду фасаду, а клієнтський код залишиться робочим.
4. Якщо відповідальність фасаду стає розмитою, подумайте про введення додаткових фасадів.

2.6.5. Переваги та недоліки

- Ізолює клієнтів від компонентів складної підсистеми.
- Фасад ризикує стати божественим об’єктом, прив’язаним до всіх класів програми.

2.6.6. Відносини з іншими патернами

- Фасад задає новий інтерфейс, тоді як Адаптер повторно використовує старий. *Адаптер* обгортає тільки один клас, а *Фасад* обгортає цілу

підсистему. Крім того, *Адаптер* дозволяє двом існуючим інтерфейсам працювати спільно, замість того, щоб визначити повністю новий.

- Абстрактна фабрика може бути використана замість Фасаду для того, щоб приховати платформо-залежні класи.
- Легковаговик показує, як створювати багато дрібних об’єктів, а Фасад показує, як створити один об’єкт, який відображає цілу підсистему.
- Посередник та Фасад схожі тим, що намагаються організувати роботу багатьох існуючих класів.
    - *Фасад* створює спрощений інтерфейс підсистеми, не вносячи в неї жодної додаткової функціональності. Сама підсистема не знає про існування *Фасаду*. Класи підсистеми спілкуються один з одним безпосередньо.
    - *Посередник* централізує спілкування між компонентами системи. Компоненти системи знають тільки про існування *Посередника*, у них немає прямого доступу до інших компонентів.
- Фасад можна зробити Одинаком, оскільки зазвичай потрібен тільки один об’єкт-фасад.
- Фасад схожий на Замісник тим, що замінює складну підсистему та може сам її ініціалізувати. Але, на відміну від *Фасаду*, *Замісник* має такий самий інтерфейс, що і його службовий об’єкт, завдяки чому їх можна взаємозаміняти.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЇЇ ОПИС

3.1. Проектування діаграм класів

UML (англ. Unified Modeling Language) — уніфікована мова моделювання, використовується у парадигмі об'єктно–орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, яка називається UML–моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML–моделей як інтерпретованого коду можлива кодогенерація.

Різновид використання (англ. Use Case) — у розробці програмного забезпечення та системному проектуванні це опис поведінки системи, як вона відповідає на зовнішні запити. Іншими словами, різновид використання описує, «хто» і «що» може зробити з розглянутою системою.

Методика різновидів використання застосовується для виявлення вимог до поведінки системи, відомих також як функціональні вимоги.

На діаграмі показано, фрагмент UML діаграми з використанням патерну фасад - простий інтерфейс для роботи зі складною підсистемою, яка містить безліч класів.

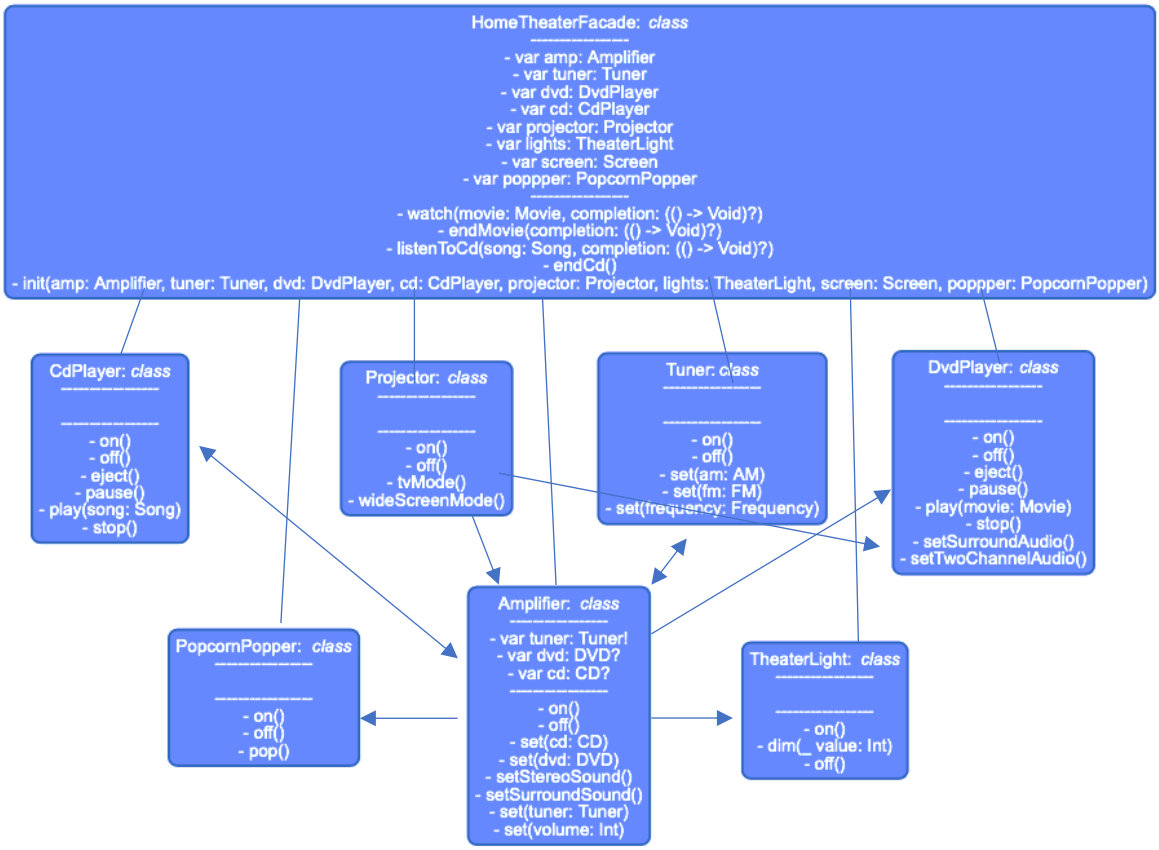


Рис 5. Діаграма патерну Фасад

На наступній діаграмі представлено фрагмет діаграми Template Method, який дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

