ЭТАП 2. АРХИТЕКТУРА ПРОГРАММНОГО РЕШЕНИЯ

Заказчик:	НИУ ВШЭ, Международная лаборатория интеллектуальных систем и структурного анализа, Паринов Андрей Андреевич
Название проекта:	Клиент-серверное iOS приложение Ассистента Студента
Исполнители	Зубарева Наталия Дмитриевна, Мостачев Андрей Олегович, Поволоцкий Виктор Александрович, Сальникова Алиса Дмитриевна

1) Описание проекта

Цель проекта

Целью проекта является разработка клиент-серверного мобильного приложения для IOS - цифрового ассистента студентов, предназначенного для решения проблемы поиска студентами проектов для учебных курсов.

Назначение разработки

Приложение используется в сфере образования, конкретно проектной деятельности и должно предоставлять агрегированные описания проектов, которые студенты смогут просматривать и выбирать в соответствии со своими интересами с помощью системы рекомендаций.

Основная решаемая проблема - ознакомление с проектами и выбор проекта, в том числе предложение своей темы проекта и возможная регистрация сделанного выбора в информационной системе ВШЭ. Приложение должно позволить пользователю просмотреть список предложенных проектов, максимально полезную информацию о них, также увидеть наиболее актуальные из них, предложенные системой рекомендаций, выбрать подходящий, предложить свой.

Ключевые требования к разработке

Приложение должно позволять студентам

- авторизоваться в приложении с помощью учетных данных ЕЛК ВШЭ;
- заполнять данные в учетной записи, в том числе о проектных интересах;
- просматривать все предлагаемые проекты и подробную информацию о них;
- выбирать проект для участия и отменять этот выбор;
- получать рекомендации проектов от системы;

2) Модель жизненного цикла

Для реализации проекта выбрана **итерационная модель** жизненного цикла.

Главным аргументом в пользу выбора этой модели является гибкость, адаптивность этой модели разработки и тесный контакт с заказчиком в процессе. Так как от разрабатываемого проекта требуется детальное соответствие требованиям заказчика, итеративная модель с регулярными встречами для подтверждения требований и приемки выполненных этапов очень подходит для нашей команды. Кроме этого, так как приложение разрабатывается для использования в университетской среде, где часто происходят изменения ландшафта систем и приложений, имеет смысл сохранение максимальной гибкости и выделение требований

для каждой подсистемы приложения максимально близко к моменту ее создания для сохранения актуальности разработки.

В силу желаемой гибкости разработки для нас не имело бы смысла выбирать линейные модели жизненного цикла, например водопадную, так как выделение всех требований к программе в самом начале и последующая разработка без общения с заказчиком сказалась бы плачевно на соответствии программы актуальным запросам заказчика.

Из более гибких моделей, потенциально была рассмотрена инкрементная, однако мы пришли к выводу, что версионность, реализуемая при инкрементной модели, удовлетворяет заказчика в большей степени - так, на одном из начальных этапов разработки уже можно получить минимально функционирующее приложение, и уточнить и усовершенствовать его функциональность в дальнейшем, в то время как инкрементная модель грозила бы невозможностью в частности протестировать систему целиком до самого завершения проекта.

В рамках организации работы нашей команды выбор этой модели жизненного цикла означает, что работа организована в формате спринтов, которые предполагают регулярные встречи с руководителем для оценки результатов предыдущего спринта, обозначения задач на текущий спринт и обсуждения другой важной информации по проекту, как например внесение изменений по результатам работы.

3) Design Patterns

В нашем проекте особенно активно используются следующие паттерны:

Паттерн	Общее описание паттерна	Назначение и решаемая задача	Последствия применения
Builder	Паттерн, предоставляющий способ создания составного объекта с возможным изменением его содержания.	Создание экранов View-части в приложении, генерация страниц, для этого данный паттерн используется в соответствующих классах ViewModel.	Позволяет варьировать содержимое страниц, изолирует создание интерфейса от данных.
Factory Method	Паттерн, дающий дочерним классам интерфейс для создания объектов другого класса с выбором создаваемого класса наследниками в момент создания.	Создание элементов разметки на экранах приложения, для этого находится в экземплярах View, соответствующих страницам.	Объекты интерфейса создаются централизованно вне зависимости от их конкретного класса.
Singleton	Паттерн, гарантирующий существование единственного экземпляра класса в приложении и	Существование экземпляров классов модели, сервисов - все они создаются в виде синглтонов и разделяются	Централизует хранение информации, позволяет избежать конфликтов поддержания

	обеспечивающий доступ к этому экземпляру.	всем приложением.	актуальности данных.
Observer	Паттерн, обеспечивающий получение классом оповещений об изменении состояния других объектов, наблюдения за ними.	Наблюдение за изменениями данных модели, обеспечение поддержания актуальности отображения, для этого они создаются как наблюдатели модели данных.	Автоматически обновляет отображение информации при ее изменении без создания копий данных.

