2. Патерни MVC, MVP, MVVM

1.1 Цілі та задачі патерна Model-View-Controller

Ідеї MVC сформулював Трюгве Реєнскауг (Trygve Reenskaug) під час роботи у Xerox PARC наприкінці 70-х років. У ті часи для роботи з ЕОМ не обійтися без наукового ступеня і постійного вивчення об'ємної документації. Завдання, яке Реенскауг вирішував разом із групою дуже сильних розробників, полягала у тому, щоб спростити взаємодію пересічного користувача з комп'ютером. Необхідно було створити кошти, які з одного боку були б гранично простими та зрозумілими, а з іншого — давали б можливість керувати комп'ютером та складними програмами. Реєнскауг працював у команді, яка займалася розробкою портативного комп'ютера "для дітей різного віку" - Dynabook, а також мови SmallTalk під керівництвом Алана Кея (Alan Kay). Саме тоді й там закладалися поняття доброзичливого інтерфейсу. Робота Реєнськауга разом із командою багато в чому вплинула на розвиток сфери IT. Наведемо цікавий факт, який безпосередньо не відноситься до MVC, але ілюструє значущість тих розробок. У 2007 році після презентації Apple iPhone Алан Кей сказав: “Коли вийшов Macintosh, у Newsweek запитали, що я про нього думаю. Я сказав: це перший персональний комп'ютер, гідний критики. Після презентації Стів Джобс підійшов і запитав: чи гідний iPhone критики? І я відповів: Зробіть його розміром п'ять на вісім дюймів, і ви завоюєте мир”. Через 3 роки, 27 січня 2010 року, Apple представила iPad діагоналлю 9,7 дюйми. Тобто Стів Джобс майже буквально дотримувався поради Алана Кея. Проект, над яким працював Реннскауг, вівся протягом 10 років. А перша публікація про MVC від його творців побачила світ ще через 10 років. Мартін Фаулер, автор низки книг та статей з архітектури ПЗ, згадує, що він вивчав MVC за працюючою версією SmallTalk. Оскільки інформації про MVC з першоджерела довго не було, а також з інших причин, з'явилася велика кількість різних трактувань цього поняття. В результаті багато хто вважає MVC схемою або патерном проектування. Рідше MVC називають складеним патерном або комбінацією кількох патернів, що працюють спільно для реалізації складних програм. Але насправді, як було сказано раніше, MVC - це насамперед набір архітектурних ідей/принципів/підходів, які можна реалізувати різними способами з використанням різних шаблонів... Далі ми спробуємо розглянути основні ідеї, закладені в концепції MVC. Оскільки інформації про

MVC з першоджерела довго не було, а також з інших причин, з'явилася велика кількість різних трактувань цього поняття. В результаті багато хто вважає MVC схемою або патерном проектування. Рідше MVC називають складеним патерном або комбінацією кількох патернів, що працюють спільно для реалізації складних програм. Але насправді, як було сказано раніше, MVC - це насамперед набір архітектурних ідей/принципів/підходів, які можна реалізувати різними способами з використанням різних шаблонів... Далі ми спробуємо розглянути основні ідеї, закладені в концепції MVC. Оскільки інформації про MVC з першоджерела довго не було, а також з інших причин, з'явилася велика кількість різних трактувань цього поняття. В результаті багато хто вважає MVC схемою або патерном проектування. Рідше MVC називають складеним патерном або комбінацією кількох патернів, що працюють спільно для реалізації складних програм. Але насправді, як було сказано раніше, MVC - це насамперед набір архітектурних ідей/принципів/підходів, які можна реалізувати різними способами з використанням різних шаблонів... Далі ми спробуємо розглянути основні ідеї, закладені в концепції MVC. Рідше MVC називають складеним патерном або комбінацією кількох патернів, що працюють спільно для реалізації складних програм. Але насправді, як було сказано раніше, MVC - це насамперед набір архітектурних ідей/принципів/підходів, які можна реалізувати різними способами з використанням різних шаблонів... Далі ми спробуємо розглянути основні ідеї, закладені в концепції MVC. Рідше MVC називають складеним патерном або комбінацією кількох патернів, що працюють спільно для реалізації складних програм. Але насправді, як було сказано раніше, MVC - це насамперед набір архітектурних ідей/принципів/підходів, які можна реалізувати різними способами з використанням різних шаблонів... Далі ми спробуємо розглянути основні ідеї, закладені в концепції MVC.

