Планируется разработка полноценного программно-аппаратного комплекса, включающего в себя аппаратуру для снятия и беспроводной передачи сигналов с вибрационных датчиков на сервер, а затем анализа и обработки данных с помощью данного программного средства.

Применение данной программы позволит пользователям уменьшить эксплуатационные издержки, повысит гибкость использования и снизит потребности в специализированных аппаратно-программных комплексах.

Список использованных источников:

1. Sheffer C. Machinery Vibration Analysis & Predictive Maintenance / C. Sheffer, P. GirdHar. – Newnes, 2004

VIPER АРХИТЕКТУРА В МОБИЛЬНОЙ РАЗРАБОТКЕ

Белорусский государственный университет информатики а радиоэлектроники
г. Минск, Республика Беларусь

Антоненко Д.А.

Куликов С.С. – к-т. наук, доцент.

В последнее время наблюдается тенденция появления новых архитектур построения мобильных приложений. Это вызвано тем, что не все имеющиеся архитектуры позволяют достичь необходимой гибкости построения приложений, необходимого уровня тестируемости, эффективного распределения функциональности между компонентами. В результате часто приходится создавать свои собственные архитектурные решения, что занимает достаточно много времени. В связи с этим начался поиск некоторого универсального решения, решающего большинство проблем.

Исторически, базовой архитектурой при построении мобильных приложений являлась архитектура Модель-Вид-Контроллер (англ. - MVC). Со временем появилось много её вариаций, среди которых наиболее известными являются: Модель-Вид-Презентер (англ. MVP), Модель-Вид-Вид-Модель (англ. MVVM). Они устраняют определённые недостатки архитектуры Модель-Вид-Контроллер, но не решают всех её проблем и недостатков.

VIPER – новая архитектура построения приложений (в частности – мобильных), базирующаяся на принципах «чистой архитектуры» (англ. clean architecture). Она призвана решить большинство (если не все) проблем MV(X)-подобных архитектур и заменить их полностью. Ниже приведена схема различных компонентов VIPER-архитектуры и связей между ними.

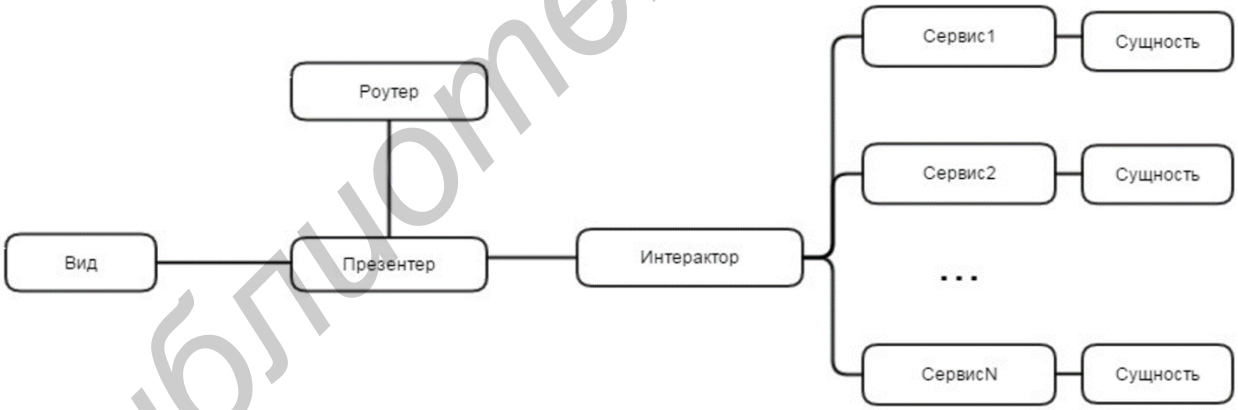

Рисунок 1 – VIPER архитектура, схема различных компонентов и связей между ними.

Основными понятиями, используемыми в данной архитектуре, являются:

- Вид – отображает, что сообщил Презентер и передаёт ввод данных пользователем назад Презентеру.
- Интерактор – содержит бизнес-логику, связанную с данными (сущность) или сетевыми операциями, как например создание новых экземпляров сущностей или загрузка их с сервера.
- Презентер – отвечает за пользовательский интерфейс, связанный с (но независимый от графических библиотек конкретной платформы) бизнес-логикой, вызывает методы на Интеракторе.
- Сущности – простые объекты данных, но не уровень доступа к данным, так как ответственность за это лежит на Интеракторе.
- Роутер – отвечает за непосредственно переходит между VIPER модулями.
- Сервис – единица Интерактора, выполняющая определённый вид работ.