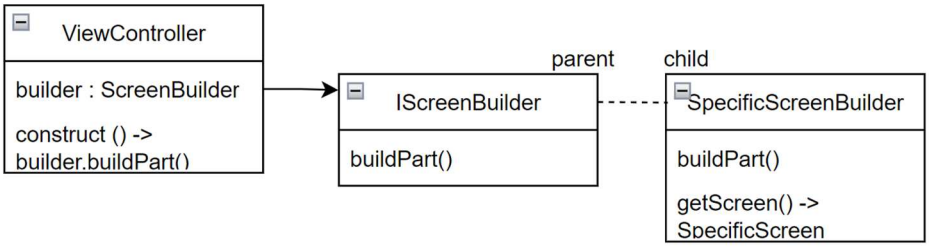

Рис. 1 - структура паттерна Builder

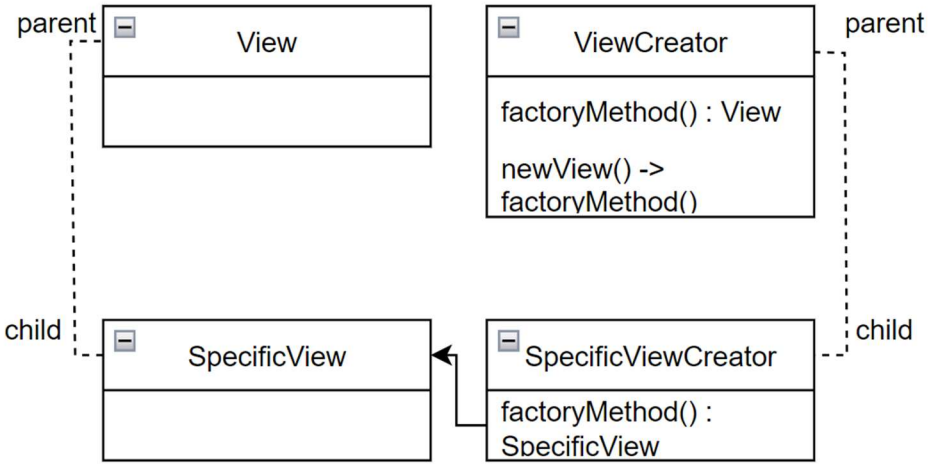

Рис. 2 - структура паттерна FactoryMethod

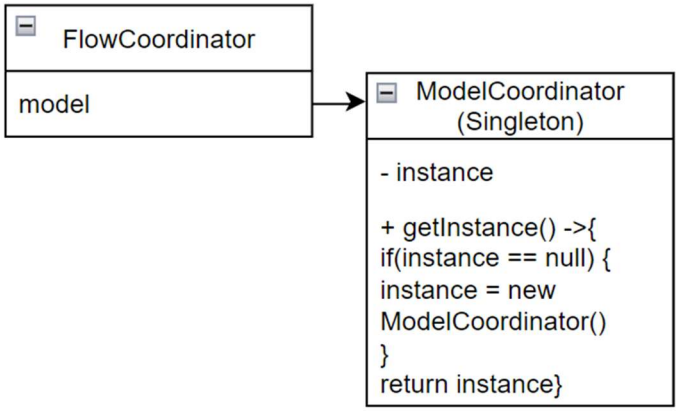

Рис. 3 - структура паттерна Singleton

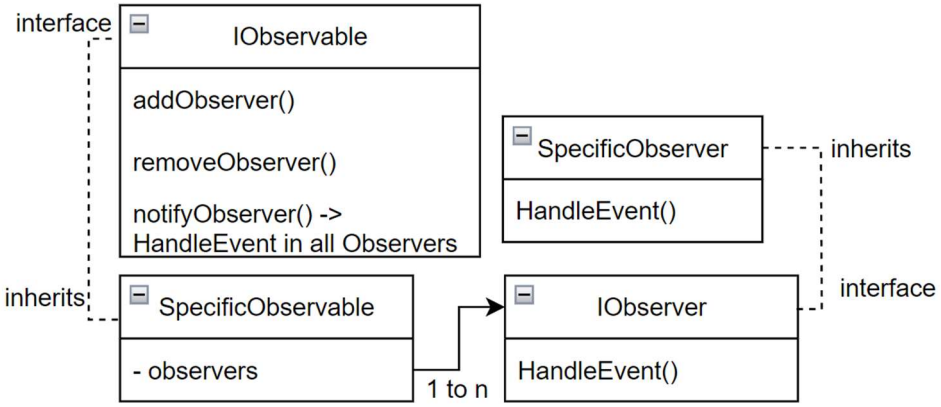

Рис. 4 - структура паттерна Observer

Глобально использование паттернов направлено на создание более надежного, грамотно структурированного кода, уменьшения числа зависимостей между разными структурами и улучшения понимания кода у разработчиков.

