**Практическая работа №3.**

**Теоретическое введение.**

**Чистая архитектура**

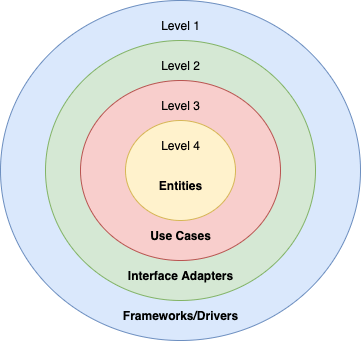
Когда мы разрабатываем долгосрочные системы, мы должны ожидать изменяемой среды.

В целом, наши функциональные требования, фреймворки, устройства ввода-вывода и даже дизайн нашего кода могут измениться по разным причинам. Имея это в виду, чистая архитектура является ориентиром для кода с высокой степенью поддержки, учитывая все окружающие нас неопределенности.

В этой статье мы создадим пример API для регистрации пользователей, следуя [чистой архитектуре Роберта К. Мартина](https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html). Мы будем использовать его оригинальные слои – сущности, варианты использования, интерфейсные адаптеры и фреймворки / драйверы.

Чистая архитектура компилирует множество дизайнов кода и принципов, таких как [SOLID](https://www.baeldung.com/solid-principles), [стабильные абстракции](https://wiki.c2.com/?StableAbstractionsPrinciple) и другие. Но основная идея заключается в том, чтобы разделить систему на уровни, основанные на ценности для бизнеса. Следовательно, самый высокий уровень имеет бизнес-правила, причем каждый нижний уровень приближается к устройствам ввода-вывода.

Кроме того, мы можем переводить уровни в слои. В данном случае все наоборот. Внутренний слой соответствует самому высокому уровню и так далее:

Рисунок 1 — слои в чистой архитектуре

Помня об этом, мы можем иметь столько уровней, сколько требуется нашему бизнесу. Но всегда учитывайте правило зависимости – более высокий уровень никогда не должен зависеть от более низкого.

**MVC**

MVC — это не паттерн проектирования. MVC — это именно набор архитектурных идей и принципов для построения сложных систем с пользовательским интерфейсом. Но для удобства, чтобы каждый раз не повторять: “Набор архитектурных идей…”, мы будем называть MVC паттерном.

Начнем с простого. Что же скрывается за словами Model-View-Controller?

При разработке систем с пользовательским интерфейсом, следуя паттерну MVC нужно разделять систему на три составные части. Их, в свою очередь, можно называть модулями или компонентами. Говори как хочешь, но дели на три. У каждой составной компоненты будет свое предназначение.

Model. Первая компонента/модуль — так называемая модель. Она содержит всю бизнес-логику приложения.

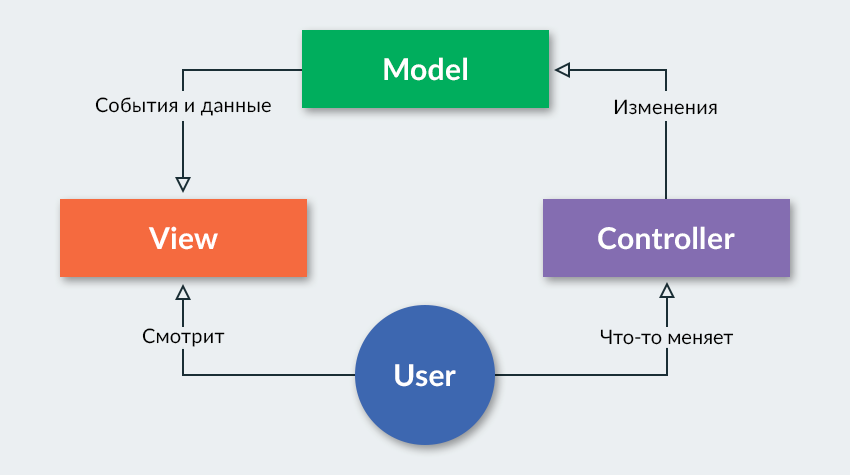
View. Вторая часть системы — вид. Данный модуль отвечает за отображение данных пользователю. Все, что видит пользователь, генерируется видом.

