Лекция 1. Архитектура серверных частей интернет-ресурсов.

"Если вы считаете хорошую архитектуру слишком дорогой, попробуйте использовать плохую". — Брайан Фут (Brian Foote) и Джозеф Йодер (Joseph Yoder).

Список основных терминов и сокращений.

Сервер (программное обеспечение) - программный компонент вычислительной системы, выполняющий сервисные (обслуживающие) функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

Сервер (аппаратное обеспечение) - выделенный или специализированный компьютер для выполнения сервисного программного обеспечения без непосредственного участия человека.

Клиент — это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

Программа, являющаяся клиентом, взаимодействует с сервером, используя определённый протокол. Она может запрашивать с сервера какие-либо данные, манипулировать данными непосредственно на сервере, запускать на сервере новые процессы и т. п. Полученные от сервера данные клиентская программа может предоставлять пользователю или использовать как-либо иначе, в зависимости от назначения программы. Программа-клиент и программа-сервер могут работать как на одном и том же компьютере, так и на разных. Во втором случае для обмена информацией между ними используется сетевое соединение.

На одном и том же компьютере могут одновременно работать программы, выполняющие как клиентские, так и серверные функции. Например, веб-сервер может в качестве клиента получать данные для формирования страниц от SQL-сервера.

Сервер и клиент (рабочая станция) могут иметь одинаковую аппаратную конфигурацию, так как различаются лишь по участию в своей работе человека.

База данных — это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категорировать. Базы данных функционируют под управлением систем управления базами данных (сокращенно СУБД).

API (Application Programming Interface - прикладной программный интерфейс) - набор функций и подпрограмм, обеспечивающий взаимодействие клиентов и серверов.

API (в клиент-сервере) - описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

Сервис - легко заменяемый компонент сервисно-ориентированной архитектуры со стандартизированными интерфейсами.

Веб-сервис(веб-служба) - идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами.

Web API - используется в веб-разработке, содержит, как правило, определённый набор HTTP-запросов, а также определение структуры HTTP-ответов, для выражения которых используют XML− или JSON−форматы.

Web API является практически синонимом для веб-службы, хотя в последнее время за счёт тенденции Web 2.0 осуществлен переход от SOAP к REST типу коммуникации. Веб-интерфейсы, обеспечивающие сочетание нескольких сервисов в новых приложениях, известны как гибридные.

Архитектура клиент-сервер.

Основы архитектуры клиент-сервер.

Когда идет речь о вычислительных моделях, то модель клиент-сервер заняла прочное место среди методов распределенных вычислений. Часто данную модель называют просто архитектура клиент-сервер. Данная модель — это идея разделения системы или приложения на отдельные задачи, размещаемые на различных платформах для большей эффективности. Уже применение данной идеи лежит в основе архитектуры клиент-сервер, распределенных вычислений, архитектуры приложений и т.д. На рисунке 1 представлена общая структура архитектуры клиент-сервер:

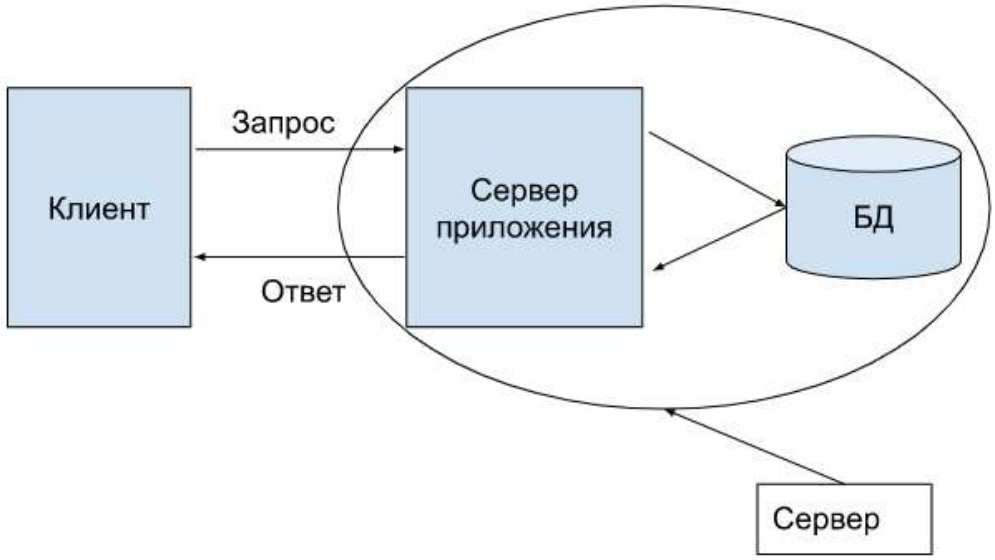

Рисунок 1. Общая структура архитектуры клиент-сервер.

