УДК 004.4

Проектирование архитектуры сервиса на основе подходов Clean Architecture и Domain Driven Design

Згода И.К., Половикова О.Н.
Алтайский государственный университет, г. Барнаул
playa-ogg@yandex.ru, ponOlgap@gmail.com

Аннотация

В данной статье рассматривается проектирование архитектуры веб-сервиса для мониторинга результатов проведения занятий с VR-тренажером на основе подходов Clean Architecture и Domain Driven Design, описываются ключевые аспекты проектирования, включая структуру проекта, разделение ответственностей между слоями приложения, моделирование предметной области и применение паттернов проектирования. Особое внимание уделяется основным компонентам архитектуры и принципам их взаимодействия. Также в статье обсуждаются перспективы дальнейшего развития разрабатываемого сервиса.

Ключевые слова: Clean Architecture, Domain Driven Design, web-сервис, VR-тренажер, dependency rule, entity, aggregate, value object, repository, слои приложения.

1. Введение

В настоящее время технологии виртуальной реальности активно внедряются в различные сферы образования, включая военное дело. Одним из таких инновационных решений является VR-тренажер по сборке и разборке огнестрельного оружия, который используется в рамках образовательного модуля “Основы военного дела” и позволяет студентам практиковать сложные и потенциально опасные процедуры в максимально реалистичных условиях, обеспечивая безопасную и контролируемую среду для обучения. Тренажер использует высококачественные 3D-модели оружия и включает в себя несколько режимов, таких как:

1) режим обучения, в котором предусмотрена система подсказок и инструкций, позволяющая пользователю освоить последовательность составных частей оружия при сборке,

2) режим тренировки, который позволяет практиковать полученные навыки без ограничений по времени,

3) режим экзамена, в котором учитываются все ошибки студента и время, затраченное на выполнение задания.

Кроме этого, тренажер оснащен виртуальной клавиатурой, предназначенной для аутентификации пользователей корпоративной сети и технологией сбора json файлов, состоящих из результатов экзамена, представляющих собой действия студентов и временные метки выполнения задач. Таким образом, VR-тренажер обладает достаточным функционалом для освоения практических навыков сборки и разборки огнестрельного оружия, однако для полноценного использования тренажера в образовательном процессе необходимо обеспечить мониторинг результатов проведения занятий, исходя из этого, возникает необходимость в разработке сервиса, который бы позволил:

Осуществлять аутентификацию пользователей корпоративной сети университета через LDAP (Lightweight Directory Access Protocol) службу.

4) Собирать данные о действиях студентов во время занятий с VR-тренажером,

5) Хранить статистику по пользователям,

6) Обеспечивать просмотр и выборку данных для анализа успеваемости.

Подобный сервис позволит значительно повысить эффективность обучения, предоставив преподавателям возможность анализировать и оценивать этапы работы студентов с тренажером, а студентам возможность отслеживать итоги своих занятий.

В условиях быстрого развития технологий и изменения требований к образовательным программам, базовые архитектуры, такие как монолит, являются неэффективными, поскольку повышают связность (coupling) системы и понижают когезию (cohesion). Это затруднит разработку, тестирование и поддержку, так как внесение изменений в один программный модуль влечет за собой изменения во всех частях системы, с которыми он связан, а добавление нового функционала может потребовать переписать код части проекта. Чтобы избежать этих проблем, важно спроектировать адаптивную и устойчивую архитектуру, наиболее подходящими подходами являются Clean Architecture и Domain Driven Design (DDD), они дополняют друг друга, создавая синергетический эффект. Их совместное использование предоставляет значительные преимущества для разработки сложных и масштабируемых приложений, а именно позволяет эффективно разделять ответственность между различными слоями приложения, что способствует лучшему пониманию и структурированию бизнес-логики. Clean Architecture изолирует предметную область от технических деталей, облегчая тестирование и упрощая замену компонентов инфраструктуры. В свою очередь, DDD фокусируется на моделировании домена, что делает бизнес-логику более предсказуемой и легко тестируемой. В результате, совместное использование этих подходов обеспечивает высокую гибкость, масштабируемость и улучшенную тестируемость, что существенно повышает качество и устойчивость разрабатываемого сервиса.

2. Проектирование сервиса

Clean Architecture, предложенная Робертом Мартином, фокусируется на разделении ответственностей и независимости компонентов, что позволяет минимизировать зависимости между различными частями системы. Основные принципы данного подхода включают независимость от фреймворков, баз данных, пользовательских интерфейсов и внешних агентов. Это достигается разделением приложения на слои и следование правилу зависимостей (dependency rule), это правила, при которых внутренние слои не должны зависеть от внешних. Схема слоев приложения представлена на рисунке 1.