1.1.1 Основні ідеї та принципи

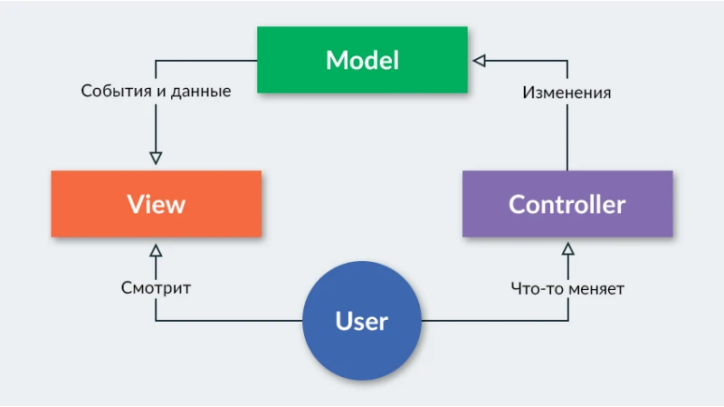
MVC - це набір архітектурних ідей і принципів для побудови складних інформаційних систем з інтерфейсом користувача;

MVC – це абревіатура, яка розшифровується так: Model-View-Controller.

MVC – це не патерн проектування. MVC - це саме **набір архітектурних ідей і принципів** для побудови складних систем з інтерфейсом користувача. Але для зручності, щоб щоразу не повторювати: "Набір архітектурних ідей…", ми називатимемо MVC патерном. Почнемо із простого. Що ж ховається за словами Model-View-Controller? При розробці систем з інтерфейсом користувача,

слідуючи патерну MVC потрібно розділяти систему на три складові. Їх, своєю чергою, можна називати модулями чи компонентами. Говори як хочеш, але поділи на три. Кожна складова компоненти матиме своє призначення. **Model.** Перша компонента/модуль – так звана модель. Вона містить всю бізнес-логіку програми. **View.** Друга частина системи – вид. Цей модуль відповідає за відображення даних користувачеві. Усі, що бачить користувач, генерується видом. **Controller.** Третьою ланкою цього ланцюга є контролер. У ньому зберігається код, який відповідає за обробку дій користувача (будь-яка дія користувача в системі обробляється в контролері). Модель – найнезалежніша частина системи. Така незалежна, що вона не повинна нічого знати про модулі Вид і Контролер. Модель настільки незалежна, що її розробники можуть практично нічого не знати про Від і Контролер. Основне призначення Виду - надавати інформацію з Моделі у зручному для сприйняття користувача форматі. Основне обмеження Виду — він не повинен змінювати модель. Основне призначення Контролера – обробляти дії користувача. Саме через Контролер користувач вносить зміни до моделі. Точніше дані, які зберігаються в моделі.

Наведемо схему:

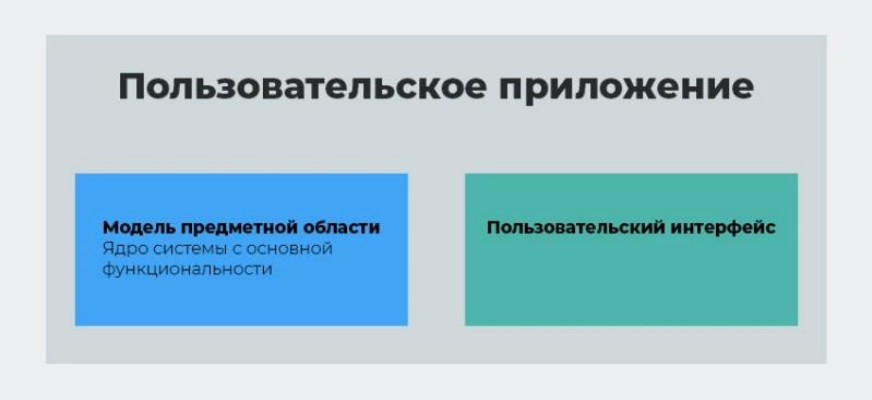


З цього можна зробити цілком логічний висновок. Складну систему слід розбивати на модулі. Опишемо коротко кроки, як можна досягти такого поділу.