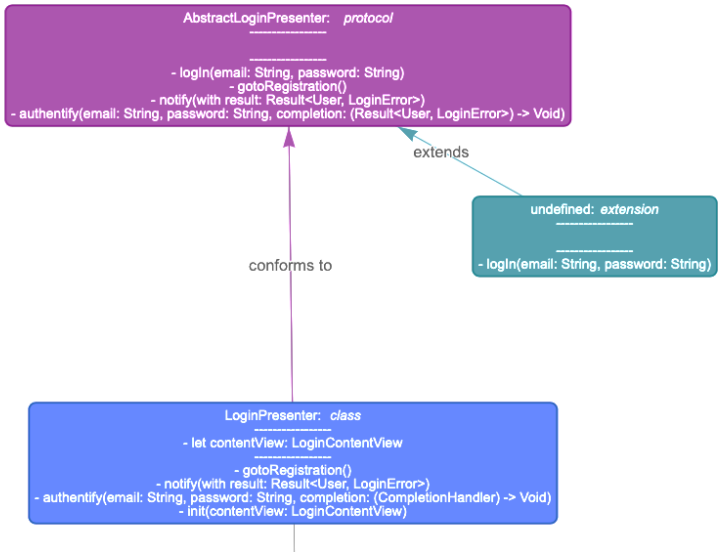


Рис 6. Діаграма патерна Шаблонний метод

І останнім фрагментом являється патерн асинхронна обробка:

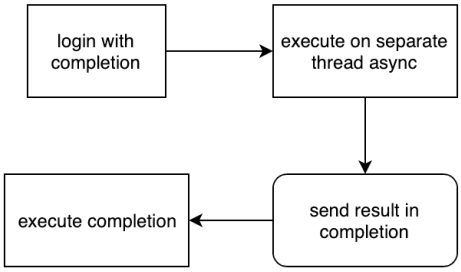


Рис 7. Діаграма патерна Асинхронне виконання

3.2. Проектування інтерфейсу користувача

Керуючись вимогами до продукту графічний інтерфейс користувача повинен містити:

- Вікно логування;
- Вікна реєстрації нового користувача;
- Головне меню домашнього кінотеатру;
- Список доступного контенту за категоріями;
- Вікно обраного контенту.

Рис 8. Графічний інтерфейс додатку

3.3 Програмна реалізація

Програмна реалізація здійснена на основі патерну MVP всієї архітектури додатку, для того щоб що відділити візуальне відображення та поведінку обробки подій у різні класи. До прикладу візьмемо компонент логування. Ми

маємо створений графічний інтерфейс та клас для роботи з UI, що екстендить протокол LoginContentViewProtocol:

```
protocol LoginContentView {
   func showHud()
   func hideHud()
   func setDefaultState()
   func setEmptyFieldsState()
   func setWrongPasswordState()
   func setunregisteredUserState()
   func navigate(to controller: UIViewController)
}
```

І використовує Презентер LoginViewPresenter застосовуючи себе як content view для презентера:

```
private lazy var presenter: AbstractLoginPresenter = {
      return LoginPresenter(contentView: self)
}()
```

Презентер для логування виконано на основі патерну Шаблонний метод. Ми маємо абстрактний протокол, який реалізує базовий інтерфейс виконання функції логування та не реалізовані кроки для виконання цієї операції. Як ми можемо побачити механізм логування складається з 2-х кроків, а саме аутентифікації користувача та сповіщення про результат обробки:

```
protocol AbstractLoginPresenter: class {
   func logIn(email: String, password: String)
   func gotoRegistration()
   func notify(with result: Result<User, LoginError>)
   func authentify(email: String, password: String, completion: (Result<User, LoginError>) ->
Void)
}
```

Після чого ми бачимо реалізацію послідовностей виконання операцій. Всі операції виконуються на окремому потоці аби не перешкоджати роботі графічного інтерфейсу та основано на потерні Асинхронна обробка. Після виконання етапу аутентифікації ми повертаємось на головний потік та повертаємо результат виконання для користувача.

```
extension AbstractLoginPresenter {
```

```
    func logIn(email: String, password: String) {
        DispatchQueue.global().asyncAfter(deadline: .now() + 3) { [weak self] in
            self?.authentify(email: email, password: password) { (result) in
                DispatchQueue.main.async {
                    self?.notify(with: result)
                }
            }
        }
    }
}
```

Після цього було створено конкретний клас, який реалізує абстрактний інтерфейс логування не змінюючи послідовність кроків алгоритму логування. А саме тому ми визначаємо конкретну поведінку для аутентифікації та сповіщення:

```
class LoginPresenter: AbstractLoginPresenter {
    private unowned let contentView: LoginContentView
    init(contentView: LoginContentView) {
        self.contentView = contentView
    }
    func notify(with result: Result<User, LoginError>) {
        contentView.hideHud()
        switch result {
        case .success(let user):
            contentView.navigate(to: ApplicationRoute.get(.homeTheater(user: user)))
        case .failure(let error):
            switch error {
            case .unregisteredUser: contentView.setunregisteredUserState()
            case .wrongPassword: contentView.setWrongPasswordState()
            case .emptyData: contentView.setEmptyFieldsState()
            }
        }
    }

    func authentify(email: String, password: String, completion: (CompletionHandler) -> Void) {
        guard !email.isEmpty, !password.isEmpty else {
            completion(.failure(.emptyData))
            return
        }

        guard let user = UserRepository.getUser(by: email) else {
            completion(.failure(.unregisteredUser))
            return
        }

        guard user.password == password else {
            completion(.failure(.wrongPassword))
            return
        }
```

```
        completion(.success(user))
    }
}
```

Як можна було помітити у поданих прикладах використано патерн шаблонний метод в компанії з Асинхолнною обробкою, яка реалізована на комплішин блоках.

Наступним патерном, реалізація якого була основною для програми – це Фасад. Тому у прикладі представлена реалізація класу HomeTheterFacade, де описано послідовність кроків для перегляду фільму чи слухання музики. Можна зробити висновок, що зручніше мати такий клас, який пов’язує між собою групу компонентів в одне ціле для зручного перевикористання в інших місцях:

```
class HomeTheaterFacade {
    private var amp: Amplifier
    private var tuner: Tuner
    private var dvd: DvdPlayer
    private var cd: CdPlayer
    private var projector: Projector
    private var lights: TheaterLight
    private var screen: Screen
    private var poppper: PopcornPopper

    init(amp: Amplifier, tuner: Tuner, dvd: DvdPlayer, … ) {
        self.amp = amp
            …
            …
    }

    func watch(movie: Movie, completion: (() -> Void)?) {
        print("Getting ready to watching movie...")
        poppper.on()
        poppper.pop()
        lights.dim(10)
        screen.down()
        projector.on()
        amp.set(dvd: dvd)
        amp.setSurroundSound()
        amp.set(volume: 5)
        dvd.on()
        dvd.play(movie: movie)
        completion?()
    }
```

```
    func endMovie(completion: (() -> Void)?) {
        print("Shutting movie theater down...")
        poppper.off()
        lights.on()
        screen.up()
        projector.off()
        amp.off()
        dvd.stop()
        dvd.eject()
        dvd.off()
        completion?()
    }

    func listenToCd(song: Song, completion: (() -> Void)?) {
        amp.set(cd: cd)
        amp.setStereoSound()
        cd.on()
        cd.play(song: song)
        completion?()

    }

    func endCd() {
        amp.off()
        cd.eject()
        cd.off()
    }
}
```

РОЗДІЛ 4. АНАЛІЗ РЕЗУЛЬТАТІВ РОЗРОБЛЕНОГО ДОДАТКУ

Проаналізувавши розроблену програму та її елементів на основі патернів Фасад, Шаблонний метод та Асинхронне виконання, можна зробити висновок, що за допомогою використання патернів ми отримали гнучкі модулі, які легко розширювати та перевикористовувати.