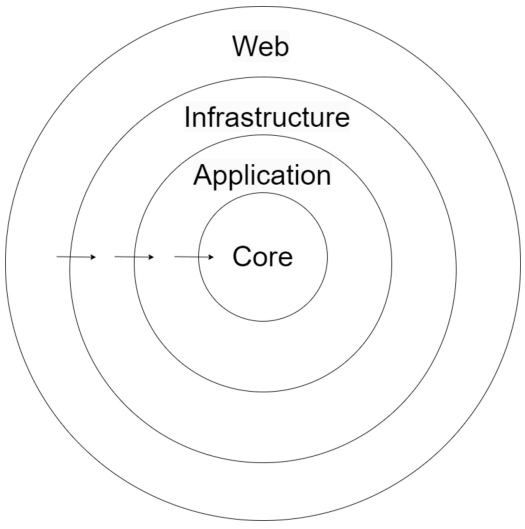

Рисунок 1. Схема слоев приложения

Стрелками на схеме изображены направления зависимостей, согласно правилу зависимостей, они могут указывать только во внутрь круга. Чтобы зависимость была направлена

в сторону, обратную потоку данных, и внутренние слои могли обращаться к внешним, не нарушая dependency rule, применяется принцип инверсии зависимостей. Например, из слоя Application нужно обратиться к слою Infrastructure, но напрямую это сделать нельзя, так как внутренние круги не должны знать о внешних. При таком подходе в слое Application необходимо определить интерфейс, а слой Infrastructure предоставит реализацию и правило не будет нарушено [1].

Поход DDD, разработанный Эриком Эвансом, акцентирует внимание на моделировании предметной области, разделении моделей на контексты ограниченности (bounded context) и использование таких концепций как сущности (entities), агрегаты (aggregates), объекты значений (value objects), события домена (domain events) и хранилища (repositories). Применяя этот подход к рассматриваемой предметной области, получаем схему домена, представленную на рисунке 2.