Модель состоит из клиента, имеющего дружественный интерфейс. Клиент представляет собой программу представления данных, которая для их получения посылает запросы серверу, который в свою очередь может делать запрос к базе данных, обрабатывает данные и возвращает их к клиенту. Возможны случаи разделение обработки данных, когда часть работы сервера в виде обработки данных выполняет клиент. Но нужно понимать, что в этом случае очень важно разделение обязанностей и уровней доступа к данным на стороне клиента.

Если рассматривать в общем случае как модель, архитектура клиент-сервер — это сетевое окружение, в котором управление данными осуществляется на серверном узле, а другим узлам (клиентам) предоставляется доступ к данным.

К достоинствам клиент-серверной архитектуры можно отнести:

- основная нагрузка ложится на сервер(а), в системе имеется одна (или несколько) мощная(ых) серверная(ых) конфигурация часто сложной конструкции и множество “слабых” машин клиентов, что снижает общую стоимость всей системы;
- данные находятся в безопасности, так как сервер дает клиенту только требуемую выборку данных для клиента, основываясь на его уровне доступа;
- разграничение полномочий между клиентом и сервером;
- кроссплатформенность - реализаций клиента может быть сколько угодно: от веб-браузера до приложений мобильных платформ;
- нет дублирования сервисов на каждом клиенте, основная обработка данных лежит на сервере.

Нужно понимать, что наряду с плюсами существуют некоторые минусы:

- неработоспособность сервера может сделать неработоспособной всю информационную систему. Неработоспособным сервером следует считать сервер, производительности которого не хватает на обслуживание всех клиентов, а также сервер, находящийся на ремонте, профилактике и т. п., в случае отсутствия горизонтальной масштабируемости (см. далее);
- поддержка работы данной системы требует отдельного специалиста;
- высокая стоимость серверного оборудования.

С рассмотрения модели клиент-сервер спустимся на уровень ниже в конкретику интернет-ресурсов и введем следующее определение:

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер (в широком смысле).

Четырьмя основными компонентами архитектуры веб-приложений являются:

- Клиент – в общем случае это браузер, но в ситуации взаимодействия двух серверов один является сервером, а другой клиентом.
- Веб-сервер – это сервер, принимающий запросы от клиентов и выдающий им ответы. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.
- База данных.
- API.

Самая простая клиент-серверная конфигурация, изображенная на рисунке 1, не является единственно возможной. Существуют более сложные конфигурации, состоящие из множества серверов, так называемые трехзвенные и при их расширение н-звенные архитектуры. Рассмотрим эти основные архитектуры:

- двухзвенная архитектура(изображена на рисунке 1). Состоит из 2 базовых компонентов. Сервер напрямую отвечает на запросы клиентов;
- трехзвенная архитектура(изображена на рисунке 2). Сервер уже состоит из более двух самостоятельных частей, которые взаимодействуют друг с другом.

