**1. Введение в web-технологии.** Web. Internet. Общее устройство web-приложений. Сети. Топология сети. Клиент-Сервер. Модель OSI. Протокол. TCP/IP. HTTP, версии. URI\URL\URN. HTTP Request. HTTP Response. Методы/заголовки/тело. GET и POST. Передача параметров. Коды ответов. Основные заголовки. Content-Type. MIME-типы. HTML. Основные теги. Формы. XML. JSON. Статические сайты и их проблема. Динамическая генерация.

Web - Система связанных ссылками документов, расположенных на различных компьютерах сети Интернет (Internet). Работает на основе Интернета, является основным способом доступа к информации

Интернет

• Глобальная сетевая инфраструктура

• Сеть сетей

• Позволяет любым компьютерам в ней общаться друг с другом – с помощью разных

протоколов

• Web использует HTTP

Веб-приложение - клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети.

Как работают веб-приложения?

Через браузер

Пользователь отправляет запрос(request). Сервер обрабатывает и отправляет ответ(response)

Сети

- Локальные (сетевой принтер, диск, игра по сетке)

- Глобальные (интернет)

Топология сети

Это граф, вершинам которого соответствуют конечные узлы сети (компьютеры и коммуникационное оборудование (маршрутизаторы)), а рёбрам — физические или информационные связи между вершинами.

Клиент

• Участник(узел) сети

• Запрашивает и получает данные

Сервер

• Участник(узел) сети

- Обычно более мощный компьютер

- Иногда один из клиентов

• Получает запросы клиентов

• Обрабатывает запросы (формирует ответ)

• Отправляет ответ клиенту

Кто отправляет данные пользователю?

Сервер.

Как клиент/сервер знают куда посылать запрос/ответ?

С помощью протоколов(http, в нашем случае).

Основным объектом манипуляции в HTTP является ресурс, на который указывает URL в запросе клиента.

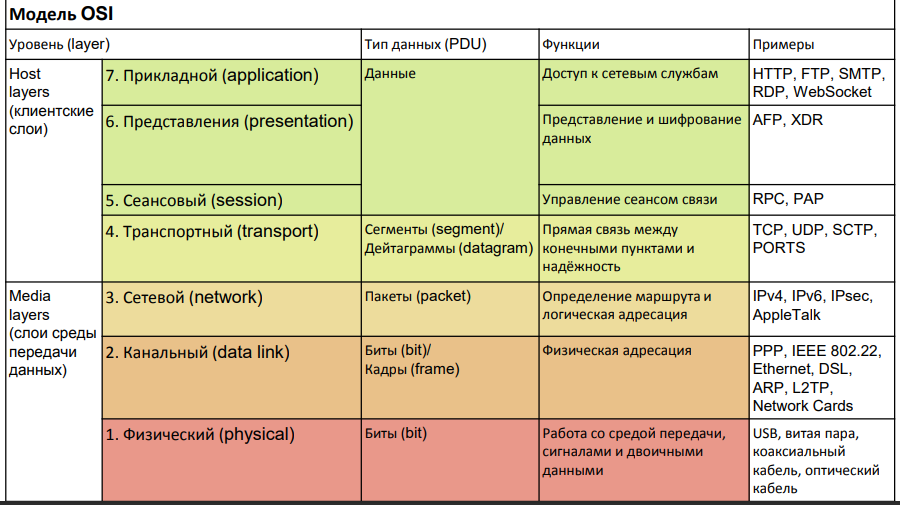
Модель OSI

• Open Systems Interconnection basic reference model (Эталонная Модель Взаимодействия Открытых Систем)

• 7 уровней взаимодействия компьютеров в сетях

- Со своими форматами данных

- Со своими правилами передачи (протоколами)



Протокол передачи данных – набор соглашений интерфейса логическоро уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок.

Сетевой протокол – правила, действия, форматы, регламентирующие обмен данными между узлами сети

TCP/IP

• Transmission Control Protociol/Internet Protocol(DARPA)

• Прикладной уровень

• Транспортный уровень

• Сетевой(межсетевой) уровень

• Канальный уровень

HTTP

HTTP — это протокол, позволяющий получать различные ресурсы, например HTML-документы. Основой HTTP является [технология «клиент-сервер»](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80). Протокол HTTP лежит в основе обмена данными в Интернете. В основе самого HTTP лежит HttpListener. (Во второй и третьей лекции это подробно разобрано).