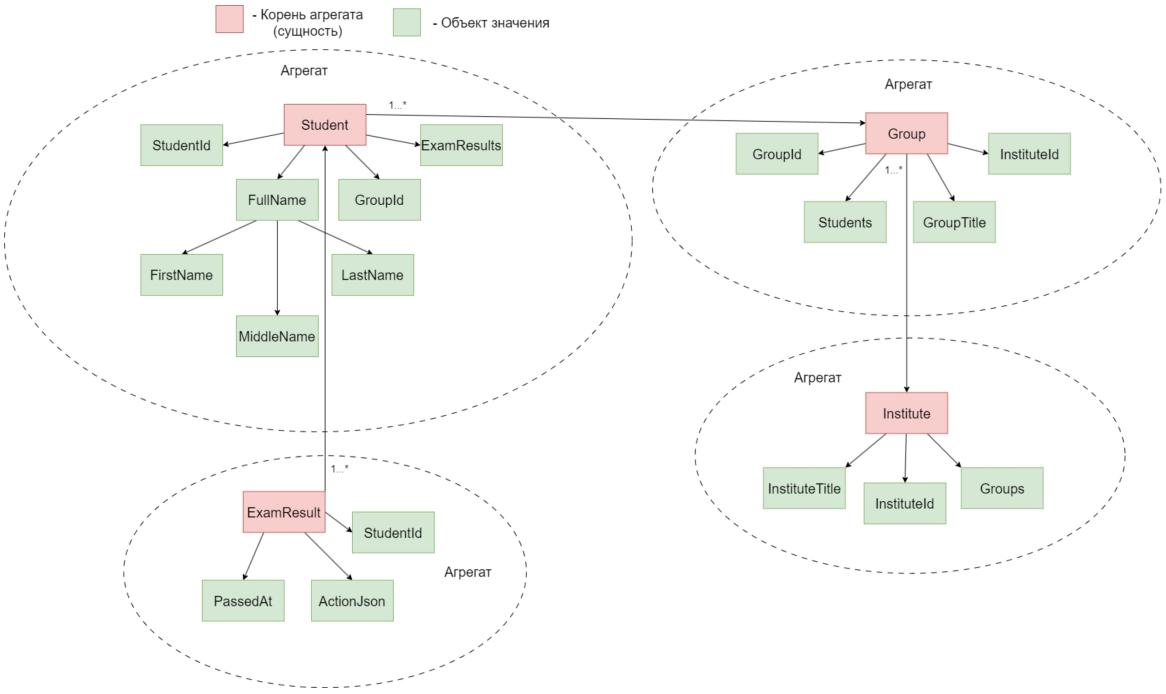

Рисунок 2. Домен

Агрегатом называется кластер из сущностей и объектов значений. У каждого агрегата есть глобально уникальный идентификатор, корень, представляющий собой сущность, и граница, внутри которой должны выполняться бизнес-инварианты. Любое обращение к агрегату должно происходить через его корень и обязательно по идентификатору, а не по ссылке. Сущность представляет собой объект, который имеет уникальную идентичность, отличающую ее от других сущностей независимо от атрибутов, сущности также содержат в себе бизнес-логику, управляющую ее состоянием, это явление называется богатой моделью (rich model). Объектом значения является неизменяемый тип данных, который не имеет уникальной идентичности, а эквивалентность определяется сравнением атрибутов, инкапсулированных внутри модели [2]. Все агрегаты, представленные на рисунке 2, попадают в один ограниченный контекст, так как включены в одну предметную область. Также стоит отметить, что разделение сущностей в рассматриваемой предметной области на отдельные агрегаты позволяет системе быть более гибкой, поскольку каждый агрегат имеет свою ответственность, бизнес-правила и инварианты. Например, результат экзамена может включать проверку корректности json файлов, а студент иметь ограничения на количество экзаменов, которые он может пройти за определенный период. Таким образом, каждый агрегат обеспечивает консистентность своих данных, которые могут изменять-

ся не затрагивая другие агрегаты. Например, результат экзамена может быть проверен и подтвержден независимо от студента. Наличие отдельного интерфейса и репозитория для каждой сущности подчеркивает их независимость. Интерфейсы репозиториев обеспечивают абстракцию доступа к данным и позволяют отделить бизнес-логику от деталей реализации хранения данных. Структура проекта Core, который определяет домен, представлена на рисунке 3.

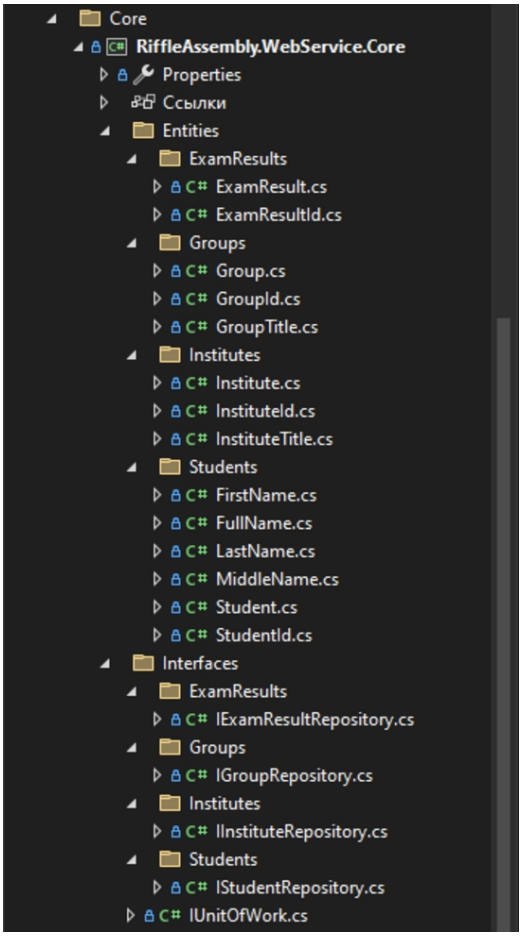

Рисунок 3. Core

Интерфейс IUnitOfWork представляет собой абстракцию для управления транзакциями и обеспечивает атомарность операций, выполняемых над несколькими репозиториями. Например, после успешной аутентификации система получает ФИО, институт и группу студента, если института и группы еще нет в базе данных, то все 3 сущности должны быть записаны в нее в рамках одной транзакции. Реализации репозиториев находятся в слое Infrastructure, за связь Core и Infrastructure отвечает слой Application. Структура проекта Application представлена на рисунке 4.

В слое Application используется паттерн проектирования CQRS (Command Query Responsibility Segregation), который разделяет операции чтения и записи данных на две отдельные ветки. Команды представляют собой операции, которые изменяют состояние системы и являются неизменяемым типом данных. Обработчики команд выполняют логику изменения состояния системы. В контексте web-сервиса мониторинга результатов проведения занятий с VR-тренажером состояние системы будут изменять только команды на запись, так как данные сервиса, кроме результатов экзамена, является отражением данных из LDAP. Запросами называются операции получения данных, они не изменяют состояние системы. Применение данного подхода позволяет оптимизировать каждую модель для своих задач, масштабировать систему горизонтально и улучшить произво-