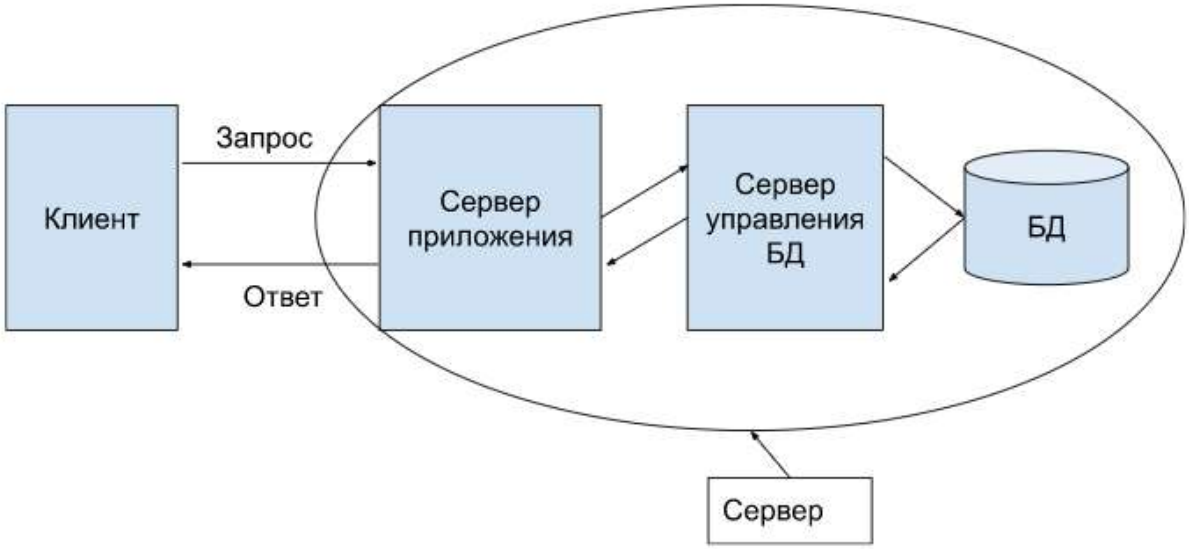

Рисунок 2. Трехзвенная архитектура.

Плюсами трехзвенной архитектуры дополнительно являются:

1. Высокая степень гибкости и масштабируемости.
2. Высокая степень безопасности (т.к. защиту можно определить для каждого сервиса или уровня).
3. Высокая производительность (т.к. задачи распределены между серверами).

Виды сервисов

Существует великое множество возможных сервисов, как самостоятельных, так и в составе приложений. Рассмотрим возможные виды сервисов:

- Серверы приложений.

Сервер приложений (англ. application server) — это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения.

- Веб-серверы.

Являются подвидом серверов приложений. Изначально предоставляли доступ к гипертекстовым документам по протоколу HTTP. Сейчас

поддерживают расширенные возможности, в частности, передачу произвольных данных.

- Серверы баз данных.

Серверы баз данных используются для обработки запросов. На сервере находится СУБД для управления БД и ответов на запросы.

- Файл-серверы.

Файл-сервер хранит информацию в виде файлов и предоставляет пользователям доступ к ней. Как правило, файл-сервер обеспечивает и определенный уровень защиты от несанкционированного доступа.

- Прокси-сервер.

Прокси-сервер (от англ. proxy - представитель, уполномоченный; часто просто прокси, сервер-посредник) - промежуточный сервер (комплекс программ) в компьютерных сетях, выполняющий роль посредника. Существует несколько видов прокси-серверов:

  - Веб-прокси — широкий класс прокси-серверов, служащий для кэширования данных.
  - Обратный прокси — прокси-сервер, который, в отличие от веб-прокси, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Часто используется для балансировки сетевой нагрузки между несколькими веб-серверами и повышения их безопасности, играя при этом роль межсетевого экрана на прикладном уровне.

- Файрволы (брандмауэры).

Межсетевые экраны, анализирующие и фильтрующие проходящий сетевой трафик, с целью обеспечения безопасности сети.

- Почтовые серверы.

Предоставляют услуги по отправке и получению электронных почтовых сообщений.

Понятие масштабируемости.

Вернемся к 1 пункту плюсов трехуровневой архитектуры, а точнее к понятию масштабируемости. Масштабируемость - способность работать с увеличенной нагрузкой путем наращивания ресурсов без фундаментальной перестройки архитектуры и/или модели реализации при добавлении ресурсов. Система называется масштабируемой, если она способна увеличивать производительность пропорционально дополнительным ресурсам. Основными являются горизонтальная и вертикальная масштабируемость.

Вертикальная масштабируемость(англ. scaling up) представляет собой увеличение производительности компонентов серверной системы в интересах повышения производительности всей системы. Следует понимать, что система ограничена объемом памяти, скоростными характеристиками и т.д. Данный метод не снимает нагрузку на всю систему, но является самым простым. Ярким примером является увеличение оперативной памяти, установка более мощного процессора.

Горизонтальная масштабируемость(англ. scaling out) представляет собой как разбиение системы на более мелкие структурные компоненты и разнесение их, так и увеличение количества компонентов, параллельно выполняющих одну и ту же функцию. Данный метод часто требует изменение программного обеспечения для налаживания взаимодействия между новыми компонентами. Частым примером является добавление еще одного сервера тех же характеристик к существующему.

Протоколы передачи данных.