Крок 1. Відокремити бізнес-логіку програми від інтерфейсу користувача

Ключова ідея MVC полягає в тому, що будь-яка програма з інтерфейсом користувача в першому наближенні можна розбити на 2 модулі: модуль, що відповідає за реалізацію бізнес-логіки програми, і інтерфейс користувача. У першому модулі буде реалізовано основний функціонал програми. Даний модуль буде ядром системи, в якому реалізується модель предметної області застосування. У концепції MVC цей модуль буде нашою буквою M, тобто.

моделлю. У другому модулі буде реалізований весь інтерфейс користувача, включаючи відображення даних користувачеві і логіку взаємодії користувача з додатком. Основна мета такого поділу – зробити так, щоб ядро системи (Модель у термінології MVC) могло незалежно розроблятися та тестуватися. Архітектура програми після такого поділу буде виглядати так:



Крок 2. Використовуючи шаблон Спостерігач, домогтися ще більшої незалежності моделі, а також синхронізації інтерфейсів користувача

Тут ми маємо 2 цілі:

1. Досягти ще більшої незалежності моделі.
2. Синхронізувати інтерфейси користувача.

Зрозуміти, що мається на увазі під синхронізацією інтерфейсів користувача, допоможе наступний приклад. Припустимо, ми купуємо квиток у кіно через інтернет і бачимо кількість вільних місць у кінотеатрі. Одночасно з нами купувати квиток у кіно може хтось ще. Якщо цей хтось купить квиток раніше за нас, нам би хотілося побачити, що кількість вільних місць на наш сеанс зменшилася. А тепер поміркуємо про те, як це може бути реалізовано всередині програми. Припустимо, у нас є ядро системи (наша модель) та інтерфейс (веб сторінка, на якій ми здійснюємо покупку). На сайті 2 користувача одночасно вибирають місце. Перший користувач купив квиток. Другому користувачеві необхідно відобразити цю інформацію на сторінці. Як це має статися? Якщо ми з ядра системи оновлюватимемо інтерфейс, наше ядро, наша модель, буде залежною від інтерфейсу. При розробці та тестуванні моделі доведеться пам'ятати різні способи оновлення інтерфейсу. Для цього необхідно реалізувати шаблон Спостерігач. З його допомогою модель розсилає повідомлення про зміни всім передплатникам. Інтерфейс, будучи таким передплатником, отримає повідомлення та оновиться. Шаблон Спостерігач дозволяє моделі з одного боку інформувати інтерфейс (вид і контролер) про те, що в ній відбулися зміни, а з іншого - фактично нічого про них не знати, і тим самим залишатися незалежною.