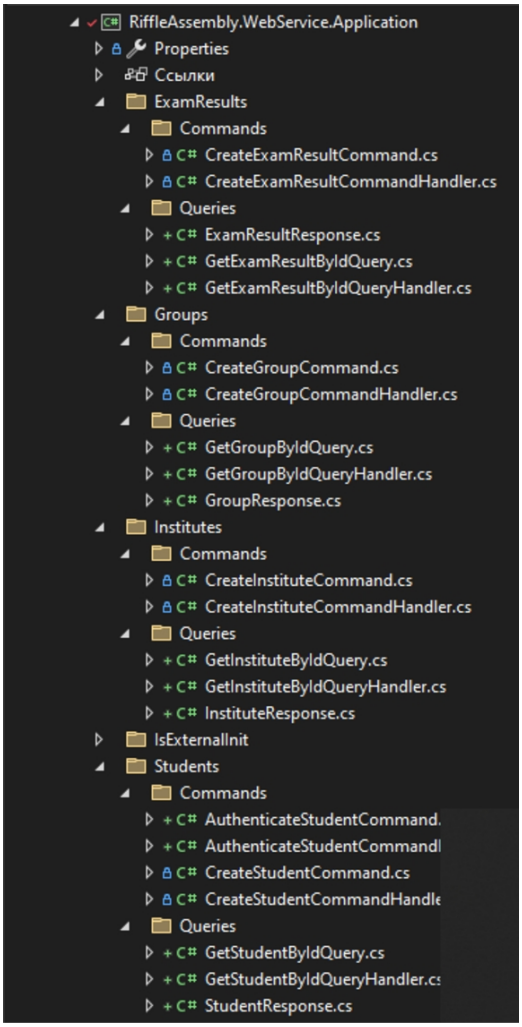

Рисунок 4. Application

дительность системы, например за счет денормализации модели данных для ускорения операции чтения и асинхронной обработки операций. Слой Application взаимодействует со слоем Infrastructure, для выполнения операций, связанных с доступом к данным, внешним сервисам и другим инфраструктурным компонентам.

Слой Infrastructure содержит в себе реализации репозиториев и UnitOfWork (инструмент для управления транзакциями), конфигурации для ORM (Object Relational Mapping), миграции и контекст базы данных, сервисы и адаптеры. Слой Infrastructure взаимодействует со слоем Web через интерфейсы и сервисы, определенные в слое Application и Domain. Взаимодействие между этими слоями происходит через зависимости, которые инжектируются в контроллеры и другие компоненты слоя Web. Это взаимодействие обеспечивает четкое разделение ответственности и позволяет слою Web сосредоточиться на обработке HTTP-запросов.

Слой Web отвечает за взаимодействие с пользователями и внешними системами через веб-интерфейсы, включая VR-тренажер и LDAP службу. Этот слой содержит компоненты, необходимые для обработки HTTP-запросов, представления данных и взаимодействия с другими слоями приложения. Данный слой включает в себя контроллеры, маршруты, контейнер внедрения зависимостей (DI Container), строку подключения к базе данных и другие настройки.

Совместное использование кода несколькими слоями достигается с помощью использования паттерна Shared Kernel, это отдельная библиотека, ссылка на которую имеется у Core и Application. Shared Kernel содержит в себе реализацию паттерна Result, который

улучшает обработку результатов операций в контексте асинхронного программирования. Этот паттерн помогает ясно и четко представлять успешные и неуспешные результаты операций, а также упрощает обработку ошибок и исключений. Основная идея паттерна Result заключается в том, что результат операции (будь то успешный или неуспешный) представляется в виде объекта, который содержит информацию о состоянии операции. Этот объект может содержать как успешный результат, так и информацию об ошибке, если операция завершилась неудачей. Result используется для валидации моделей в слое Core и Application. Структура проекта SharedKernel представлена на рисунке 5.

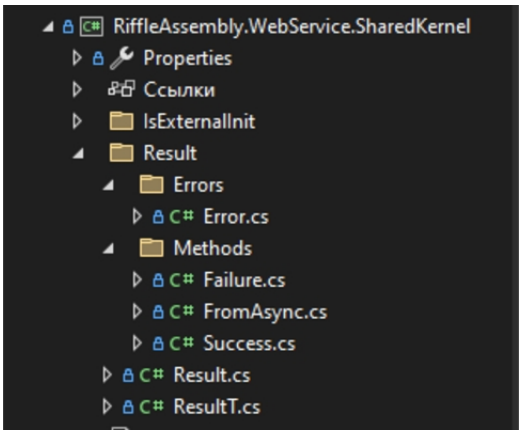

Рисунок 5. Shared Kernel

3. Итоги

На данный момент реализованы:

- скрипт для аутентификации пользователей корпоративной сети через LDAP службу,
- механизм сбора json файлов с результатами экзамена в VR-тренажере,
- слой домена (Core),
- слой бизнес-логики приложения (Application),
- общее ядро (Shared Kernel).

Необходимо реализовать:

- слой Infrastructure,
- слой Web.

Идея применения методов машинного обучения для автоматизации оценивания результатов экзамена является перспективной и может значительно улучшить процесс обучения и оценки.

Список литературы

1. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. — СПб. : Питер, 2018. — 352 с.
2. Вернон В. Реализация методов предметно-ориентированного проектирования. — М. : Вильямс, 2019. — 688 с.