4) Тип архитектуры приложения

Для реализации приложения выбрана клиент-серверная структура с организацией клиентской части по принципу **Model-View-ViewModel + coordinator**.

Выбор клиент-серверной структуры обоснован необходимостью хранить большие объемы данных (о пользователях и проектах), а также проводить определенные вычисления (в частности работу алгоритма по подбору проектов-рекомендаций). Таким образом, чтобы не нагружать клиентскую часть приложения подобными работами, они вынесены на сервер и предоставляются приложению через сервисы.

Организация MVVM + coordinator была выбрана для повышения качества создаваемого приложения путем обеспечения принципа одной ответственности блока кода - в этой парадигме разделены данные модели, интерфейс, создание интерфейса и прослойка, координирующая данные и интерфейс.

Блок Model отвечает за модель данных - там находятся классы пользователя, проекта и т.д., то есть данные, отображаемые приложением.

View содержит классы, отвечающие непосредственно за интерфейс приложения - там находится разметка страниц, элементы для отображения информации и взаимодействия пользователя с программой.

ViewModel - это блок, отвечающий за связь между данными из Model и их отображениями во View. Он содержит классы, соответствующие каждому классу View и настраивающие события, изменения, связь данных между интерфейсом и моделью. Именно из ViewModel через сервисы осуществляется общение с серверной частью приложения.

Coordinator - это блок высшей иерархии, классы, находящиеся там, создают экземпляры View и ViewModel для каждого необходимого отображения. Он помогает избежать генерации этих объектов и перемещения между ними на их уровне (в классах этих же блоков, созданных ранее), вместо этого вынося эти функции выше.

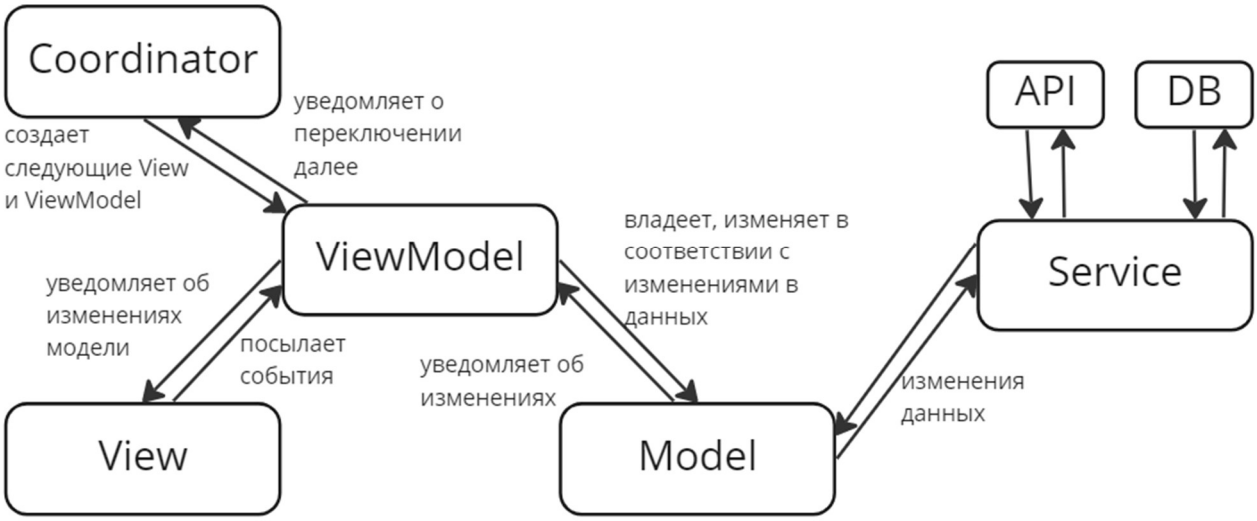

Рис. 5 - общая архитектура приложения

Заказчик Личная подпись /Паринов А.А

Ответственный по проекту Личная подпись /Зубарева Н.Д.

Дата: 20.12.2022