Отчет по анализу и рекомендациям для приложения "Деньги под Контролем" (Finance Analyzer)

Введение

Настоящий отчет представляет собой всесторонний анализ мобильного приложения "Деньги под Контролем" (Finance Analyzer), разработанного davidbugayov. Целью данного анализа является оценка текущей функциональности приложения, изучение его кодовой базы, проведение сравнительного анализа с ведущими конкурентами на рынке приложений для ведения личного бюджета, а также выработка детальных рекомендаций по улучшению как функциональных возможностей, так и архитектуры кодовой базы. Приложение "Деньги под Контролем" позиционируется как личный финансовый помощник, призванный упростить учет доходов и расходов, предоставить наглядную статистику и помочь пользователям в достижении финансовых целей. Анализ основан на информации, доступной в магазине приложений RuStore, а также на изучении открытого исходного кода проекта на GitHub.

1. Функциональный анализ приложения "Деньги под Контролем"

Приложение "Деньги под Контролем" (Finance Analyzer) представляет собой инструмент для управления личными финансами, доступный в RuStore. Согласно описанию в магазине приложений, его ключевые функциональные возможности включают:

- **Управление доходами и расходами:** Это основная функция, позволяющая пользователям регистрировать все финансовые операции.
- **Простой и красивый интерфейс:** Разработчик акцентирует внимание на интуитивно понятном дизайне, что делает приложение доступным даже для новичков в финансовом планировании.
- **Мгновенное добавление операций:** Подчеркивается скорость и удобство ввода данных, что является критически важным для приложений такого рода, поскольку пользователи часто вносят операции "на ходу".

- **Умная категоризация:** Приложение предлагает удобную систему категорий, где часто используемые категории всегда находятся под рукой, что ускоряет процесс классификации трат.
- **Наглядная статистика:** Для визуализации финансовой картины используются диаграммы и графики, позволяющие пользователям быстро оценить свои доходы и расходы.
- **Анализ привычек:** Функция, направленная на выявление финансовых трендов пользователя, что помогает улучшить финансовое здоровье и принимать более осознанные решения.
- **Виджеты и напоминания:** Эти элементы призваны обеспечить быстрый доступ к ключевой информации и своевременные уведомления о предстоящих платежах или достижении лимитов.
- **Полностью офлайн-режим:** Важное преимущество для пользователей, ценящих конфиденциальность и независимость от интернет-соединения. Все данные хранятся локально на устройстве, без необходимости регистрации или подписок.
- **Адаптивный дизайн:** Приложение оптимизировано для корректного отображения и работы на различных устройствах и размерах экранов.
- **Эффективность ресурсов:** Заявлено, что приложение потребляет минимум системных ресурсов, обеспечивая при этом максимальную пользу и надежность.
- **Постановка и достижение целей:** Пользователи могут устанавливать финансовые цели и отслеживать прогресс их достижения.

Отзывы пользователей

Анализ отзывов пользователей в RuStore позволяет получить представление о восприятии приложения реальными пользователями и выявить области для потенциальных улучшений. В целом, отзывы положительные, что свидетельствует о востребованности базовых функций и удобстве использования. Однако, были отмечены следующие моменты:

- **Удобство ввода данных:** Пользователь Олег отметил, что при вводе суммы в калькуляторе необходимо вручную переключаться на цифровую клавиатуру, что является неудобством. Он предложил автоматическое переключение, так как в этом поле требуются только цифры. Разработчик признал эту проблему и отметил, что калькулятор и иконки были добавлены, но с подкатегориями пока есть сложности.
- **Подкатегории:** Тот же пользователь Олег предложил брать подкатегории из примечаний, привязав это поле к статистике. Это указывает на потребность в

более детализированной и гибкой системе категоризации для глубокого анализа финансов.
- **Сравнение с конкурентами:** Олег также упомянул приложение "Бюджет 2.0" как пример хорошей реализации, что подчеркивает важность анализа конкурентов для выявления лучших практик.
- **Общая удовлетворенность:** Многие пользователи, такие как Юля и Анна, выразили высокую степень удовлетворенности, назвав приложение удобным и лучшим для учета финансов, что подтверждает его ценность для целевой аудитории.