Исходя из рассмотренной архитектуры, можно сделать следующий выводы:

а) Распределение обязанностей – несомненно, VIPER является чемпионом в распределении обязанностей между своими компонентами.

б) Тестируемость – обеспечивается отличная тестируемость благодаря имплементации принципа единственной обязанности.

в) Удобство использования – высокая стоимость «ремонтопригодности». Необходимо написать большое количество интерфейсов для классов с очень малыми обязанностями.

Таким образом, VIPER помогает нам быть более точными, что касается разделения проблем, разделяя большое количество кода одного класса на несколько меньших классов. За счёт поддержания единственной ответственности в каждом классе это упростит разработку классов, используя разработку через тестирование (англ. TDD), которое позволяет нам более быстро реагировать на изменяющиеся требования и создавать лучшее программное обеспечение. В тоже время использовать данную архитектуру неудобно при отсутствии большого количества бизнес-логики на экран. Возникают проблемы при необходимости использования повторно одних и тех же экранов при разных сценариях, так как необходимо хранить зависимости между модулями, который сейчас активен и который будет показан следующим. Сложно «наследоваться» от других модулей приложения.

БЫСТРОЕ ОПРЕДЕЛЕНИЕ ПРИНАДЛЕЖНОСТИ ПОЛЬЗОВАТЕЛЯ К СЕГМЕНТУ ПО IP-АДРЕСУ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Базаревский В. Э.

Брацевич П. Ю. – кандидат технических наук

Рассматриваются алгоритмы быстрого определения принадлежности пользователя к сегменту IP-адресов, особенности хранения и организации сегментов для ускорения работы. Рассмотрена структура данных дерево отрезков, особенности работы с ним и построение.

В современных интернет-технологиях одним из основных способов монетизации приложений является продажа интернет-рекламы, тем самым делая её одним из важнейших элементов экосистемы сети интернет. Вместе с тем следует отметить следующий тренд вместо показа рекламы на определенных сайтах, становится возможным эффективно показывать рекламу её целевой аудитории за счет четкого таргетирования по интересам. Одним из наиболее распространенных методов таргетирования рекламы, является сегментации пользователей на основе их интересов, географического местоположения, используемых устройств, гендерных и иных признаков.

Среди различных средств сегментирования пользователей следует выделить метод сегментирования пользователей по IP-адресам. Так как фактически имеет место географическая сегментация пользователей, то на основе дополнительных сведений об отличительных особенностях абонентов конкретных IP-адресов результирующие сегменты несут в себе более сложную информацию (например, интересы, возраст, пол, профессию и т.д. пользователей). В качестве примера, всех абонентов, выходящих с IP-адресов, принадлежащих университету, можно отнести к сотрудникам университета или студентам.

Зачастую, сегментация по IP является не тривиальной задачей, так как пространство IP-адресов сильно разрежено, представляет собой большое количество интервалов разной длинны, при этом сильно пересекающихся друг с другом.

Ввиду описанных выше требований, необходима реализация эффективной структуры данных для определения сегмента по отдельно взятому IP-адресу, оптимальной с точки зрения поиска, масштабируемости и потребления ресурсов. Для решения этой задачи был предложен ряд решений, которые помогли оптимизировать процесс определения принадлежности пользователя к тому или иному сегменту.

Сегмент представляется в виде областей IP-адресов с начальным и конечным IP-адресами. IP-адрес в 4-ой версии представляет собой 32-битовое число. Удобной формой записи IP-адреса (IPv4) является запись в виде четырёх десятичных чисел значением от 0 до 255, разделённых точками. Для того чтобы оптимизировать операцию сравнения адресов и ускорить проверку на вхождение адреса в сегмент, IP-адрес из строкового представления был конвертирован к типу Long. В таком численном виде операция сравнения занимает меньше процессорного времени.

Кроме задачи сравнения адресов существует задача организации сегментов в структуру, оптимизирующую операцию поиска и определения принадлежности IP-адреса к конкретному сегменту. Самый простой способ решать представленную задачу, это завести линейный массив. При такой реализации время нахождения ответа на запрос имеет порядок O(n). Изменение же значения какого-либо сегмента требует O(1) времени. Операция изменения значения для выбора архитектуры представления сегментов в данном случае является не определяющей, так как изменение сегментов происходит редко, а операция поиска в свою очередь выполняется при каждом обращении клиента к сервису. Альтернативной структурой организации данных о сегментах пользователей для оперативного поиска является дерево отрезков.

Дерево отрезков представляет собой дерево, листьями которого являются элементы исходного массива. Остальные вершины являются результатом какой-либо операции над детьми. В нашем случае удобнее всего выбрать операцию объединения сегментов. Таким образом, корень хранит результат