В результаті виконання завдання було розроблено сторінку логування з перевірками даних на валідність та показом анімації опрацювання запитів:

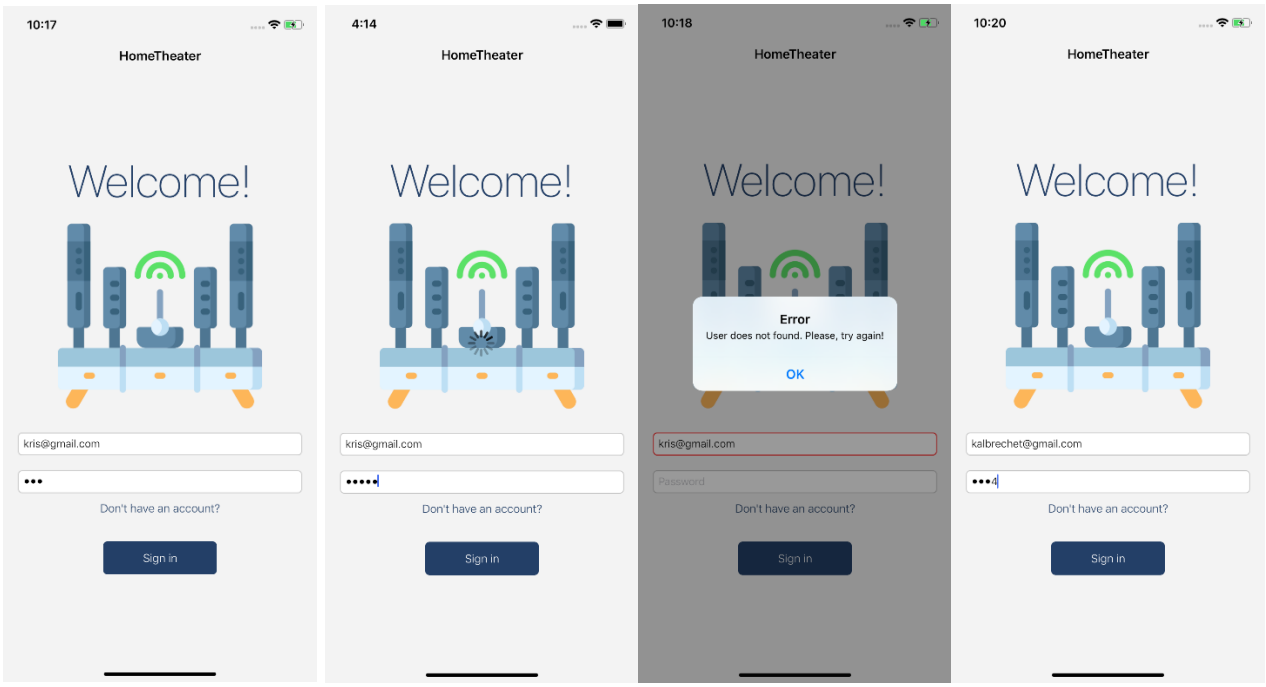
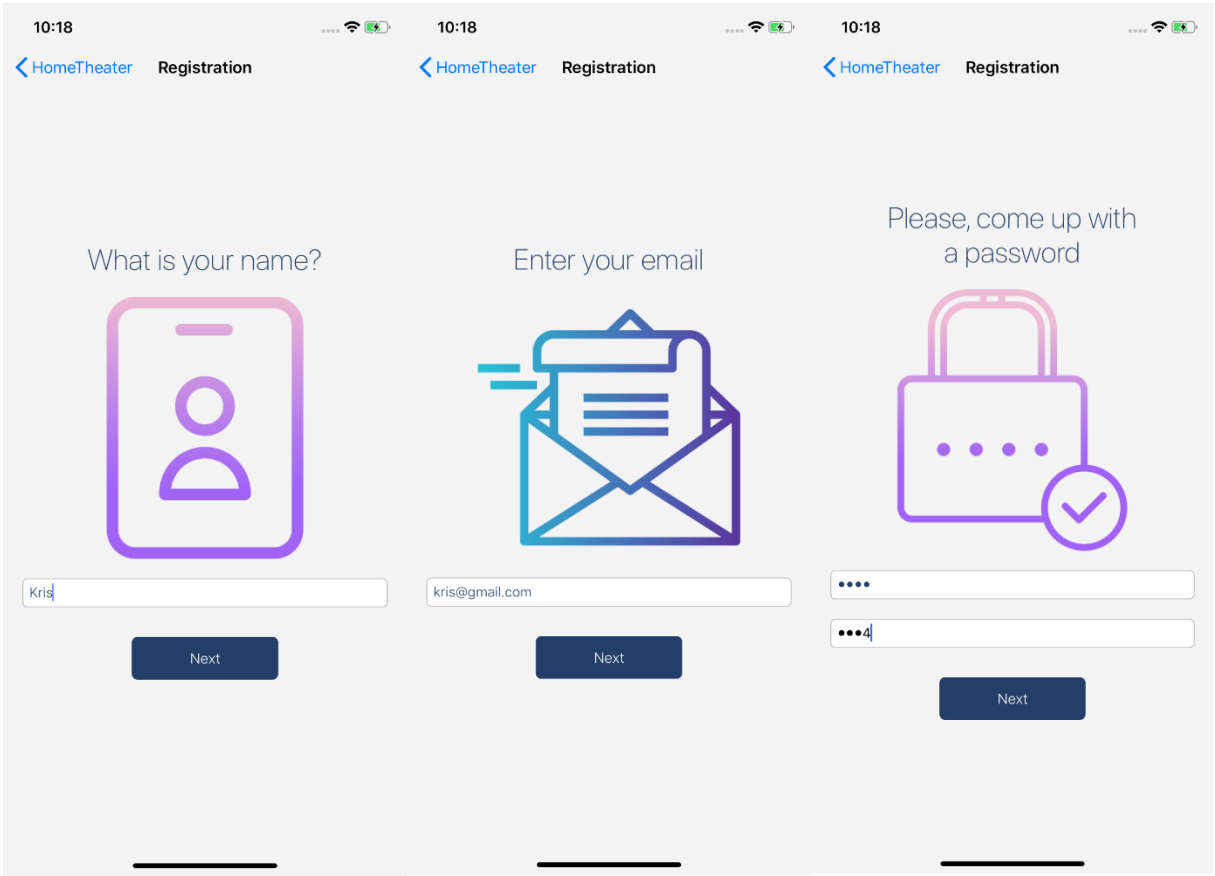