URI\URL\URN.

• URI (Uniform Resource Identifier) – унифицированный (единообразный) идентификатор ресурса. Это строка символов, которая используется для идентификации какого-либо ресурса по его адресу или по его имени, либо по тому и тому вместе

• URL (Uniform Resource Locator) – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса.

• URN (Uniform Resource Name) – единообразное название (имя) ресурса

HTTP Request.

Структура:

- Стартовая строка GET / HTTP/2(метод, path, протокол)

- Заголовки (Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Connection)

- Тело (для GET отсутствует)

HTTP Response.

Структура:

- Протокол Код Ответа HTTP/2 200 OK

- Заголовки(d ate, server, content-language, last-modified, content-type, content-encoding)

- Тело ответа(содержимое)

GET и POST.

• Описывают действие над ресурсом

• Самые часто используемые

**GET** – получить определённый ресурс. со стороны сервера на клиента

• Самый используемый запрос

– Ввод в адресную строку

– Нажатие на ссылку

• Ничего не меняет на сервере (идемпотентный)

• Может начать какой-либо процесс. В этом случае в ответе надо дать информацию о ходе процесса

GET /path/resource?param1=value1&param2=value2 HTTP/1.1

**POST** – создать новый ресурс. клиент отправляет post запросы

• Метод для изменения чего-то на сервере, обычно добавления

– Требующего предварительной обработки

• Часто используется для отправки данных с веб-форм или загружаются файлы на сервер

• Не является идемпотентным

• Хранит параметры в теле запроса

– Поэтому пароли и другую конфиденциальную информацию посылаем внутри POST-запросов

Передача параметров.

Параметры запроса – основное средство передачи информации от клиента к серверу(вместе с URL)

Передаются после ?

Коды ответов.

5 блоков

• 1xx – информационные ответы

101 – смена протокола

• 2хх – Success

• 3хх – Redirection(ресурс переехал)

• 4xx – Client Error

403 - Forbidden

404 – Not Found

• 5xx – Server Error

500 – Internal Server Error

502 Bad Gateway

Основные заголовки.

• Общие:

Date, Cache-Control, Conneciton

• Закоголовки запроса

User-Agent, Accept, Accept-Language, Accept-Encoding

• Заголовки ответа

Keep-Alive, Last-Modified(когда ресурс был изменён), Age(как долго ресурс будет жить), Location, Server

• Заголовки сущности

Content-Length, Content-Language, Content-Encoding

Content-Type. MIME-типы.

Content-Type – тип содержимого, которое передаётся от сервера к клиенту или от клиента к серверу

Чтобы сервер/клиент знал что за файлы передаются и как их можно обрабатывать используется MIME(Multipurpose Internet Mail Extension), который записывается в значение Content-Type

Базовые типы MIME:

- application

- audio

- image

- message

- text

- video

- multipart

- … (можно расширять)

Пример:

Content-Type: text/html

Content-Type: text/css

Content-Type: text/json

HTML. Основные теги.

HTML – Hyper Text Markup Language, стандарт оформления документов в вебе.

Основные теги: html, body, head, title, a, h1-h6, ul, ol, li, div, img, br, p, form, input

Общие атрибуты тегов: class, id

Формы

Основная конструкция с помощью которой можно передавать информацию на сервер.

<form action=”/login” method=”?”>

<input type=”text” name=”username”/>

<input type=”password” name=”password”/>

<input type=”submit” value=”OK”>

</form>

XML. JSON.

Кроме HTML сущетсвуют еще форматы

• XML(eXtensible Markup Language)

* Формат хранения объектов
* Система произвольных тегов
* Строгий! Не прощает ошибок

• JSON(активноиспользуемый и более легковесный чем XML)

* Требует меньше памяти
* Быстрее передаётся

Статические сайты и их проблема.