Эти отзывы подтверждают, что, несмотря на общую положительную оценку, существуют конкретные области, где приложение может быть улучшено для повышения удобства использования и расширения аналитических возможностей. В частности, автоматизация ввода данных и улучшение системы категоризации являются ключевыми направлениями для дальнейшего развития.

2. Анализ кодовой базы и рекомендации по улучшению архитектуры

Изучение репозитория Financeanalyzer на GitHub выявило ряд аспектов, которые могут быть улучшены для повышения масштабируемости, поддерживаемости и общей надежности приложения. В файле `README-IMPROVEMENTS.md` разработчик уже обозначил некоторые из этих проблем и предложил решения. Ниже представлен более детальный анализ и рекомендации, основанные на этих предложениях и общих лучших практиках разработки мобильных приложений.

Архитектурные улучшения

2.1. Унификация репозиториев

Проблема: В текущей реализации проекта наблюдается избыточность в использовании интерфейсов для репозитория транзакций (`ITransactionRepository` и `TransactionRepository`). Это может привести к дублированию кода, усложнению логики и нарушению принципов чистого кода, таких как DRY (Don't Repeat Yourself) и Single Responsibility Principle (SRP).

Решение: * **Объединение интерфейсов:** Следует объединить `ITransactionRepository` и `TransactionRepository` в один унифицированный интерфейс. Это упростит структуру проекта и сделает ее более предсказуемой. * **Принцип Interface Segregation Principle (ISP):** Если существуют различные группы методов, используемые разными клиентами, можно

рассмотреть выделение более мелких, специализированных интерфейсов. Однако, в данном случае, если `ITransactionRepository` и `TransactionRepository` по сути представляют собой один и тот же набор операций с транзакциями, их объединение будет более целесообразным. * **Базовый интерфейс `BaseRepository<T>`:** Введение общего базового интерфейса `BaseRepository<T>` с универсальными методами CRUD (Create, Read, Update, Delete) операций позволит стандартизировать взаимодействие с различными типами данных и уменьшить дублирование кода при работе с другими сущностями в будущем (например, категориями, счетами). * **Перенос специфичных методов:** Все методы, специфичные для работы с транзакциями, должны быть инкапсулированы в `TransactionRepository`.

2.2. Централизация навигации

Проблема: Логика навигации в приложении распределена между различными компонентами, что делает ее сложной для отслеживания, тестирования и модификации. Частичное дублирование кода навигации может привести к ошибкам и несогласованности пользовательского опыта.

Решение: * **Создание `NavigationManager`:** Внедрение единого `NavigationManager` (или аналогичного компонента) позволит централизовать всю логику навигации. Это обеспечит единую точку входа для всех навигационных событий, упростит отладку и тестирование. * **Типизированные Actions:** Использование типизированных Actions (например, с помощью Safe Args в Android Navigation Component) вместо строковых маршрутов значительно повысит безопасность типов и уменьшит вероятность ошибок, связанных с опечатками в строковых идентификаторах. * **Поддержка Deep Linking:** Интеграция поддержки Deep Linking позволит пользователям переходить к определенным экранам приложения из внешних источников (например, веб-ссылок, уведомлений), улучшая пользовательский опыт и возможности взаимодействия. * **Navigation Result Callbacks:** Реализация механизма Navigation Result Callbacks позволит передавать данные между фрагментами или активити после завершения навигационной операции, что упростит взаимодействие между различными частями приложения.

2.3. Модульная архитектура

Проблема: Вся логика приложения сосредоточена в одном монолитном модуле `app`. Это затрудняет масштабирование проекта, усложняет параллельную разработку командами, увеличивает время сборки и снижает возможность переиспользования кода в других проектах или частях приложения.