Рис 9. Результати тестування функції логування

Функція протестована з урахуванням всім можливих сценаріїв. Логування працює правильно та згідно з вимогами курсового проекту.

Наступним маємо огляд функціоналу реєстрації користувача в системі. Тут було розроблено 3 вікна з вводом інформації потрібної для реєстрації та використано Fade in/Fade out анімацію для зміни вікон. А також створено механізм валідації вводу користувача.



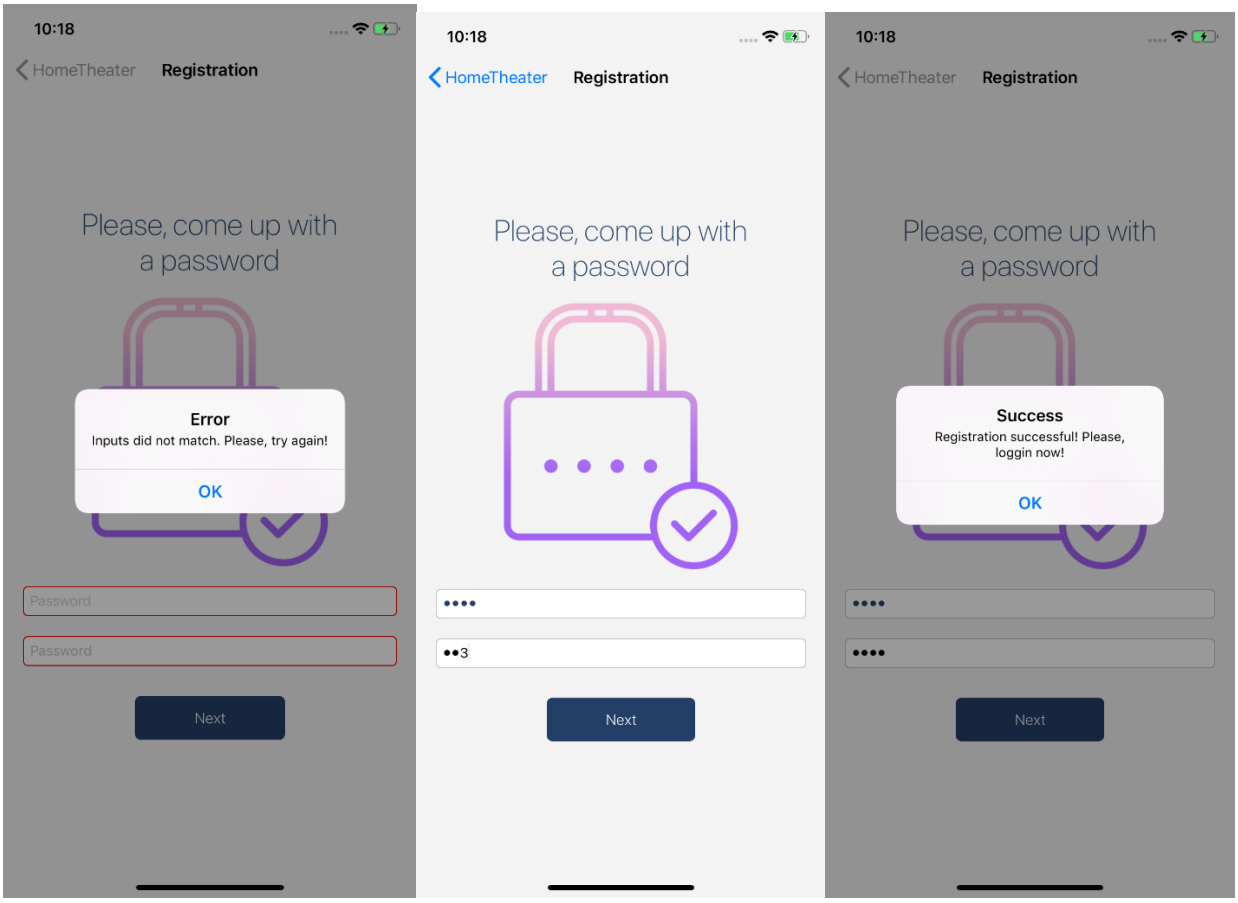


Рис 10. Механізм реєстрації

Після реєстрації та логування ми потрапляємо на головне вікно аплікації домашнього кінотеатру, де знаходиться вибір перегляду фільму або слухання музики:

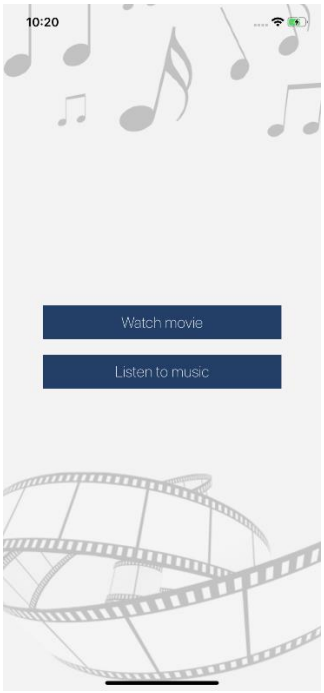


Рис 11. Вікно домашнього кінотеатру

Після вибору дії користувача ми потрапляємо на таблицю з можливим вибором контенту як фільмів так і музики:

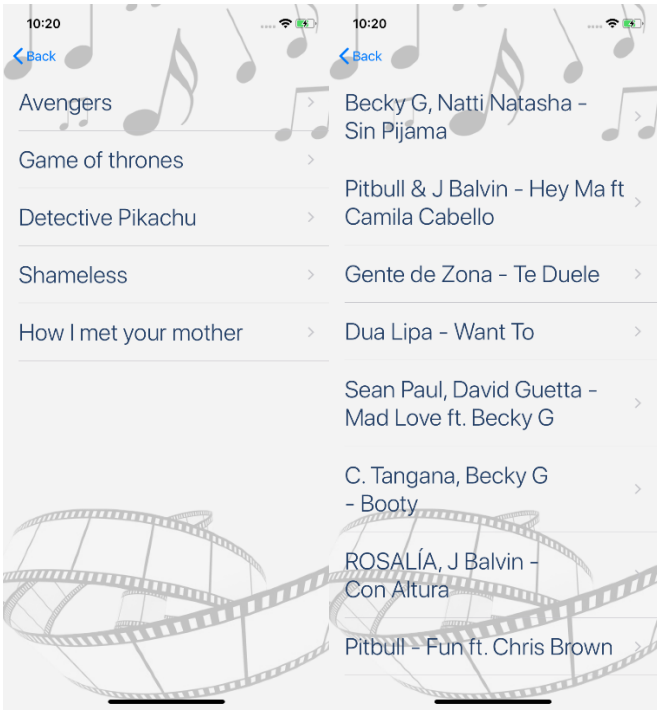


Рис 12. Вікно таблиці з контентом

І останнім вікном в програмі являється перегляд контенту з стрічкою прогресу, який допомагає користувачеві розуміти скільки залишилось до перегляду.

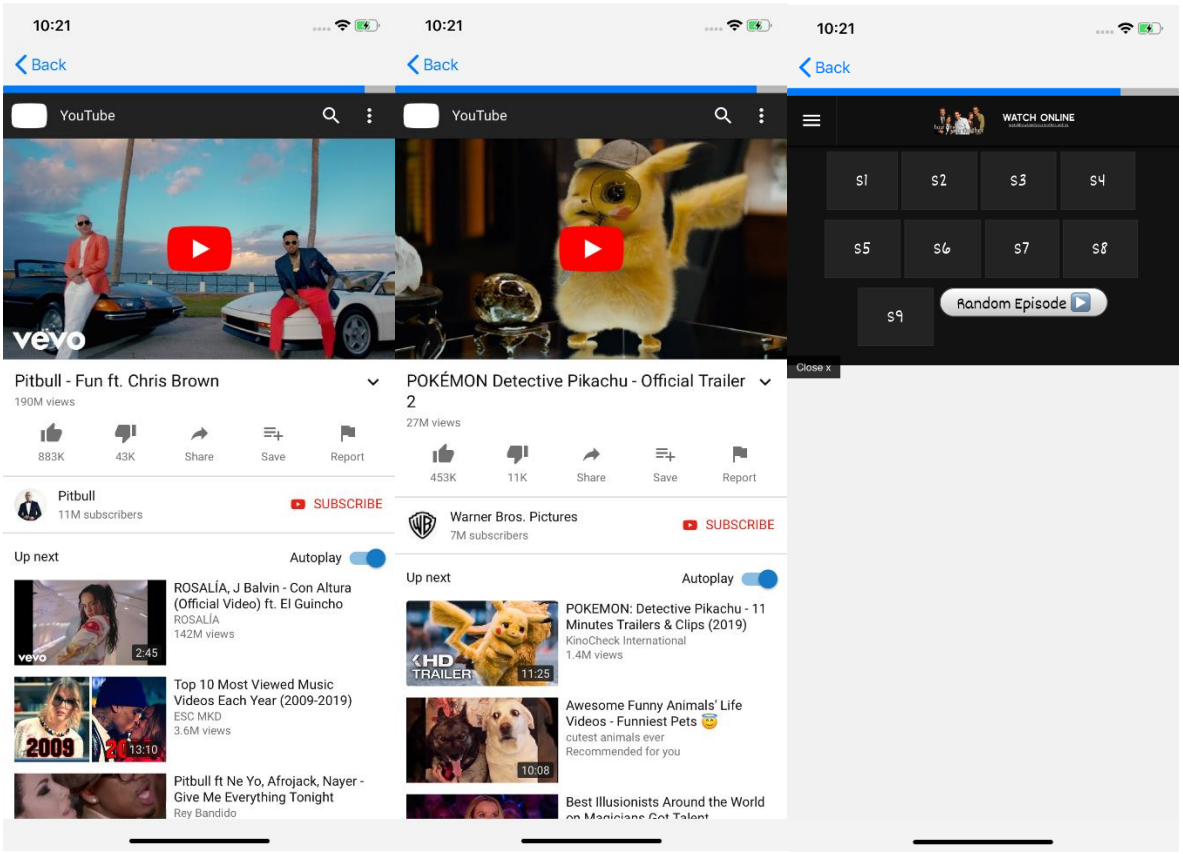


Рис 13. Вікно перегляду контенту

ВИСНОВОК

Під час написання даного курсового проекту було розроблено функціонал реєстрації та клас домашнього кінотеатру, застосовуючи механізми ООАП, а саме шаблони проектування Шаблонний метод, Асинхронне виконання та Фасад.

В ході виконання проекту було описано вище перелічені патерни – Фасад, Шаблонний метод та Асинхронна обробка, реалізовано UML діаграми та повністю функціонуючий продукт, а також проведено аналіз виконаних робіт.

В результаті виконання роботи було ознайомлено із розробленням додатку, який реалізовується на основі патернів програмування.

Створений програмний продукт протестований та працює належним чином.