Протокол передачи данных - набор определенных правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

Сетевой протокол - набор правил и действий (и их очередности), позволяющий осуществлять соединение и обмен данными между двумя и более включенными в сеть устройствами. Сетевые протоколы предписывают правила работы компьютерам, которые подключены к сети. Они строятся по многоуровневому принципу. Протокол некоторого уровня определяет одно из технических правил связи.

В общей классификации протоколы делятся на низкоуровневые, протоколы верхнего уровня и протоколы промежуточного уровня. К промежуточному уровню относятся коммуникационные протоколы и протоколы аутентификации. Протоколами верхнего уровня являются прикладные, сеансовые протоколы и протоколы представления. Физический, канальный, сетевой и транспортный протоколы относят к низкоуровневым протоколам.

Протоколы определяются следующими организациями: IETF (Internet Engineering Task Force), IEEE (Institute of Electrical and Electronics Engineers), ISO (International Organization for Standardization), ITU-T (International Telecommunication Union — Telecommunication sector).

В настоящее время для сетевых протоколов используется две основных сетевых модели: сетевая модель OSI (Open System Interconnection - взаимодействие открытых систем) и стек протоколов TCP/IP.

Вот как традиционно протоколы TCP/IP вписываются в модель OSI:

Распределение протоколов по уровням модели OSI

	TCP/IP	OSI	
7	Прикладной	Прикладной	HTTP, SMTP, SNMP, FTP, Telnet, SSH, SCP, SMB, NFS, RTSP, BGP
6		Представления	TLS, SSL
5		Сеансовый	RPC, NetBIOS, PPTP, L2TP
4	Транспортный	Транспортный	TCP, UDP, GRE

3	Сетевой	Сетевой	IP, ICMP, IGMP, OSPF, RIP, IPX
2	Канальный	Канальный	Ethernet, Token ring, HDLC, PPP, X.25, Frame relay, ISDN
1		Физический	электрические провода, радиосвязь, волоконно-оптические провода, инфракрасное излучение

В рамках данного курса мы ограничимся рассмотрением 7 и 6 уровней модели OSI или 4 уровня модели IP.

Модель OSI : 6 и 7 уровни логической модели работы сети:

- 6-ой уровень, уровень представления служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- 7-ой уровень, прикладной уровень является пограничным между прикладной программой и другими уровнями, обеспечивая удобный интерфейс связи для сетевых программ пользователя.

Аналогично рассмотренному, стек протоколов TCP/IP содержит 4-ый - прикладной уровень (application layer) для решения тех же задач.

Наиболее известные прикладные протоколы, используемые в сети Интернет:

- Протокол RTP (Real-time Transport Protocol), протокол работает на прикладном уровне (OSI - 7) и используется при передаче трафика реального времени.
- HTTP (Hyper Text Transfer Protocol) — это протокол передачи гипертекста.
- FTP (File Transfer Protocol) — это протокол передачи файлов со специального файлового сервера на компьютер пользователя.
- POP3 (Post Office Protocol) — это стандартный протокол почтового соединения.

- SMTP (Simple Mail Transfer Protocol) — протокол, который задает набор правил для передачи почты.
- TELNET — это протокол удаленного доступа.

В массе своей эти протоколы работают поверх TCP или UDP и привязаны к определённому порту, порты определены Агентством по выделению имен и уникальных параметров протоколов (IANA),например:

- HTTP на TCP-порт 80 или 8080;
- FTP на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд);
- SSH на TCP-порт 22;
- запросы DNS на порт UDP (реже TCP) 53;
- обновление маршрутов по протоколу RIP на UDP-порт 520.

Понятия “толстого” и “тонкого” клиента.

При классификации компонентов архитектуры клиент-сервер существует понятия “толстый” и “тонкий” клиент.

При применении толстого клиента полная функциональность приложения обеспечивается вне зависимости от сервера. Схематично это изображено на рисунке 3. В данном случае сервер чаще всего выступает в роли хранилища информации, а вся логика приложения, как и механизм отображения данных располагаются и выполняются на клиенте. Даже при отсутствии соединения с сервером работа ведется с локальными копиями данных, а при возобновлении соединения происходит синхронизация данных.

