Министерства науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Разработка инвестиционного профиля

Студент:

Ефремов Дмитрий Владимирович

Группа:

P33212

Преподаватель: Ключев Аркадий Олегович

Санкт-Петербург

2024 г.

Оглавление

Введение

Актуальность работы

Разработка инвестиционного профиля на Android с Kotlin обеспечивает актуальный опыт, сочетая в себе применение передовых технологий и лучших практик в мобильной разработке. Проект является отличной возможностью для получения практического опыта в области Dependency Injection, MVX архитектуры, а также применения основ ООП и принципов SOLID. Такой подход дает возможность углубленно изучить технологии, которые в настоящее время являются стандартом в разработке мобильных приложений. Полученные навыки и знания предоставляют прочную основу для эффективной работы в сфере разработки в целом.

Цели и задачи работы

Цель - разработать приложение инвестиционный портфеля с отзывчивым и легким UI для обеспечения быстрого взаимодействия с акциями.

Задачи:

- разработать макет пользовательского интерфейса
- сверстать макет в Kotlin
- реализовать фронтенд на базе Андроид
- реализовать имитацию бэкенда
- реализовать тестирование приложения
- написать документацию с помощью Obsidian.

Техническое задание

Введение

Проект целенаправлен на разработку мобильного приложения, предназначенного для отображения информации о различных акциях с финансовых рынков. Пользователям предоставляется возможность получить актуальные данные о состоянии и стоимости акций, а также получить обзор инвестиционного портфеля.

Назначение разработки

Цель проекта состоит в создании мобильного приложения, которое будет предоставлять информацию о различных акциях, их категориях, а также общую статистику по инвестиционному портфелю. Основное назначение разработки заключается в обеспечении пользователей удобным доступом к подробной информации о портфеле и возможностью быстрого анализа данных.

Требования к программе или программному изделию

- Приложение должно быть разработано на языке программирования Kotlin.
- Применение архитектурного подхода MVX для обеспечения чистой разделенности ответственностей.
- Использование DI (Dependency Injection), для управления зависимостей в MVX
- Разделение акций по категориям для удобства навигации пользователя.
- Возможность просмотра общей сводки по инвестиционному портфелю, включая данные о профите, процентной доходности и других важных параметрах.

Функции системы

- Отображение списка акций с их краткой информацией и категориями.
- Общая вкладка с основной информацией о портфеле, включая общий профит, процентную доходность и другие ключевые показатели.
- Подробная информация о каждой акции при выборе соответствующего элемента.

Состав системы

Model:

- Классы данных
- Классы для представления информации об акциях.

View:

- Экран со списком акций и их краткой информацией.
- Экран с общей сводкой по инвестиционному портфелю.
- Всплывающее окно с подробной информацией о выбранной акции.

Presenter:

- Контроллеры для обработки взаимодействия пользователя с экранами.
- Логика обновления данных и их представления в представлении.