ДОДАТОКИ
Додаток А

```
class ContentController: UIViewController, ContentViewProtocol {

    @IBOutlet weak var webView: WKWebView!
    @IBOutlet weak var progressView: UIProgressView!

    var content: Content!
    var isVideoContent: Bool!

    deinit {
        webView.removeObserver(self, forKeyPath: #keyPath(WKWebView.estimatedProgress))
    }

    private lazy var presenter: ContentPresenter = {
        return ContentPresenter(contentView: self, isVideoContent: isVideoContent)
    }()

    func showContent() {
        webView.load(URLRequest(url: URL(string: content.url)!))
        webView.addObserver(self, forKeyPath: #keyPath(WKWebView.estimatedProgress), options: .new, context: nil)
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        presenter.play(content: content)
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        presenter.stop()
    }

    override func observeValue(forKeyPath keyPath: String?,
                               of object: Any?, change: [NSKeyValueChangeKey : Any]?,
                               context: UnsafeMutableRawPointer?) {

        if keyPath == "estimatedProgress" {
            progressView.progress = 0.2 + Float(webView.estimatedProgress) * 0.8
        }
    }
}
class ContentTableController: UIViewController {

    @IBOutlet weak var tableView: UITableView!

    var isVideoContent: Bool = false

    private var contents = [Content]()

    override func viewDidLoad() {
        super.viewDidLoad()
        tableView.tableFooterView = UIView()
        tableView.estimatedRowHeight = 1
        tableView.rowHeight = UITableView.automaticDimension
        contents = isVideoContent ? MoviesRepository.get() : SongsRepository.get()
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if let controller = segue.destination as? ContentController {
            controller.content = sender as? Content
            controller.isVideoContent = isVideoContent
        }
    }
```

```
    private func present(content: Content) {
        performSegue(withIdentifier: "ShowContent", sender: content)
    }
}

extension ContentTableController: UITableViewDataSource {

    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return contents.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "ContentCell", for: indexPath)
        cell.textLabel?.text = contents[indexPath.row].title
        return cell
    }


}
extension ContentTableController: UITableViewDelegate {

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)
        present(content: contents[indexPath.row])
    }
}
protocol ContentViewProtocol: class {
    func showContent()
}

class ContentPresenter {

    private unowned let contentView: ContentViewProtocol
    private var isVideoContent: Bool
    private let theaterFacade = HomeTheaterFacade(amp: Amplifier(), tuner: Tuner(), dvd: DvdPlayer(), cd: CdPlayer(), projector:
Projector(), lights: TheaterLight(), screen: Screen(), poppper: PopcornPopper())


    init(contentView: ContentViewProtocol, isVideoContent: Bool) {
        self.contentView = contentView
        self.isVideoContent = isVideoContent
    }

    func stop() {
        theaterFacade.endMovie(completion: nil)
    }

    func play(content: Content) {
        if isVideoContent {
            theaterFacade.watch(movie: content) { [weak self] in
                self?.contentView.showContent()
            }
        } else {
            theaterFacade.listenToCd(song: content) { [weak self] in
                self?.contentView.showContent()
            }
        }
    }
}
class HomeTheaterController: UIViewController {

    @IBOutlet weak var movieButton: UIButton!
    @IBOutlet weak var musicButton: UIButton!

    private var user: User!

    func set(user: User) {
        self.user = user
    }

    override func viewWillAppear(_ animated: Bool) {
```

```
        super.viewWillAppear(animated)
        navigationController?.isNavigationBarHidden = true
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        navigationController?.isNavigationBarHidden = false
    }

    @IBAction func onListenToMusic(_ sender: Any) {
        performSegue(withIdentifier: "ShowList", sender: false)
    }

    @IBAction func onWatchMovie(_ sender: Any) {
        performSegue(withIdentifier: "ShowList", sender: true)
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if let controller = segue.destination as? ContentTableController {
            controller.isVideoContent = (sender as! Bool)
        }
    }
}
typealias Movie = Content
typealias Song = Content

class HomeTheaterFacade {

    private var amp: Amplifier
    private var tuner: Tuner
    private var dvd: DvdPlayer
    private var cd: CdPlayer
    private var projector: Projector
    private var lights: TheaterLight
    private var screen: Screen
    private var poppper: PopcornPopper

    init(amp: Amplifier,
         tuner: Tuner,
         dvd: DvdPlayer,
         cd: CdPlayer,
         projector: Projector,
         lights: TheaterLight,
         screen: Screen,
         poppper: PopcornPopper) {

        self.amp = amp
        self.tuner = tuner
        self.dvd = dvd
        self.cd = cd
        self.projector = projector
        self.lights = lights
        self.screen = screen
        self.poppper = poppper
    }

    func watch(movie: Movie, completion: (() -> Void)?) {
        print("Getting ready to watching movie...")
        poppper.on()
        poppper.pop()
        lights.dim(10)
        screen.down()
        projector.on()
        amp.set(dvd: dvd)
        amp.setSurroundSound()
        amp.set(volume: 5)
        dvd.on()
        dvd.play(movie: movie)
        completion?()
    }
```

```
    func endMovie(completion: (() -> Void)?) {
        print("Shutting movie theater down...")
        poppper.off()
        lights.on()
        screen.up()
        projector.off()
        amp.off()
        dvd.stop()
        dvd.eject()
        dvd.off()
        completion?()
    }

    func listenToCd(song: Song, completion: (() -> Void)?) {
        amp.set(cd: cd)
        amp.setStereoSound()
        cd.on()
        cd.play(song: song)
        completion?()

    }

    func endCd() {
        amp.off()
        cd.eject()
        cd.off()
    }
}
class PopcornPopper {

    func on() {
        print("PopcornPopper on...")
    }

    func off() {
        print("PopcornPopper off...")
    }

    func pop() {
        print("PopcornPopper pop...")
    }
}
class TheaterLight {

    func on() {
        print("TheaterLight on...")
    }

    func dim(_ value: Int) {
        print("TheaterLight dim - \(value)...")
    }

    func off() {
        print("TheaterLight off...")
    }
}

class DvdPlayer {

    func on() { print("DvdPlayer on...") }
    func off() {print("DvdPlayer off...")}
    func eject() {print("DvdPlayer eject...")}
    func pause() {}
    func play(movie: Movie) {print("DvdPlayer play...")}
    func stop() {print("DvdPlayer stop...")}
    func setSurroundAudio() {print("DvdPlayer set surround audio...")}
    func setTwoChannelAudio() {print("DvdPlayer set two channel audio...")}
}
class Projector {

    func on() {
```

```
        print("Projector on...")
    }
    func off() {
        print("Projector off...")
    }
    func tvMode() { }
    func wideScreenMode() { }
}
class CdPlayer {

    func on() {
        print("CdPlayer on...")
    }
    func off() {
        print("CdPlayer off...")
    }
    func eject() {
        print("CdPlayer eject...")
    }
    func pause() {}
    func play(song: Song) {
        print("CdPlayer play song...")
    }
    func stop() {
        print("CdPlayer stop...")
    }
}

class Amplifier {

    private var tuner: Tuner!