Решение: * **Разделение на Gradle-модули:** Приложение должно быть разделено на логические Gradle-модули, каждый из которых отвечает за определенный аспект функциональности или слой архитектуры. Рекомендуемая структура модулей: * `core`: Содержит базовые компоненты, общие утилиты, расширения, константы и другие элементы, которые используются во всем приложении. * `data`: Отвечает за работу с данными, включая репозитории, источники данных (локальные базы данных, сетевые клиенты) и модели данных. * `domain`: Содержит бизнес-логику приложения, модели предметной области (entities) и Use Cases (интеракторы), которые определяют, как данные используются и обрабатываются. * `ui`: Включает общие компоненты пользовательского интерфейса, темы, стили и другие элементы, связанные с представлением. * `feature:transactions`, `feature:analytics`, `feature:budget` и т.д.: Отдельные функциональные модули, каждый из которых инкапсулирует логику и UI для конкретной функции (например, модуль для управления транзакциями, модуль для аналитики, модуль для бюджетирования). Это позволяет командам работать над разными функциями независимо и обеспечивает лучшую изоляцию кода.

2.4. Улучшение DI (Dependency Injection)

Проблема: Все зависимости в проекте объявлены в одном большом DI-модуле. Это делает модуль громоздким, сложным для понимания, тестирования и поддержки. Изменения в одной части могут непреднамеренно повлиять на другие.

Решение: * **Разделение AppModule:** Разделить существующий `AppModule` на более мелкие, логически сгруппированные модули, такие как `DatabaseModule` (для зависимостей базы данных), `RepositoryModule` (для репозиториев), `UseCaseModule` (для Use Cases) и т.д. Это улучшит читаемость и поддерживаемость кода. * **Koin Features:** Использование Koin features (или аналогичных механизмов в других DI-фреймворках, таких как Hilt/Dagger) для управления зависимостями с ограниченным временем жизни (например, для зависимостей, которые должны быть доступны только в пределах определенного жизненного цикла компонента, такого как Activity или Fragment). * **Scope-driven инъекция:** Внедрение scope-driven инъекции для feature-модулей позволит создавать и управлять зависимостями, которые специфичны для конкретных функций, обеспечивая их изоляцию и предотвращая утечки памяти.

2.5. Улучшение кэширования

Проблема: Текущая реализация кэширования содержит дублирующийся код и требует ручного управления, что увеличивает вероятность ошибок и усложняет поддержку.

Решение: * **Паттерн Repository с Mediator:** Использование паттерна Repository с Mediator (например, комбинация Room для локального хранения данных и кэша в памяти) позволит создать единый источник истины для данных и эффективно управлять их жизненным циклом. Репозиторий будет отвечать за выбор источника данных (кэш, база данных, сеть) и их синхронизацию. * **Аннотации для автоматического управления кэшем:** Если возможно, рассмотреть использование аннотаций или других механизмов для автоматизации управления кэшем (например, аннотации для кэширования результатов методов или инвалидации кэша при изменении данных). * **Kotlin Flow:** Использование Kotlin Flow для работы с данными позволит реализовать реактивное кэширование, где изменения в источнике данных автоматически отражаются в UI. Это упростит обработку асинхронных операций и управление состоянием данных.

Улучшения UI/UX

2.6. Компонентный подход

Проблема: Отсутствие четкой структуры для компонентов UI может привести к несогласованности дизайна, дублированию кода и усложнению разработки новых экранов.

Решение: * **Выделение компонентов по уровням:** Применение атомарного дизайна или аналогичного компонентного подхода позволит организовать UI-элементы по уровням абстракции: * **Atoms:** Базовые, неделимые элементы UI (например, кнопки, поля ввода, иконки, типографика). * **Molecules:** Композиции атомов, образующие функциональные, но все еще простые компоненты (например, формы поиска, карточки товаров). * **Organisms:** Более сложные композиции молекул и атомов, представляющие собой секции или блоки функциональности (например, шапки страниц, футеры, навигационные панели). * **Templates:** Общие шаблоны экранов, которые определяют расположение организмов и молекул, но не содержат конкретных данных. * **Pages:** Конкретные экраны приложения, которые являются экземплярами шаблонов с реальными данными.