Статические сайты:

• URI связан с файловой системой напрямую

• Пусть сайт расположен на сервере в папке srv/www/mysite

GET-запрос на www.mysite.ru/index.html

Вернёт файл /srv/www/mysupersite/index.html

Проблема статических сайтов:

• Одно содержимое для всех

- А интернет становится персональным

Умерли в начале 2000х

Динамическая генерация.

• Решение проблемы персонализации контента

• Содержимое ответа генерируется сервером при обработке запроса

**2. Введение в http-серверы**. Как работает сервер? Обработка запроса. Порты (параллельная работа сетевых приложе ний). Открытость портов. Номера портов. Web-сервер. CGI. IIS. Self-hosted приложения. HttpListener. HttpListenerRequest и HttpListenerResponse. OutputStream. Класс Stream. ContentLength64. Параметры. Маршрутизация. Варианты реализации. Web API.

Как работает сервер?

Принимает request от клиента,

Извлекает path, headers, params, content(если есть)

Генерирует content (http-response)

Обработка запроса.

Обработкой запросов занимаются сетевые приложения. Таких приложений на сервере может быть несколько.

Порты (параллельная работа сетевых приложений).

Так как на сервере может работать несколько приложений, то возникает вопрос, какое именно должно получить запрос от клиента.

Поэтому вводится еще один уровень абстракции, которое называется **Портом**

**•** На сервере к каждому приложению привязывается свой порт

• По умолчанию закрыты

• Записываются в заголовках протокола транспортного уровня OSI(TSP, UDP, …)

<http://10.12.13.14:1234/path/>

• Определяют процесс-получатель пакета

Открытость портов.

• Процессы могут слушать(listen) порты(ожидая событий)

• Открытый порт – доступный извне порт прослушиваемый процессом

• Закрытость порта определяется приложением firewall

Номера портов.

- HTTP: 80

- HTTPS: 443

- SSH: 22

- FTP: 21

- SMTP: 25

- POP: 110

• Внутри сервера запрос может переадресовываться другому приложению на другом порту

• Выдаются ОС процессам

• Порты одного протокола не пересекаются с портами другого (TCP и UDP, например)

• Во многих ОС прослушивание портов 0-1023 требует особых привелегий (root)

Web-сервер.

• Обрабатывает задаччи по обработке http запросов/ответов

• Примеры: Apache, nginx, IIS

• Сам ничего не генерирует. Фарух хуй. Работает вместе с (web-) приложениями, которые могут это сделать

Разница Web-сервера и сервера приложений?

Веб-сервер предназначен для обслуживания статических страниц, например HTML и CSS, тогда как сервер приложений отвечает за генерацию динамического содержимого путём выполнения кода на стороне сервера

CGI. IIS.

CGI(Common Gateway Interface) – стандарт взаимодействия web-сервера и сервера приложения

IIS – web-сервер, который лежит в основе Windows-сервер от Microsoft

• Управляет пулом web-приложений

• Разделяет адресные пространства

• Заботится об отказоустойчивости сайта и web-сервера в целом

• Как работает?

– Драйвер http.sys перехватывает запросы

– http.sys обращается к Windows Activation Service

– WAS запрашивает конфигурацию

– W3SVC получает конфигурационную информацию (данные пула приложений, параметры приложений)

– В отдельном процессе исполняется приложение (W3WP), формирующее ответ http.sys – http.sys отправляет ответ браузеру клиента

Self-hosted приложения.

Обычно разработка web-приложений на .NET Framework осуществляется с помощью инструментария ASP.NET

Но в .NET Core можно сделать Self-hosted приложения, которые будут использовать Kestrel-сервер

Self-hosted приложение – приложение, самостоятельно слушающее порт по некоторому адресу, обрабатывающее запросы и генерирующее ответы

Self-hosted приложения реализовываются с помощью классов HttpListener

HttpListener.

• Класс из System.Net для прослушивания подключений по протоколу HTTP

• Реализован на основе драйвера http.sys OC Windows.