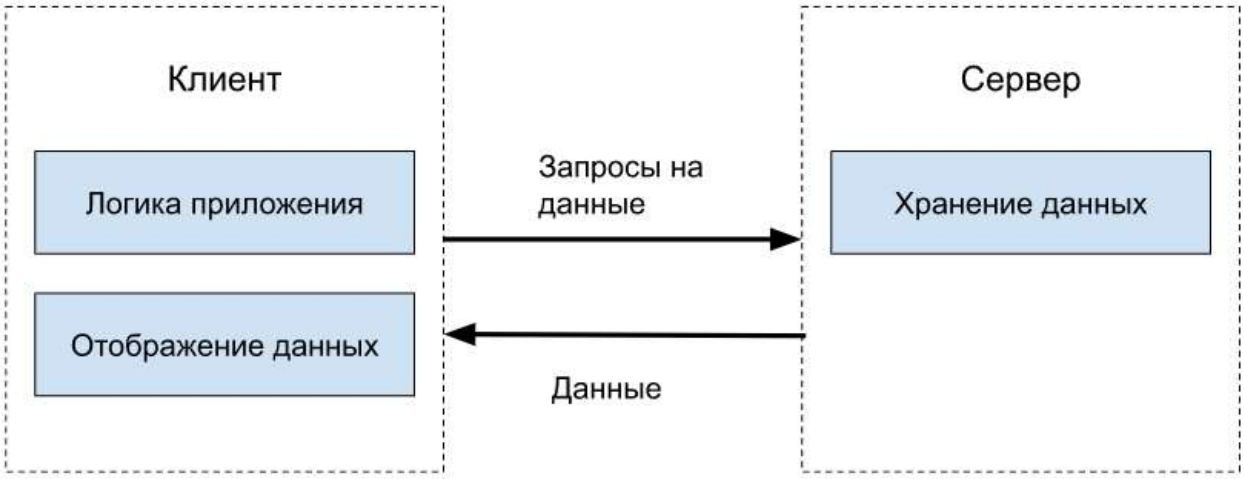

Рисунок 3. Толстый клиент.

Тонким клиентом называют компьютеры и программы, функционирующие в терминальной или серверной сети. Множество задач по обработке данных осуществляются на главных компьютерах, к которым присоединено приложение и компьютер. Тонкий клиент же в отличие от толстого только отображает данные, принятые от сервера. Вся логика приложения выполняется на более производительном сервере, что не требует клиентских мощностей, кроме хорошего и стабильного канала связи. К сожалению, любой сбой на сервере и в канале связи влечет “падение” всего приложения. Данная технология изображена на рисунке 4.

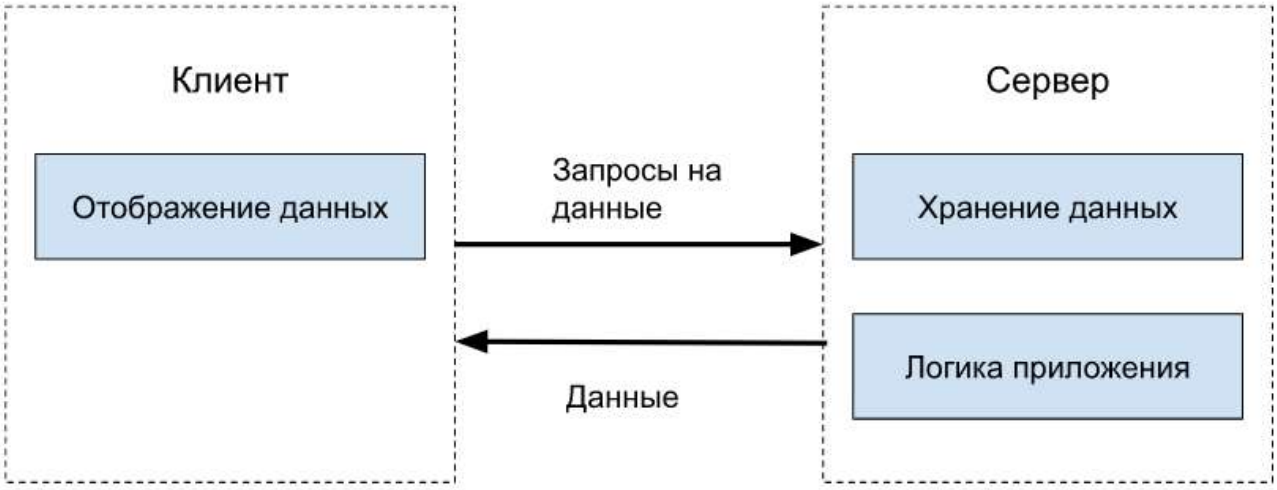

Рисунок 4. Тонкий клиент.