    private var dvd: DVD?
    private var cd: CD?

    func on() {
        print("Amplifier on...")
    }

    func off() {
        print("Amplifier off...")
    }

    func set(cd: CD) {
        print("Amplifier setiing cd...")
    }

    func set(dvd: DVD) {
        print("Amplifier setting dvd...")
    }

    func setStereoSound() {
        print("Amplifier setting stereo sound...")
    }

    func setSurroundSound() {
        print("Amplifier setting surround sound...")
    }

    func set(tuner: Tuner) {
        print("Amplifier setting tunner...")
    }

    func set(volume: Int) {
        print("Amplifier setting volume...")
    }
}
class Screen {

    func up() {
        print("Screen up...")
    }
```

```
    func down() {
        print("Screen down...")
    }
}
typealias AM = String
typealias FM = String
typealias Frequency = Double

class Tuner {

    func on() {
        print("Tunner on...")
    }
    func off() {
        print("Tunner off...")
    }

    func set(am: AM) {}
    func set(fm: FM) {}
    func set(frequency: Frequency) {}
}
class RegisterationController: UIViewController {

    @IBOutlet weak var nextButton: UIButton!
    @IBOutlet weak var additionalTextField: UITextField!
    @IBOutlet weak var mainTextField: UITextField!
    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var titleLabel: UILabel!
    @IBOutlet weak var scrollView: UIScrollView!

    private var presenter: RegisterationPresenter!

    override func viewDidLoad() {
        super.viewDidLoad()
        presenter = RegisterationPresenter(contentView: self)
        mainTextField.delegate = self
        additionalTextField.delegate = self
        title = "Registration"
        addKeyboardObservers()
    }

    deinit {
        NotificationCenter.default.removeObserver(self)
    }

    @IBAction func onNextDidTap(_ sender: Any) {
        presenter.onNext(currentData: mainTextField.text ?? "",
                         checker: additionalTextField.text ?? "")
    }
}

extension RegisterationController: UITextFieldDelegate {

    func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool {
        textField.setBorder()
        return true
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        (textField == mainTextField ? mainTextField : additionalTextField).resignFirstResponder()
        return true
    }
}

extension RegisterationController {

    func addKeyboardObservers() {
        NotificationCenter.default.addObserver(self,
                                               selector: #selector(onKeyboardWillShow(_:)),
                                               name: UIResponder.keyboardWillShowNotification,
                                               object: nil)
```

```
        NotificationCenter.default.addObserver(self,
                                               selector: #selector(onKeyboardWillHide(_:)),
                                               name: UIResponder.keyboardWillHideNotification,
                                               object: nil)
    }

    @objc func onKeyboardWillShow(_ notification: Notification) {
        guard let keyboardRect = notification
            .userInfo?["UIKeyboardFrameEndUserInfoKey"] as? CGRect else {
                return
        }

        let insets = UIEdgeInsets(top: 0, left: 0, bottom: keyboardRect.height, right: 0)
        scrollView.contentInset = insets
    }

    @objc func onKeyboardWillHide(_ notification: Notification) {
        scrollView.contentInset = .zero
    }
}

private extension RegisterationController {

    func resetState() {
        mainTextField.setBorder()
        additionalTextField.setBorder()
        mainTextField.clear()
        additionalTextField.clear()
    }

    func set(model: RegisterRouteModel) {
        titleLabel.text = model.title
        additionalTextField.isHidden = !model.isPassword
        imageView.image = model.image
        mainTextField.placeholder = model.placeholder
        mainTextField.isSecureTextEntry = model.isPassword
        additionalTextField.placeholder = model.placeholder
        resetState()
    }
}

extension RegisterationController: RegisterationContentView {

    func showHud() {
        MBProgressHUD.showAnimated(onView: view)
    }

    func hideHud() {
        MBProgressHUD.hideAnimated(forView: view)
    }

    func setSuccessfulRegState() {
        showAlert("Success",
                  with: "Registration successful! Please, loggin now!",
                  completion: { [weak self] in
            self?.popToLoginPage()
        })
    }

    func setDataDidNotMatchState() {
        mainTextField.setBorder(with: .red)
        additionalTextField.setBorder(with: .red)
        mainTextField.clear()
        additionalTextField.clear()
        showAlert(with: "Inputs did not match. Please, try again!")
    }

    func setUserEmptyDataState() {
        mainTextField.setBorder(with: .red)
        additionalTextField.setBorder(with: .red)
```

```
        showAlert(with: "Fields cannot be empty! Please, enter email and password!")
    }

    func setUserAlreadyExistState() {
        mainTextField.setBorder(with: .red)
        additionalTextField.setBorder(with: .red)
        showAlert(with: "User with such email has been already registered! Please, try another one!")
    }

    func setUndefinedErrorState() {
        showAlert(with: "Something went wrong! Please, try again later!")
    }

    func popToLoginPage() {
        navigationController?.popToRootViewController(animated: true)
    }

    func update(with model: RegisterRouteModel, animated: Bool) {
        guard animated else {
            set(model: model)
            return
        }

        UIView.animate(withDuration: 0.5, animations: {
            self.view.subviews.forEach({ $0.alpha = 0 })
        }) { _ in
            self.set(model: model)
            UIView.animate(withDuration: 0.5) {
                self.view.subviews.forEach({ $0.alpha = 1 })
            }
        }
    }
}

protocol RegisterationContentView: class {
    func update(with model: RegisterRouteModel, animated: Bool)
    func setSuccessfulRegState()
    func setDataDidNotMatchState()
    func setUserEmptyDataState()
    func setUndefinedErrorState()
    func setUserAlreadyExistState()
    func showHud()
    func hideHud()
}

class RegisterationPresenter {

    private unowned let contentView: RegisterationContentView
    private var currentIndex = 0
    private let models = [
        RegisterRouteModel(title: "What is your name?",
                           isPassword: false,
                           placeholder: "Name",
                           image: UIImage(named: "name")!),
        RegisterRouteModel(title: "Enter your email",
                           isPassword: false,
                           placeholder: "Email",
                           image: UIImage(named: "email")!),
        RegisterRouteModel(title: "Please, come up with a password",
                           isPassword: true,
                           placeholder: "Password",
                           image: UIImage(named: "password")!)
    ]

    private var user: User?

    init(contentView: RegisterationContentView) {
        self.contentView = contentView
        contentView.update(with: models[0], animated: false)
        user = User(name: "", email: "", password: "")
    }
```

```
    func onNext(currentData: String, checker: String) {
        guard !currentData.isEmpty else {
            contentView.setUserEmptyDataState()
            return
        }

        switch currentIndex {
        case 0:
            user?.name = currentData
        case 1:
            guard UserRepository.getUser(by: currentData) == nil else {