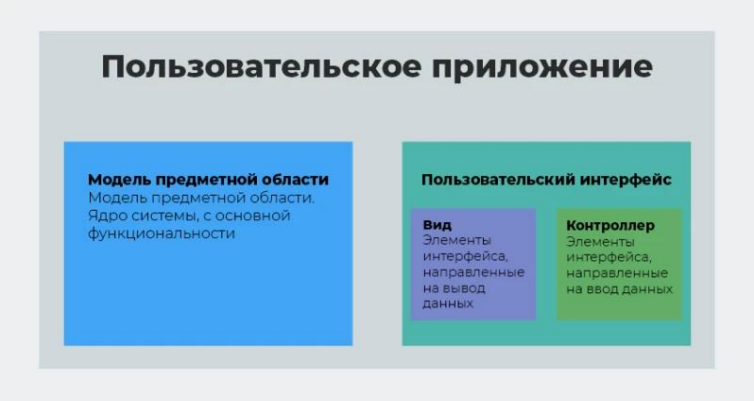
З іншого боку, це дозволить синхронізувати інтерфейси користувача. З його допомогою модель розсилає повідомлення про зміни всім передплатникам. Інтерфейс, будучи таким передплатником, отримає повідомлення та оновиться. Шаблон Спостерігач дозволяє моделі з одного боку інформувати інтерфейс (вид і контролер) про те, що в ній відбулися зміни, а з іншого - фактично нічого про них не знати, і тим самим залишатися незалежною. З іншого боку, це дозволить синхронізувати інтерфейси користувача. З його допомогою модель розсилає повідомлення про зміни всім передплатникам. Інтерфейс, будучи таким передплатником, отримає повідомлення та оновиться. Шаблон Спостерігач дозволяє моделі з одного боку інформувати інтерфейс (вид і контролер) про те, що в ній відбулися зміни, а з іншого - фактично нічого про них не знати, і тим самим залишатися незалежною. З іншого боку, це дозволить синхронізувати інтерфейси користувача.

Крок 3. Поділ інтерфейсу на Вид та Контролер

Продовжуємо ділити програму на модулі, але вже на нижчому рівні ієрархії. На цьому кроці інтерфейс користувача (який був виділений в окремий модуль на кроці 1) ділиться на вигляд і контролер. Важко провести сувору межу між видом і контролером. Якщо говорити про те, що вид це те, що бачить користувач, а контролер це механізм, завдяки якому користувач може взаємодіяти з системою, можна виявити деяке протиріччя. Елементи керування, наприклад, кнопки на веб-сторінці або віртуальна клавіатура на екрані телефону, це, по суті, частина контролера. Але вони так само видно користувачеві, як будь-яка частина виду. Тут швидше йдеться про функціональний поділ. Основне завдання інтерфейсу користувача - забезпечити взаємодію користувача з системою. Це означає, що в інтерфейсу всього 2 функції:

- виводити та зручно відображати користувачеві інформацію про систему;
- вводити дані та команди користувача в систему (передавати їх системі);

Дані функції визначають те, як потрібно ділити інтерфейс на модулі. У результаті, архітектура системи виглядає так:



Отже, у нас з'явився додаток із трьох модулів, які називаються Модель, Вид та Контролер. Резюмуємо:

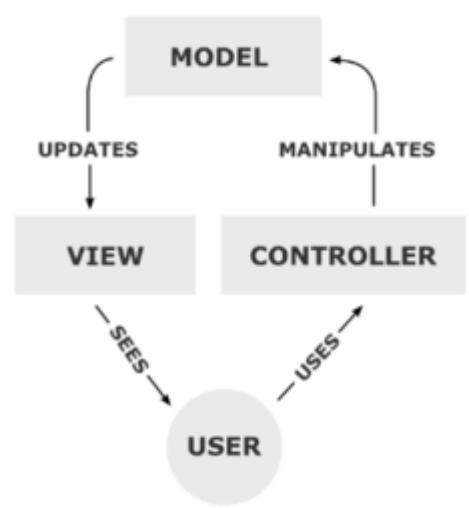
1. Дотримуючись принципів MVC, систему потрібно розділяти на модулі.
2. Найважливішим і найнезалежнішим модулем має бути модель.
3. Модель – ядро системи. Потрібна можливість розробляти та тестувати її незалежно від інтерфейсу.
4. Для цього на першому етапі сегрегації системи необхідно розділити її на модель та інтерфейс.
5. Далі, за допомогою шаблону Спостерігач, зміцнюємо модель в її незалежності і отримуємо синхронізацію інтерфейсів користувача.
6. Третім кроком ділимо інтерфейс на контролер та вигляд.
7. Все, що на введення інформації від користувача в систему - це контролер.
8. Все що на виведення інформації від системи до користувача це у вид.