Рассматривая два этих подхода в совокупности, следует выделить существенные различия:

- Различные требования к каналу связи, для тонкого клиента это существенно важно, т.к. потеря соединения сильно влияет на работоспособность и даже полностью тормозит работу приложения.
- Обширные функции продукта при использовании толстого клиента не совместимы с политикой безопасности, при использовании же тонкого клиента обработка данных идет на уровне сервера, что безопаснее.
- При использовании толстого клиента значительно усложняется совместная работа с данными, их синхронизация и обновление, хотя данную проблему решает методика тонкого клиента, но все же данную проблему в большей или меньшей степени можно отнести к обеим методикам.
- При использовании толстого клиента требуются значительные мощности на пользовательском компьютере для обработки данных, что повышает требования к пользовательским машинам.
- Технология тонкого клиента упрощает работу с исправлением ошибок в программном обеспечении, его обновлении и т.д., за счет того, что исправление логики приложения идет только 1 раз на сервере.

Принципы построения архитектуры веб-приложений.

Для понимания различий между архитектурой системы и архитектурой программного обеспечения в нашем случае веб-приложения обратимся к определению архитектуры программного обеспечения: Архитектура программного обеспечения (англ. software architecture) — совокупность важнейших решений об организации программной системы.

Архитектура включает:

- выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;
- соединение выбранных элементов структуры и поведения во всё более крупные системы;
- архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

Разработка архитектуры очень важный и сложный процесс. От полученного результата зависит дальнейшее развитие программного продукта. Или будет идти наслоение ненужных слоев и идей, перерастающее в огромный ком со сложными взаимоотношениями внутри, который без хорошо написанной документации будет сложно содержать и поддерживать. Или архитектура изначально была выбрана соответственно проекту и требованиям, что в результате дает расширяемый и модернизируемый продукт.

Для решения проблем проектирования были созданы архитектурные конструкции, называемые паттернами проектирования или часто коротко паттернами. Существует фундаментальный паттерн (шаблон), нашедший свое применение повсеместно, не только в веб-разработке. Название паттерну дают первые буквы его основных компонентов: Model View Controller.

Первая часть данного паттерна это модель (Model). Это представление содержания функциональной бизнес-логики приложения. Модель, как и любой компонент архитектуры под управлением данного паттерна не зависит от остальных частей продукта. То есть слой, содержащий модель ничего не знает о элементах дизайна и любом другом отображении пользовательского интерфейса. Подводя итог, модель обладает следующими признаками:

- модель — это бизнес-логика приложения;
- модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;

- для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);
- для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения.

Представление (View) это есть отображение данных, получаемых от модели. Никакого влияния на модель представление оказать не может. Данное разграничение является разделением компетенций компонентов приложения и влияет на безопасность данных. Если рассматривать интернет-ресурсы представлением является html-страница.

Третьим компонентом системы является контроллер. Данный компонент является неким буфером между моделью и представлением. Обобщенно он управляет представлением на основе изменения модели.

Существуют разные виды MVC-паттерна, различающихся 3 компонентом системы. Наиболее распространенными видами являются:

- Model-View-Presenter;
- Model-View-View Model;
- Model-View-Controller.

Особенностью паттерна Model-View-Presenter является то, что он позволяет создавать абстракцию представления. Для реализации данного метода выделяется интерфейс представления. А презентер получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу меняет модель. Данный подход представлен на рисунке 5.