                contentView.setUserAlreadyExistState()
                return
            }
            user?.email = currentData

        case 2:
            guard currentData == checker else {
                contentView.setDataDidNotMatchState()
                return
            }
            user?.password = currentData

        default:
            break
        }

        if currentIndex == 2 {
            contentView.showHud()
            if let user = user {
                UserRepository.save(user: user)
                contentView.setSuccessfulRegState()
            } else {
                contentView.setUndefinedErrorState()
            }
            contentView.hideHud()
        } else {
            currentIndex += 1
            contentView.update(with: models[currentIndex], animated: true)
        }
    }
}
protocol LoginContentView: class {
    func showHud()
    func hideHud()
    func setDefaultState()
    func setEmptyFieldsState()
    func setWrongPasswordState()
    func setunregisteredUserState()
    func navigate(to controller: UIViewController)
}

class LoginController: UIViewController {

    @IBOutlet weak var welcomeLabel: UILabel!
    @IBOutlet weak var logInButton: UIButton!
    @IBOutlet weak var registerButton: UIButton!
    @IBOutlet weak var logoView: UIImageView!
    @IBOutlet weak var usernameField: UITextField!
    @IBOutlet weak var passwordField: UITextField!
    @IBOutlet weak var scrollView: UIScrollView!

    private var lightGray = UIColor.lightGray.withAlphaComponent(0.5)
    private lazy var presenter: AbstractLoginPresenter = {
        return LoginPresenter(contentView: self)
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        usernameField.delegate = self
```

```
        passwordField.delegate = self
        addKeyboardObservers()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        setDefaultState()
    }

    deinit {
        NotificationCenter.default.removeObserver(self)
    }

    @IBAction func onRegisterDidTap(_ sender: UIButton) {
        presenter.gotoRegistration()
    }

    @IBAction func onlogInDidTap(_ sender: UIButton) {
        showHud()
        presenter.logIn(email: usernameField.text ?? "",
                        password: passwordField.text ?? "")
    }
}

extension LoginController: UITextFieldDelegate {

    func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool {
        textField.setBorder()
        return true
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        (textField == usernameField ? usernameField : passwordField).resignFirstResponder()
        return true
    }
}

extension LoginController {

    func addKeyboardObservers() {
        NotificationCenter.default.addObserver(self,
                        selector: #selector(onKeyboardWillShow(_:)),
                        name: UIResponder.keyboardWillShowNotification,
                        object: nil)

        NotificationCenter.default.addObserver(self,
                        selector: #selector(onKeyboardWillHide(_:)),
                        name: UIResponder.keyboardWillHideNotification,
                        object: nil)
    }

    @objc func onKeyboardWillShow(_ notification: Notification) {
        guard let keyboardRect = notification
            .userInfo?["UIKeyboardFrameEndUserInfoKey"] as? CGRect else {
            return
        }

        let insets = UIEdgeInsets(top: 0, left: 0, bottom: keyboardRect.height, right: 0)
        scrollView.contentInset = insets
    }

    @objc func onKeyboardWillHide(_ notification: Notification) {
        scrollView.contentInset = .zero
    }
}

extension LoginController: LoginContentView {

    func showHud() {
        MBProgressHUD.showAnimated(onView: view)
    }
```

```
    func hideHud() {
        MBProgressHUD.hideAnimated(forView: view)
    }

    func setDefaultState() {
        usernameField.setBorder()
        passwordField.setBorder()
        usernameField.clear()
        passwordField.clear()
    }

    func setWrongPasswordState() {
        usernameField.setBorder()
        passwordField.setBorder(with: .red)
        showAlert(with: "Password didn't match. Please, try again!")
    }

    func setunregisteredUserState() {
        usernameField.setBorder(with: .red)
        passwordField.setBorder()
        passwordField.clear()
        showAlert(with: "User does not found. Please, try again!")
    }

    func setEmptyFieldsState() {
        if passwordField.text == "" { passwordField.setBorder(with: .red) }
        if usernameField.text == "" { usernameField.setBorder(with: .red) }
        showAlert(with: "Fields cannot be empty! Please, enter email and password!")
    }

    func navigate(to controller: UIViewController) {
        navigationController?.pushViewController(controller, animated: true)
    }
}
enum LoginError: Error {
    case wrongPassword
    case unregisteredUser
    case emptyData
}

protocol AbstractLoginPresenter: class {
    func logIn(email: String, password: String)
    func gotoRegistration()
    func notify(with result: Result<User, LoginError>)
    func authentify(email: String, password: String, completion: (Result<User, LoginError>) -> Void)
}

extension AbstractLoginPresenter {

    func logIn(email: String, password: String) {
        DispatchQueue.global().asyncAfter(deadline: .now() + 3) { [weak self] in
            self?.authentify(email: email, password: password) { (result) in
                DispatchQueue.main.async {
                    self?.notify(with: result)
                }
            }
        }
    }
}
typealias CompletionHandler = Result<User, LoginError>

class LoginPresenter: AbstractLoginPresenter {

    private unowned let contentView: LoginContentView

    init(contentView: LoginContentView) {
        self.contentView = contentView
    }

    func gotoRegistration() {
```

```
        contentView.navigate(to: ApplicationRoute.get(.registration))
    }

    func notify(with result: Result<User, LoginError>) {
        contentView.hideHud()
        switch result {
        case .success(let user):
            contentView.navigate(to: ApplicationRoute.get(.homeTheater(user: user)))
        case .failure(let error):
            switch error {
            case .unregisteredUser: contentView.setunregisteredUserState()
            case .wrongPassword: contentView.setWrongPasswordState()
            case .emptyData: contentView.setEmptyFieldsState()
            }
        }
    }

    func authentify(email: String, password: String, completion: (CompletionHandler) -> Void) {
        guard !email.isEmpty, !password.isEmpty else {
            completion(.failure(.emptyData))
            return
        }

        guard let user = UserRepository.getUser(by: email) else {
            completion(.failure(.unregisteredUser))
            return
        }

        guard user.password == password else {
            completion(.failure(.wrongPassword))
            return
        }

        completion(.success(user))
    }
}

struct RegisterRouteModel {
    let title: String
    let isPassword: Bool
    let placeholder: String
    let image: UIImage
}

struct Content {
    let url: String
    let title: String
}
struct User {
    var name: String
    var email: String
    var password: String
}
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        UINavigationBar.appearance().setBackgroundImage(UIImage(), for: .default)
        UINavigationBar.appearance().shadowImage = UIImage()
        UINavigationBar.appearance().backgroundColor = .clear
        UINavigationBar.appearance().isTranslucent = true
        return true
    }
}
```