2.7. Унификация визуального языка

Проблема: Разные части приложения могут использовать различные стили, цветовые схемы и подходы к UI, что создает несогласованный пользовательский опыт и затрудняет брендирование.

Решение: * **Создание Design System:** Разработка полноценной Design System, включающей набор компонентов, принципы типографики, цветовую палитру,

иконографию и рекомендации по использованию. Это обеспечит единообразие и ускорит разработку. * **Сториборд для документирования:** Создание сториборда (или аналогичного инструмента, например, Storybook для Jetpack Compose) для документирования всех UI-компонентов, их состояний и вариантов использования. Это облегчит совместную работу дизайнеров и разработчиков. * **Централизованные темы и стили:** Использование централизованных тем и стилей в Android (например, с помощью Material Design 3 и Jetpack Compose Theming) позволит легко управлять внешним видом всего приложения из одного места.

2.8. Улучшение адаптивности

Проблема: Приложение может быть недостаточно адаптивным для различных размеров экранов и ориентаций устройств, что снижает удобство использования на планшетах, складных устройствах и в ландшафтном режиме.

Решение: * **Поддержка планшетов:** Разработка многопанельного интерфейса для планшетов, где на одном экране могут отображаться несколько связанных блоков информации (например, список транзакций и детали выбранной транзакции). * **WindowSizeClass API:** Использование WindowSizeClass API (в Jetpack Compose) для адаптации UI к различным размерам экрана. Это позволяет создавать гибкие макеты, которые автоматически подстраиваются под доступное пространство. * **Landscape-режим:** Оптимизация интерфейса для ландшафтного режима, особенно для экранов, где это может быть полезно (например, для отображения графиков или больших таблиц данных).

2.9. Улучшение доступности

Проблема: Отсутствие явных мер по обеспечению доступности может сделать приложение неудобным или невозможным для использования людьми с ограниченными возможностями.

Решение: * **Семантические метки для Screen Readers:** Добавление семантических меток и описаний для всех UI-элементов, чтобы программы чтения с экрана могли корректно озвучивать содержимое для пользователей с нарушениями зрения. * **Контрастность цветов:** Обеспечение достаточного контраста между текстом и фоном, а также между различными UI-элементами, чтобы улучшить читаемость для пользователей с нарушениями зрения или дальтонизмом. * **Опция увеличенного шрифта:** Предоставление пользователям возможности увеличивать размер шрифта в приложении, чтобы адаптировать его под свои индивидуальные потребности. * **Поддержка клавиатуры:** Реализация полной поддержки навигации и взаимодействия с приложением с помощью

клавиатуры, что важно для пользователей, которые не могут использовать сенсорный экран или мышь.

Технические улучшения

2.10. Управление состоянием

Проблема: В проекте используется архитектура MVI (Model-View-Intent), но ее применение не всегда последовательно. Это может привести к усложнению логики, трудностям в отладке и тестировании, а также к потенциальным ошибкам в управлении состоянием UI.

Решение: * **Строгое следование принципам MVI:** Обеспечить строгое соблюдение принципов MVI, где каждый экран или компонент имеет четко определенные: `State` (неизменяемое состояние UI), `Intent` (действия пользователя или системы) и `View` (отображение состояния и отправка интентов). Это сделает поток данных однонаправленным и предсказуемым. * **Использование StateFlow/SharedFlow:** Для управления состоянием и событиями в MVI рекомендуется использовать Kotlin Flow, в частности `StateFlow` для представления состояния UI и `SharedFlow` для одноразовых событий (например, навигации, отображения Snackbar). * **Разделение логики:** Четко разделить логику представления (View), бизнес-логику (ViewModel/Presenter) и логику данных (Repository/Use Cases). ViewModel должна быть максимально "тупой" и просто передавать интенты в Use Cases и обновлять состояние на основе результатов. * **Тестирование:** Улучшить покрытие тестами для ViewModel и Use Cases, чтобы гарантировать корректное управление состоянием и обработку всех возможных сценариев.