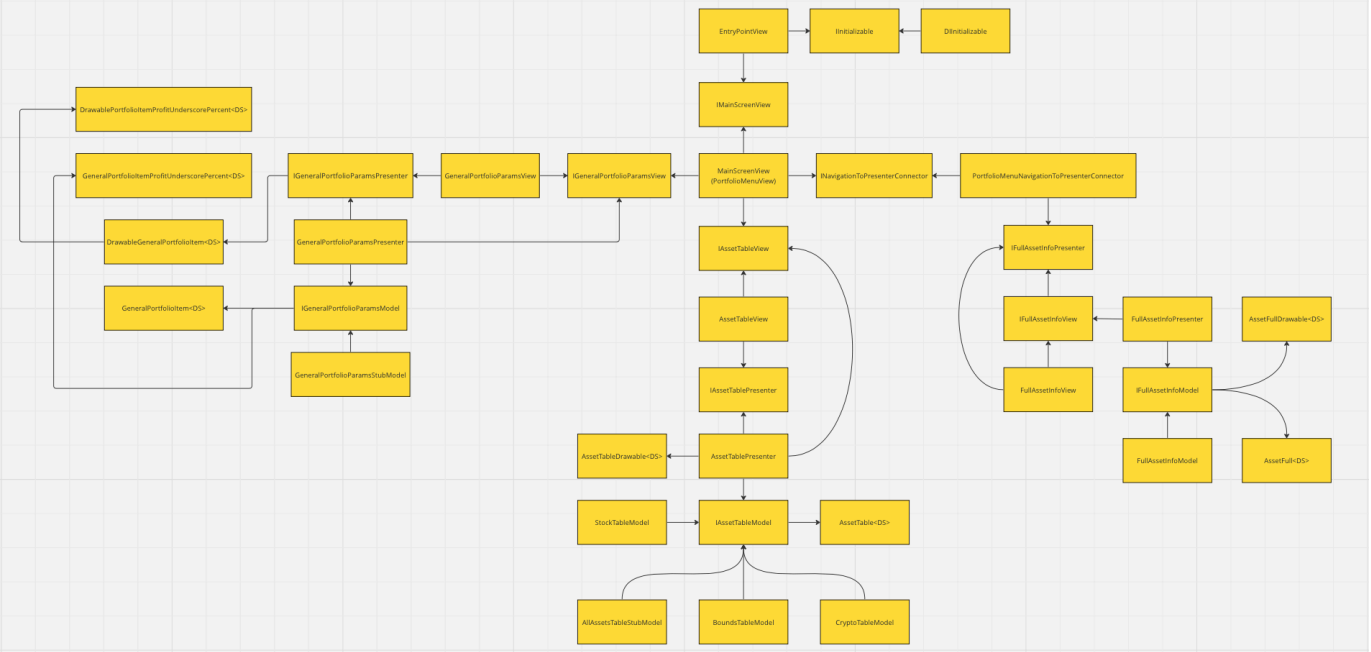
Характеристики системы

- Возможность расширения функционала приложения для добавления новых возможностей.
- Гибкость для подключения модулей.
- Легкость тестирования для обеспечения стабильной работы.

Оценка сроков, бюджета разработки и возможности разработчиков

Разработчик не имеет опыта работы с языком программирования Kotlin и мобильной разработкой, что может потребовать дополнительного времени на изучение и освоение новых концепций и инструментов.

Архитектура системы



Описание реализации

Задача входной точки (EntryPoint) заключается в запуске проекта. В нашем случае, она инициализирует только контейнеры внедрения зависимостей (DI), поэтому мы используем агрегацию и создаем экземпляр класса в конструкторе. При добавлении новых классов для инициализации, можно использовать паттерн "цепочка обязанностей".

```
internal interface IInitializable<Context> {
    fun init(context: Context)
}
```

PortfolioMenuView: Этот компонент должен передавать события нажатий на другие представления. Для этого создается коннектор с использованием обобщений для передачи аргументов.

```
interface INavigationToPresenterConnector<Binding, Presenter> where Binding: ViewBinding, Presenter:IBasePresenter {
    fun connect(layout:Binding, presenter: Presenter)
}
```

Модели представлены в виде интерфейсов, что обеспечивает легкую интеграцию с базой данных.

Пример модели:

```
interface IAssetTableModel {
    fun takeData(): ArrayList<AssetTable>
}
```

AssetTable представляет собой модуль взаимозаменяемых таблиц с различными типами активов. При вызове запроса на отображение новой таблицы, отображение должно соответственно меняться. Это достигается путем подмены модели. Когда на presenter приходит запрос, модель подменяется на необходимую, и представление перерисовывается.

Тестирование

Модульные

Этот тест проверяет, что функция takeWinLoseColor() правильно возвращает цвет убытка для отрицательного числа. Это важно, чтобы пользователь мог быстро определить, какие активы приносят ему убытки.

```
@Test
fun testNegativeTakeColor() {
    val loseColor = "#EE3870"
    val result = takeWinLoseColor(-100f)
    assertEquals(loseColor, result)
}
```

Аналогично первому тесту, только для положительных чисел и проверяет корректность возвращаемого цвета прибыли.