Controller. Третьим звеном данной цепи является контроллер. В нем хранится код, который отвечает за обработку действий пользователя (любое действие пользователя в системе обрабатывается в контроллере). Модель — самая независимая часть системы. Настолько независимая, что она не должна ничего знать о модулях Вид и Контроллер. Модель настолько независима, что ее разработчики могут практически ничего не знать о Виде и Контроллере.

Основное предназначение Вида — предоставлять информацию из Модели в удобном для восприятия пользователя формате. Основное ограничение Вида — он никак не должен изменять модель.

Основное предназначение Контроллера — обрабатывать действия пользователя. Именно через Контроллер пользователь вносит изменения в модель. Точнее в данные, которые хранятся в модели.

Приведем еще раз схему, которую тебе уже показывали на лекции:

Рисунок 2 — паттерн MVC

Из всего этого можно сделать вполне логичный вывод. Сложную систему нужно разбивать на модули.

**@Service**

Эта аннотации, которая выделяет слой бизнес логики — сервисы. Проект должен содержать такие слои:

1. Controller – слой логики, в которой реализована логика принятия запросов и выдача ответов на них.
2. Service – слой логики, в котором реализована абсолютно вся бизнес логика приложения.
3. Repositories – слой, в котором происходит связь с СУБД.
4. Models – слой, в котором реализованы описания сущностей с модели.

Как видно из списка, аннотация @Service создает слой для реализации бизнес логики. Классы, на которых весит эта аннотация, автоматически становятся bean’ами.

**Полезные ссылки**

1. Чистая архитектура — https://www.baeldung.com/spring-boot-clean-architecture
2. MVC – https://javarush.com/groups/posts/2536-chastjh-7-znakomstvo-s-patternom-mvc-model-view-controller
3. Аннотация Service – https://www.baeldung.com/spring-service-annotation-placement

**Задание**

Студенту предлагается реализовать бизнес логику ко второй практике по Java. Нужно добавить к существующему приложению реализацию «корзины».

Пользователь должен иметь такие возможности:

1. Добавить товар в корзину.
2. Удалить товар из корзины.
3. Изменить количество товара в корзине.
4. Посмотреть всю корзину.
5. Оформить заказ и очистить корзину.

При добавлении товара в корзину должна быть проверка, что товар есть в наличии. Так же обработать случай, что товар добавили в корзину, после товар закончился на складе и оформить заказ невозможно.

Сервис должен с помощью docker-compose. Должно быть использовано паттерны проектирования: Чистая архитектура, MVC.

**Вопросы к практической работе**

1. Аннотация Service.
2. Паттерн MVC.
3. Паттерн «Чистая архитектура».
4. Описать все слои RestFull приложения и описать зачем каждый из них.

**Критерии оценки**

**За выполнение данной практической работы можно максимально получить 2 балла.**

Критерии на выставление 2 баллов:

* Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
* Показана полная работоспособность серверной конфигурации в режиме реального времени.
* Дан полный и развернутый ответ на все вопросы преподавателя, как по вопросам к практике, так и по дополнительным вопросам к выполненному заданию.

Критерии на выставление 1 балла:

* Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
* Показана полная работоспособность серверной конфигурации в режиме реального времени.
* Дан полный и развернутый ответ на все вопросы преподавателя на вопросы к практической работе, но дополнительные вопросы остались не отвечены: студент не смог полностью описать и аргументированно устно объяснить ход проделанной работы, все шаги, студент не может объяснить и описать используемые технологии.

Критерии на выставление 0 баллов:

* Соблюдены общие требования выполнения практических работ, представленные в документе “Требования к выполнению практических работ”.
* Не показана полная работоспособность серверной конфигурации в режиме реального времени.
* Студент не смог ответить ни на вопросы к практической работе, ни на вопросы к ходу выполнения работы.