Эти рекомендации, основанные на анализе кодовой базы и лучших практиках, помогут улучшить архитектуру приложения "Деньги под Контролем", сделать его более масштабируемым, поддерживаемым и удобным для дальнейшего развития.

3. Сравнительный анализ с конкурентами и рекомендации по функциональности

Для выработки рекомендаций по улучшению функциональности приложения "Деньги под Контролем" был проведен сравнительный анализ с рядом популярных приложений для ведения личного бюджета, представленных на рынке. В качестве источников информации использовались обзоры и рейтинги, в частности, статья "10 лучших приложений для учёта личных финансов в 2025 году" на Lifehacker.ru [1].

Обзор конкурентов

Ниже представлена сводная таблица ключевых особенностей и функциональных возможностей рассмотренных конкурентов:

Приложение	Автоматический импорт операций	Подкатегории	Бюджетирование	Финансовые цели	Со до
Дзен-мани	Да (подписка)	Да (подписка)	Да (подписка)	Да (подписка)	Да
Money Flow	Нет	Да (подписка)	Да (подписка)	Да (подписка)	Нет
CoinKeeper	Да (подписка Platinum)	Да	Да	Да (iOS)	Да под
Wallet	Да (подписка)	Да	Да	Да	Да (по
EasyFinance	Да (подписка)	Да	Да	Да	Да (по
Monefy	Нет	Да	Нет	Нет	Да
Кошелёк	Нет	Да	Да	Нет	Нет
Money Manager	Нет	Да	Да	Нет	Нет
Тяжеловато	Нет	Да	Нет	Нет	Нет
Money Lover	Нет	Да	Да	Да	Нет

Ключевые выводы из сравнительного анализа

1. **Автоматизация ввода данных:** Большинство ведущих приложений предлагают автоматический импорт операций из банков, что является значительным преимуществом для пользователей, так как избавляет от ручного ввода. "Дзен-мани" и CoinKeeper являются яркими примерами такой реализации.
2. **Детализация категоризации:** Подкатегории являются стандартной функцией для большинства конкурентов, позволяя пользователям более

точно классифицировать свои расходы и доходы. Это соответствует запросу пользователя Олега из RuStore.
3. **Бюджетирование и финансовые цели:** Функции бюджетирования и постановки финансовых целей широко распространены и являются ключевыми для пользователей, стремящихся к финансовой дисциплине.
4. **Совместный доступ:** Некоторые приложения, такие как "Дзен-мани" и Wallet, предлагают совместный доступ, что удобно для семейного или общего бюджета.
5. **Расширенная аналитика:** Конкуренты активно используют графики, диаграммы и даже ИИ-прогнозы для предоставления глубокой аналитики. CoinKeeper на iOS даже предлагает трекинг инвестиций.
6. **Дополнительные удобства:** Сканирование чеков по QR-коду (Дзен-мани, Money Manager), экспорт данных, веб-версии (EasyFinance) и синхронизация между устройствами являются востребованными функциями.
7. **Монетизация:** Многие приложения используют модель freemium, предлагая базовые функции бесплатно и расширенные возможности по подписке.