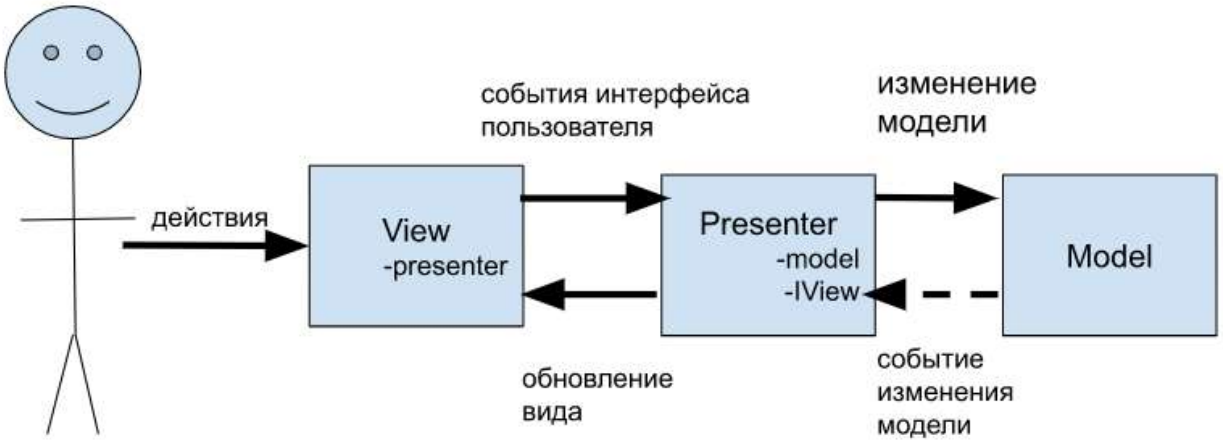

Рисунок 5. Паттерн Model-View-Presenter.

Признаки подхода с использованием презентера:

- двусторонняя коммуникация с представлением;
- представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
- презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
- одному презентеру соотвествует одно отображение.

Особенностью паттерна Model-View-View Model является связывание элементов представления со свойствами и событиями View-модели. Схематично данная идея изображена на рисунке 6.

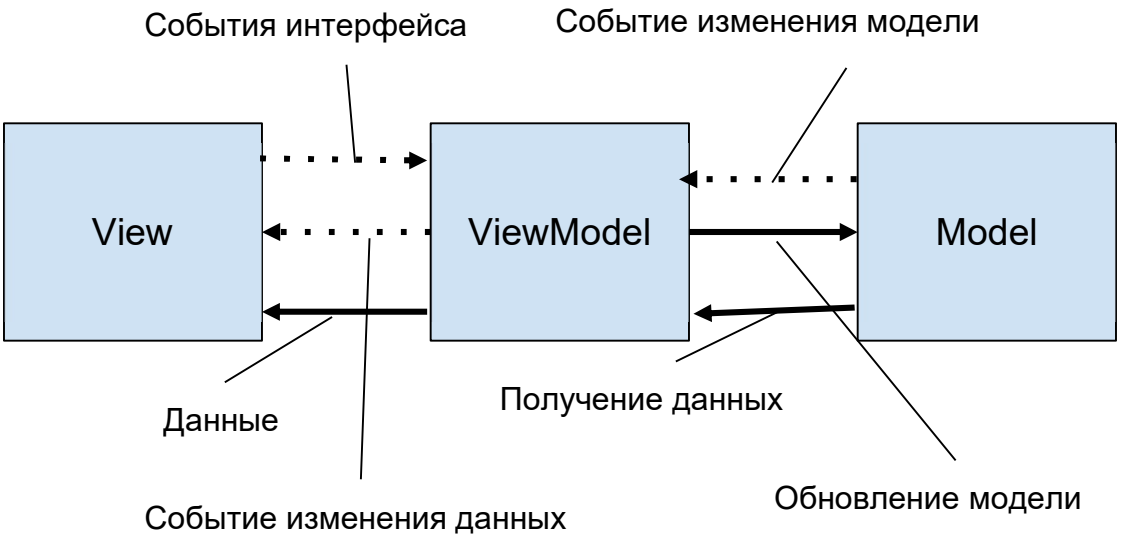

Рисунок 6. Паттерн Model-View-View Model.

Признаками данного подхода являются:

- Двусторонняя коммуникация с представлением.
- View-модель — это абстракция представления. Означает, что свойства представления совпадают со свойствами View-модели / модели.
- View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings).
- Одному экземпляру View-модели соответствует одно отображение.

Особенностью паттерна Model-View-Controller является то, что контроллер и представление зависят от модели, но при этом сама модель не зависит от двух других компонентов. Представление данного паттерна представлено на рисунке 7.