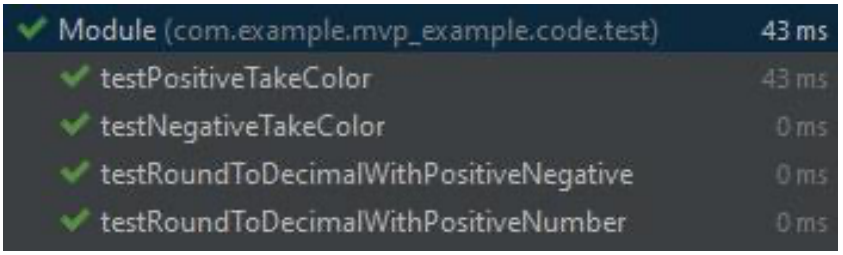
```
@Test
fun testPositiveTakeColor() {
    val winColor = "#8ED957"
    val result = takeWinLoseColor(100f)
    assertEquals(winColor, result)
}
```

Этот тест убеждается, что функция roundToDecimal() правильно округляет положительное число до указанного количества десятичных знаков. Это важно, чтобы данные отображались корректно и не было излишней точности.

```
@Test
fun testRoundToDecimalWithPositiveNumber() {
    val expected = 13.1f
    val result = roundToDecimal(13.123f, 1)
    assertEquals(expected, result)
}
```

Аналогично, только для отрицательных чисел.

```
@Test
fun testRoundToDecimalWithNegativeNumber() {
    val expected = -13.1f
    val result = roundToDecimal(-13.123f, 1)
    assertEquals(expected, result)
}
```



Имитация бэкенда

Этот тест проверяет ситуацию, когда возвращаемый имитатором бэкенда элемент данных имеет неправильный тип в одном из множества элементов. Ожидается, что будет сгенерировано исключение JsonSyntaxException.

```
@Test
fun testModelIncorrectTypeInOneOfMultiple() {
    assertFailsWith<JsonSyntaxException>("Expected JsonSyntaxException") {
        generalPortfolioIncorrectTest.takeCostElement()
    }
}
```

В этом тесте мы проверяем случай, когда возвращаемый элемент данных имитатором бэкенда имеет неправильный тип. Ожидается, что будет сгенерировано исключение JsonSyntaxException.

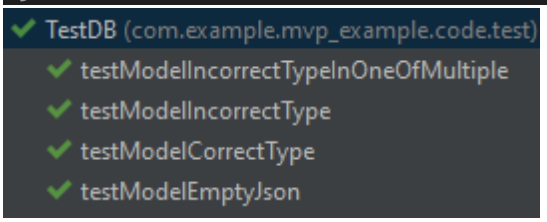
```
@Test
fun testModelIncorrectType() {
    assertFailsWith<JsonSyntaxException>("Expected JsonSyntaxException") {
        generalPortfolioIncorrectTest.takeProfitabilityElement()
    }
}
```

Данный тест проверяет сценарий, когда имитатор бэкенда возвращает пустой JSON. Ожидается, что будет сгенерировано исключение IllegalArgumentException.

```
@Test
fun testModelEmptyJson() {
    assertFailsWith<IllegalArgumentException>("Expected IllegalArgumentException") {
        generalPortfolioIncorrectTest.takeTotalProfitElement()
    }
}
```

Этот тест проверяет сценарий успешного возвращения данных из имитатора бэкенда с правильным типом. Ожидается, что операция выполнится без исключений.

```
@Test
fun testModelCorrectType() {
    val result = generalPortfolioCorrectModelTest.takeCostElement()
    assertDoesNotThrow { result }
}
```



Заключение

Были выполнены следующие этапы: разработка макета пользовательского интерфейса в Photoshop, верстка макета с использованием Kotlin, создание фронтенда приложения и имитации бэкенда. Были добавлены модульные тесты с применением фреймворка Koin для проверки корректности кода. Также была подготовлена документация в Obsidian.

Список литературы

1. Роберт Мартин: Чистая архитектура. Искусство разработки программного обеспечения
2. Руководство по языку Kotlin - https://metanit.com/kotlin/tutorial/
3. Документация по фреймворку Koin: - https://insert-koin.io/docs/reference/introduction/

Приложения

1. Github - https://github.com/mazai11111/InvestPortfolio
2. Архитектура - https://miro.com/app/board/uXjVNjx72k0=/
3. Obsidian documentation - https://drive.google.com/drive/folders/1STGe0cPgPftXscRoYlZQUHaEGpQLKPn0?usp=share_link
4. Dokka dependency connections - https://drive.google.com/drive/folders/1Jb9dN4uV5G3AQcQ3A9NJvZh5di63MTn7?usp=share_link