1.1.2 Взаємозв'язок Виду та Контролера з Моделью

Коли користувач вводить інформацію через контролер, він тим самим вносить зміни до моделі. Принаймні користувач вносить зміни до даних моделей. Коли користувач отримує інформацію через елементи інтерфейсу (через Вигляд), користувач отримує інформацію про дані моделі. Як це відбувається? Через що Вид і Контролер взаємодіють із моделлю? Адже не може бути так, що класи Вида безпосередньо використовують методи класів Моделі для читання/запису даних, інакше ні про яку незалежність Моделі не може бути й мови. Модель представляє тісно пов'язаний між собою набір класів, до яких, по-хорошому, ні Вида, ні Контролера не повинно бути доступу. Для зв'язку моделі з видом і контролером необхідно реалізувати шаблон проектування Фасад. Фасад моделі буде тим самим прошарком між Моделью та інтерфейсом, через яку Вид отримує дані у

зручному форматі, а Контролер змінює дані, викликаючи потрібні методи фасаду. Схематично, зрештою, все виглядатиме так:

MVC

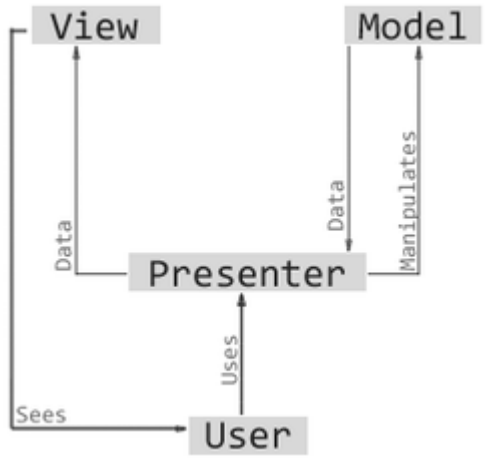


Модель (*Model*) надає дані та реагує на команди контролера, змінюючи свій стан.

Подання (*View*) відповідає відображення даних моделі користувачу, реагуючи зміни моделі.

Контролер (*Controller*) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

MVP



Модель (*Model*) - зберігає в собі всю бізнес-логіку , при необхідності отримує дані зі сховища.

Подання (*View*) — реалізує *відображення* даних, що звертається до Presenter за оновленнями. Користувач може взаємодіяти з її елементами, але коли якась подія віджету зачіпатиме логіку інтерфейсу, подання буде спрямовувати його презентеру .

Презентер (*Presenter*) - реалізує взаємодію між Моделью та Видом. Коли презентація повідомляє презентатор , що користувач щось зробив (наприклад, натиснув кнопку), презентатор приймає рішення про оновлення моделі та синхронізує всі зміни між моделлю та поданням . Презентер не спілкується із поданням безпосередньо. Натомість він спілкується через інтерфейс . Завдяки цьому презентер та модель можуть бути протестовані окремо.

MVVM

Модель (*Model*) є логікою роботи з даними та опис фундаментальних даних, необхідних для роботи програми.

Подання (*View*) — Виступає передплатником на подію зміни значень властивостей, що надаються Моделью Подання . У випадку, якщо в Моделі Уявлення змінилася якась властивість, то вона сповіщає всіх передплатників про це, і Уявлення , у свою чергу, вимагає оновленого значення властивості з Моделі Уявлення. Якщо користувач впливає на будь-який елемент інтерфейсу, Подання викликає відповідну команду, надану Моделлю Подання.

Модель Подання (*ViewModel*) - з одного боку, абстракція Подання , а з іншого - обгортка даних з Моделі, що підлягають зв'язуванню. Тобто, вона містить Модель, перетворену на Подання, а також команди, якими може користуватися Подання, щоб впливати на Модель.

Протягом останніх кількох років передовий підхід по розбиттю Android додатків на логічні компоненти еволюціонував. В значній мірі відійшли від монолітного Model View Controller (MVC) шаблону на користь більш модульних моделей, що найкраще тестуються.

Патерні Model View Presenter (MVP) та Model View ViewModel (MVVM) є одними з найбільш широко поширених альтернатив стандартному підходу, але розробники часто сперечаються, який краще підходить до Android [1].