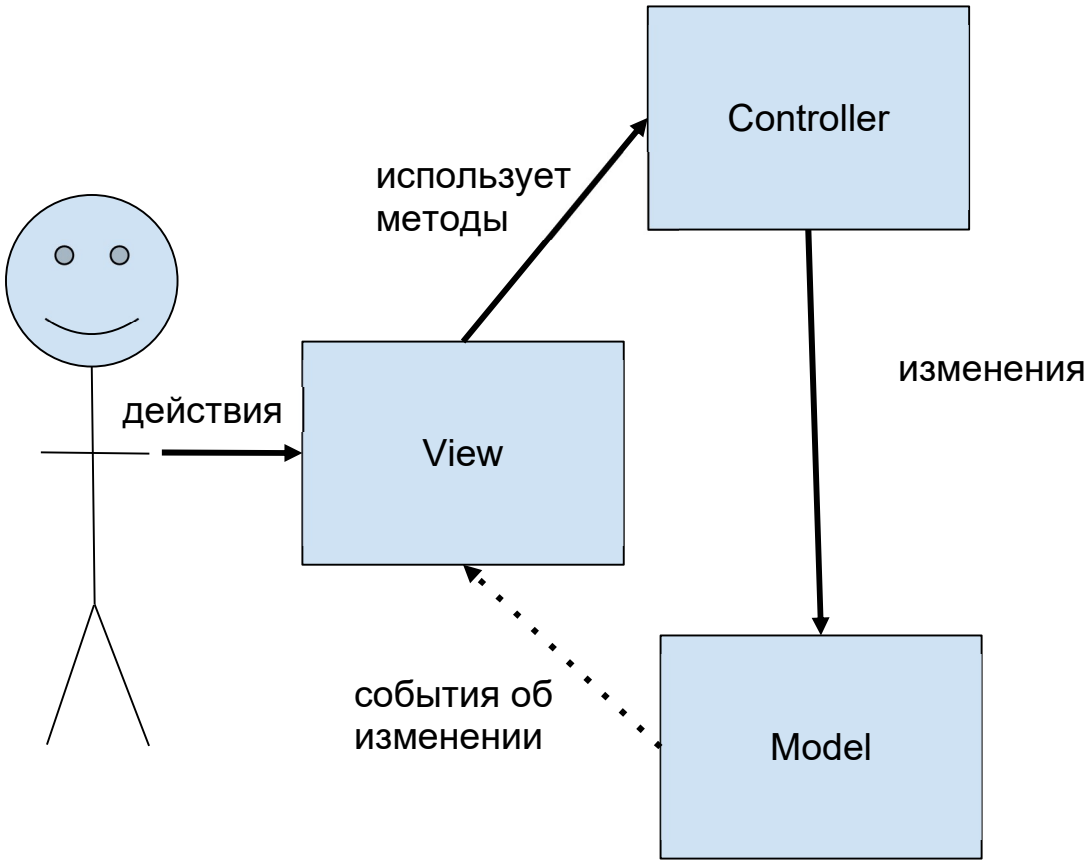

Рисунок 7. Паттерн Model-View-Controller.

Признаками данного подхода являются:

- Контроллер определяет, какое представление должно быть отображено в требуемый момент. Если рассматривать применение для разработки веб-приложений, то контроллер управляет запросами пользователя. Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.
- События представления могут повлиять только на контроллер. Контроллер может повлиять на модель и определить другое представление.
- Возможно несколько представлений только для одного контроллера.

Данный вариант чаще всего используется при разработке веб-приложений.

Задачи курса.

На основе вышесказанного целесообразно сформулировать задачи данного курса:

- изучение серверов и серверной конфигурации, их настройка;
- изучение возможных реализаций взаимодействия между клиентом и серверов;
- изучение создания сервисов (как компонента веб-приложения) и их внутренней архитектуры;
- изучение архитектуры серверных приложений, знакомство с основными типами архитектур;
- фреймворки для создания серверных частей приложений.

Список источников.

1. Дергачев А. М. Проблемы эффективного использования сетевых сервисов / Научно-технический вестник СПбГУ ИТМО. 2011. № 1 (71). С. 83-87.
2. Розенфельд Л., Морвиль П. Информационная архитектура в Интернете, 2 е издание. – Пер. с англ. – СПб: Символ Плюс, 2005 – 544 с., ил.
3. Спинеллис Д., Гусиос Г. Идеальная архитектура. Ведущие специалисты о красоте программных архитектур. – Пер. с англ. – СПб.: Символ Плюс, 2010 – 528 с., ил.
4. Фаулер, Мартин. Ф28 Архитектура корпоративных программных приложений.: Пер. с англ. — М.: Издательский дом "Вильяме", 2006 — 544 с.: ил. — Парал. тит. англ.
5. Голицына О.Л., Максимов Н.В., Партыка Т.Л., Попов И.И. ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. - 2-е изд. - Москва: ФОРУМ - ИНФРА-М, 2008. - 395 с.

6. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. — СПб.: Питер, 2003. — с. 83-93 — (Серия «Классика computer science»). ISBN 5-272-00053-6-.