Рекомендации по улучшению функциональности "Деньги под Контролем"

На основе проведенного анализа, для приложения "Деньги под Контролем" можно предложить следующие функциональные улучшения:

1. **Автоматический импорт операций:** Это наиболее востребованная функция, которая значительно упростит жизнь пользователям. Рассмотреть возможность интеграции с российскими банками (через API или парсинг SMS/push-уведомлений). Важно продумать, как это будет сочетаться с текущей офлайн-моделью приложения, возможно, предложив это как опциональную функцию с явным согласием пользователя на обработку данных.
2. **Реализация подкатегорий:** Это прямое пожелание пользователя. Внедрение иерархической системы категорий (категория -> подкатегория) позволит пользователям более детально анализировать свои траты и доходы. Необходимо продумать удобный UI для их создания, редактирования и использования.
3. **Расширенное бюджетирование:** Помимо текущей возможности ставить цели, реализовать полноценную систему бюджетирования по категориям с отслеживанием лимитов и уведомлениями о превышении. Это поможет пользователям более эффективно контролировать свои расходы.
4. **Улучшенная аналитика и визуализация:** Хотя приложение уже предлагает графики, можно расширить их функциональность: добавить больше типов диаграмм (например, круговые диаграммы для распределения расходов по

категориям, гистограммы для сравнения периодов), возможность фильтрации и группировки данных по различным параметрам (период, категория, счет).
5. **Сканирование чеков по QR-коду:** Это очень удобная функция, которая значительно ускоряет ввод расходов. Интеграция с сервисами ФНС для распознавания чеков может быть сложной, но очень ценной функцией.
6. **Совместный доступ (опционально):** Если разработчик планирует расширять аудиторию, совместный доступ к бюджету может быть полезен для семей или небольших групп. Однако это потребует изменения офлайн-модели и внедрения синхронизации данных, что является серьезным архитектурным изменением.
7. **Инвестиции и долги:** Для более продвинутых пользователей можно добавить функционал по учету инвестиций (например, отслеживание стоимости активов) и долгов (как выданных, так и полученных).
8. **Напоминания о регулярных платежах:** Расширить функционал напоминаний, чтобы пользователи могли настраивать уведомления о предстоящих регулярных платежах (коммунальные услуги, кредиты, подписки).
9. **Экспорт данных:** Предоставить пользователям возможность экспортировать свои финансовые данные в различных форматах (CSV, Excel, PDF) для дальнейшего анализа или резервного копирования.
10. **Улучшение юзабилити ввода:** Автоматическое переключение клавиатуры на цифровую при вводе суммы, как было отмечено в отзыве, является небольшим, но важным улучшением пользовательского опыта.

Реализация этих функций позволит "Деньги под Контролем" конкурировать с лидерами рынка, предлагая пользователям более полный и удобный инструмент для управления личными финансами, сохраняя при этом свои ключевые преимущества, такие как простота и офлайн-режим (если это не противоречит новым функциям).

Заключение

Приложение "Деньги под Контролем" является перспективным инструментом для управления личными финансами, обладающим рядом сильных сторон, таких как простота использования, интуитивно понятный интерфейс и полная офлайн-функциональность. Однако, как показал проведенный анализ, существует значительный потенциал для его дальнейшего развития и улучшения, как с точки зрения функциональных возможностей, так и архитектуры кодовой базы.

Ключевые рекомендации по функциональности включают внедрение автоматического импорта операций из банков, реализацию детализированных подкатегорий, расширение возможностей бюджетирования и аналитики, а также добавление таких удобных функций, как сканирование чеков и расширенные напоминания. Эти улучшения позволят приложению лучше соответствовать ожиданиям современных пользователей и успешно конкурировать с ведущими решениями на рынке.

В части кодовой базы, основные рекомендации сосредоточены на переходе к модульной архитектуре, централизации навигации, унификации репозиториев, улучшении системы Dependency Injection и оптимизации кэширования. Внедрение этих архитектурных изменений значительно повысит масштабируемость, поддерживаемость и общую надежность проекта, что критически важно для долгосрочного развития и добавления новых сложных функций.

Реализация предложенных улучшений позволит "Деньги под Контролем" стать еще более мощным и востребованным инструментом для управления личными финансами, сохраняя при этом свои уникальные преимущества и привлекая более широкую аудиторию пользователей.

Источники

[1] 10 лучших приложений для учёта личных финансов в 2025 году — Лайфхакер. URL: https://lifehacker.ru/10-money-management-apps/