MVC підхід розділяє додаток на макрорівні 3-х наборів обов'язків: даних, представлення та бізнес логіки. Робить велику роботу відділення моделі і представлення. Модель можна легко протестувати, бо вона не прив'язана ні до якого контексту. Але контролер прив'язаний настільки щільно до Android API, що важко виконувати модульне тестування. Контролери тісно пов'язані з представленням. При зміні представлення треба переписувати контролер.

За допомогою MVP зв'язок з представленням може відбуватися без прив'язки його до решти «контролерів». Це набагато чистіше. Легко протестувати блок логіки презенторів, тому що він не прив'язаний до Android API, що також дозволяє працювати з будь-яким іншим представленням. Презентори, так само, як контролери, схильні до збору додаткової бізнес- логіки. В якийсь момент, розробники часто залишаються з великими громіздкими класами, які важко розірвати один від одного.

MVVM з прив'язкою даних на Android має переваги раннього тестування і модульності, а також зменшення кількості коду, який ми повинні написати, щоб підключити представлення плюс модель. Модульне тестування стає ще простішим, тому що насправді немає ніякої залежності від представлення.

У розділенні додатку на модульні компоненти патерни MVP і MVVM краще виконують роботу, ніж MVC. Проте вони також додають додаткові складності для розробленого додатку. Для дуже простого додатку з одним або двома екранами, MVC може працювати краще. MVVM з прив'язкою даних є привабливим та робить модель більш реактивною і має менший код. Правильним підходом є також комбінування даних патернів в залежності від поставленої задачі.

1.2 Приклад використання патерна. Зв'язувальні вирази

Мова виразів дозволяє писати вирази, які з’єднують змінні з представленнями в макеті. Бібліотека зв’язування даних автоматично генерує класи, необхідні для зв’язування представлень у макеті з вашими об’єктами даних. Бібліотека надає такі функції, як імпорт, змінні та включає в себе, які можна використовувати у своїх макетах.

Ці функції бібліотеки бездоганно співіснують із наявними макетами. Наприклад, зв’язувальні змінні, які можна використовувати у виразах, визначені всередині елемента, dataякий є братом кореневого елемента макета інтерфейсу користувача. Обидва елементи загорнуті в layout тег, як показано в наступному прикладі:

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
  <data>
    <variable
      name="viewmodel"
      type="com.myapp.data.ViewModel" />
  </data>
  <ConstraintLayout... /> <!-- UI layout's root element -->
</layout>
```

Робота з спостережуваними об'єктами даних

Бібліотека зв’язування даних надає класи та методи для легкого спостереження за змінами даних. Вам не потрібно турбуватися про оновлення інтерфейсу користувача, коли змінюється базове джерело даних. Ви можете зробити свої змінні або їхні властивості доступними для спостереження. Бібліотека дозволяє зробити об’єкти, поля чи колекції доступними для спостереження.

Бібліотека зв’язування даних створює класи зв’язування, які використовуються для доступу до змінних і представлень макета. На цій сторінці показано, як використовувати та налаштовувати згенеровані класи зв’язування.

Для кожного виразу макета існує адаптер зв’язування, який виконує виклики фреймворку, необхідні для встановлення відповідних властивостей або слухачів. Наприклад, адаптер зв’язування може подбати про виклик setText()методу для встановлення властивості text або викликати setOnClickListener()метод для додавання слухача до події клацання. Найпоширеніші адаптери зв’язування, такі

як адаптери для android:textвластивості, використаної в прикладах на цій сторінці, доступні для використання в android.databinding.adaptersпакеті. Список поширених адаптерів прив'язки див . Ви також можете створити спеціальні адаптери, як показано в наступному прикладі:

```
@BindingAdapter("app:goneUnless")
fun goneUnless(view: View, visible: Boolean) {
   view.visibility = if (visible) View.VISIBLE else View.GONE
}
```

Бібліотека підтримки Android включає компоненти архітектури , які можна використовувати для розробки надійних програм, які можна тестувати та підтримувати. Ви можете використовувати компоненти архітектури з бібліотекою зв'язування даних, щоб ще більше спростити розробку вашого інтерфейсу користувача.

Бібліотека зв'язування даних підтримує двостороннє зв'язування даних. Нотація, яка використовується для цього типу зв'язування, підтримує можливість одночасно отримувати зміни даних у властивостях і прослуховувати оновлення користувача для цієї властивості.

1.3 Різниця між MVVM і MVC

Основна відмінність між MVVM і MVC полягає в їх використанні та моделях у розробці. Хоча вони певною мірою схожі, оскільки вони обидва допомагають розробникам розробляти та тестувати функції, між ними є деякі ключові відмінності. Ось деякі інші відмінності цих двох інструментів:

Визначення

MVVM розділяє різні компоненти процесу розробки на три категорії: модель, представлення та ViewModel. Зазвичай це включає розмітку коду або графічний інтерфейс користувача (GUI). MVC, або model-view-control — це спосіб, у який розробники поділяють програми на ці три компоненти. Це допомагає відрізняти бізнес-вимоги та правила від того, як користувачі працюють із програмою.

Шаблон для MVC складається з кількох кроків:

- Користувач взаємодіє з певним видом.
- Переглядач зв'язується з контролером, щоб ініціювати подію.

- Контролер оновлює модель.
- Модель надсилає повідомлення про те, що вигляд змінився.
- Представлення витягує дані моделі та оновлюється самостійно.

За допомогою MVVM користувач безпосередньо взаємодіє з представленням, яке працює з моделлю представлення. Якщо є якісь зміни, вони відбуваються безпосередньо між моделлю перегляду та самою моделлю.

Модель

Модель у MVC – це місце, де програма зберігає дані та будь-яку пов'язану логіку для команд. Це означає, що він містить інформацію та події, які відбуваються між подіями контролера. Наприклад, модель може дозволити вам змінювати та оновлювати дані в конкретній базі даних за допомогою контролера. У MVVM модель працює аналогічно, просто немає контролера, з яким вона працює, оскільки вона спілкується безпосередньо з моделлю представлення.

Перегляд

Розділ перегляду для обох — це те, як дані відображаються для користувача. У MVC це включає деякі або всі дані, які ви зберігаєте в моделі. Представлення або користувач запитує певні дані з даних моделі та отримує їх для відображення. Наприклад, клієнти, які відвідують веб-сайт, можуть бачити будь-які деталі форми на сторінці в поданні. Це схоже на MVVC, хоча перегляд також працює безпосередньо з моделлю перегляду, а не з будь-яким контролером.

Контролер

Основна відмінність у структурі обох полягає в тому, що MVC використовує контролер. Це інтерпретує дії користувача для отримання результатів. Наприклад, якщо користувач натискає гіперпосилання, контролер отримує доступ до даних для цього посилання. Ця частина також надсилає команди назад у представлення, щоб гарантувати, що користувач може переглянути результати своїх дій.

Вид-модель

Ще одна основна відмінність між MVVM і MVC полягає в тому, що MVVM використовує компонент представлення моделі. Це допомагає виконувати дії в поданні та працює безпосередньо з поданням в архітектурі. Як інший вид, він також представляє дані, що зберігаються в моделі. Це може відрізнятися від подання, оскільки воно може відображати не всі збережені дані.

Між цими двома функціями є деякі ключові відмінності. Ось деякі загальні особливості MVC:

Його можна легко використовувати для тестування та розширень.

Ви можете використовувати функції кількох інших програм.

Ви можете легко кодувати за допомогою HTML.

Ви можете простіше працювати з гіперпосиланнями та перенаправляти.

Ви можете зіставити унікальні URL-адреси.

Ось деякі з функцій, які ви можете отримати з MVVM:

Це настільна програма.

Він може зв'язувати дані з кількома інтерфейсами.

Ви можете використовувати шаблон спостерігача для зміни даних у моделі перегляду.

Ви можете використовувати його з кількома іншими програмами.

Точка входу

Ці два фреймворки мають різні точки входу. Наприклад, MVC використовує контролер як точку входу. Оскільки MVVM не має цього, він використовує представлення як точку входу.

стосунки

Обидва вони підтримують відносини «один до одного» та «один до багатьох». MVC має зв’язки «один-до-багатьох» між двома компонентами, представленням і контролером. MVVM має такий самий зв’язок, але це між компонентами представлення та моделі перегляду.

Ось деякі з переваг використання MVVM:

Тестування

MVVM можна легко протестувати та використовувати для розробників. Це може бути особливо корисним для індивідуального модульного тестування в додатку. Наприклад, під час тестування ви можете створювати сценарії для компонентів моделі та моделі перегляду окремо від використання перегляду.

За темою: яка роль інженера з розробки програмного забезпечення в тестуванні?

Повторне використання

Ви можете повторно використовувати багато компонентів для різних функцій у MVVM. Оскільки архітектура має лише слабозв’язану структуру, це дозволяє легко переміщувати, підтримувати чи повторно використовувати ці компоненти. Це може бути корисним, якщо ви хочете повторно використати загальну структуру компонентів або певні частини даних.

Логіка

MVVM використовує бізнес-логіку для виконання дій для користувача. Однією з переваг є те, що логіка відокремлена від інтерфейсу користувача. Це означає, що дії можуть відбуватися незалежно від дій, які виконує користувач під час доступу до перегляду.

Мінуси використання MVC

Ось деякі з недоліків, які ви можете побачити під час використання MVC:

Надскладний

У MVVM є кілька компонентів, які можуть бути надто складними. Наприклад, вам може знадобитися підтримувати велику кількість коду в різних компонентах. З його надійними функціями він може надто ускладнити просту розробку. Наприклад, якщо ви сподіваєтеся побудувати простий інтерфейс користувача, це може зайняти більше праці, ніж використання MVC.

Зчеплення

Хоча слабозв’язана структура має кілька переваг, це також може бути недоліком. Деякі розробники можуть віддати перевагу більшому зв’язку. Це особливо вірно у слабкому зв’язку між представленням і представленням-моделлю.

Плюси використання MVC

Ось деякі з переваг використання MVC:

Підтримка

MVC пропонує підтримку для нових і постійних клієнтів, що полегшує використання або початок. Це також може бути хорошим варіантом, якщо ви сподіваєтеся працювати в тестовому середовищі. Ви також можете отримати технічну підтримку залежно від компанії, де ви отримуєте програмне забезпечення для розробки.

Інтеграції

Є кілька простих інтеграцій з MVC, які можуть допомогти вам підключитися до інших платформ і програм. Це також стосується веб-програм, до яких ви можете підключитися. Перевага підключення до цих веб-програм полягає в тому, що ви можете отримати підтримку від інших розробників у цих різних областях.

Об'єктна незалежність

Кожен з об’єктів у MVC працює незалежно від інших. Це може бути особливо корисним, коли ви тестуєте свої програми, оскільки ви можете виконувати паралельне тестування або тестувати лише окремі компоненти в межах архітектури. Це також забезпечує спрощену роботу користувача в цілому.

Мінуси використання MVC

Ось деякі з недоліків використання MVC:

Бізнес-логіка

Одним із недоліків MVC є те, що інтерфейс користувача та бізнес-логіка мають зв’язки. Замість того, щоб функціонувати самостійно, користувачам потрібні дії для виконання команд. Це може означати створення нової бізнес-логіки кілька разів, щоб внести будь-які зміни в програму.

Вік

MVC старіший за MVVM, тому може спричинити кілька проблем. Наприклад, він може не відображатися належним чином із більш сучасними інтерфейсами користувача. Також може бути складніше виконувати або повторно використовувати тести в цій програмі.

Кілька потреб

Багатьом розробникам, які працюють у MVC, потрібні додаткові знання з технологіями та інструментами. Оскільки ви часто використовуєте це з іншими програмами, базові знання в цих областях можуть допомогти боротися з цим недоліком. Вам також може знадобитися кілька розробників, які знають інструмент для виконання паралельного тестування.