• Можно прослушивать порт, пока жив объект класса

• Есть метод GetContext(позволяет синхронно ожидать подключение), который открывает доступ к объектам HttpListenerRequest и HttpListenerResponce

• Есть свойство Prefixes, через которое нужно установить адреса URI с помощью метода Add(), которые он будет прослушивать.

var listener = new HttpListener();

listener.Prefixes.Add(“http://localhost:8080/”);

listener.Start()

listener.Stop()

HttpListenerRequest

Описывает входящий HTTP-запрос для объекта HttpListener.

Из чего состоит?

• Объекты Url, RawUrl(без хоста и порта) – куда постылается запрос

• Headers – заголовки запроса

• QueryString – строка запроса с параметрами

HttpListenerResponse.

Работа с записью данных ответа сводится к использованию потоков данных

Из чего состоит?

• ContentLenght64 - Длина, которую сильно желательно задавать, чтобы клиент знал сколько нужно ждать до загрузки ответа

• OutputStream – выходной поток

OutputStream.

Класс Stream.

Абстрактный класс, находящийся в System.IO имитирующий поток byte, выстроенных в ряд

• Предоставляет базовые методы работы с потоками данных, а именно метод чтения / записи байта или массив байтов

var listener = new HttpListtener();

listener.Prefixes.Add(“http://localhost:8080/”);

listener.Start(); // старт прослушки порта

HttpListenerContext context = listener.GetContext(); // Ожидаем запрос клиента

var content = // формируем ответ

”<html>

<head>

<meta charset='utf-8'>

</head>

<body>Привет, мир!</body>

</html>”;

var bytes = Encoding.UTF8.GetBytes(content); // преобразовываем ответ в формат массива байт

context.Response.ContentLength64 = bytes.Length; // определяем длинну массива байт

StreamWriter sw = context.Response.OutputStream; // запись массива байт в выходной поток OutputStream

sw.Write(bytes,0,bytes.Length);

sw.Close();

listener.Stop();

Маршрутизация.

• Сопоставление пути соответствующему обработчику

Пример: если происхоидит запрос: GET vk.com/im, то у нас должен быть обработчик, который извлекает из БД все сообщения для текущего пользователя и отображает на страницу.

Варианты реализации(Как маршрутизировать запросы?).

Варианты выбора обработчика:

1. По точному совпадению path(пример выше)
2. По наибольшему совпадению
3. По расширению
4. На обработчик по умолчанию

• Парсить запрос, исполнять подходящий код(каждому маршруту прописать конкретный обработчик(хардкод))

– Как сделать без хардкода?

• Установить соответствие классам обработчикам по шаблонам

– Например, рефлексией + атрибуты

Web API.

Web-приложения могут генерировать не только web-страницы в качестве ответа на запросы, можно генерировать файлы с данными для их последующей обработки клиентом(json, xml, …)

Современная web-архитектура подразумевает, что наше приложение разбито на множество маленьких частей(веб-сервисов) и они между собой взаимодействуют посредством http запросов и ответов которые описаны в формате json, затем визуализируют этот json так как нужно.

**3. http-серверы.** Продолжение.

Чего у сервера несколько?

• У одного приложения – несколько клиентов, несколько компонентов

• Запросы одного клиента обрабатывают несколько компонент

- Авторизуемся, смотрим новости, переписываемся, слушаем музыку, это разные запросы для разных обработчиков

- **Нужно общее хранилище данных, общее для обработчиков, индивидуальное для каждого клиента**

HTTP-stateless.

Сам по себе HTTP протокол является stateless(не подразумевает сохранения состояния)

На сервере должна храниться информация, влияющая на обработку запросов

- по одному request`y могут генерироваться разные respons`ы

Состояние обработчика.

• Можно сохранять данные в ассоциативную коллекцию (словарь, глобальное хранилище)

- с уникальными строковыми ключами(так работает ViewState в ASP.NET WebForms)

• У каждой компоненты есть свои собственные данные, которые она использует при обработке запрос

- У авторазации – свои

- У сообщений – свои

- У новостей - свои

Сессия.

**Сессия** – один продолжительный этап работы клиента с некоторого браузера до момента его закрытия

**Состояние сессии –** индивидуальные данные которые сохраняются на сервере пока пользователь продолжает работу в web-приложение(выполняет несколько последовательных запросов)

• Обычно – словарь, хранящийся в памяти сервера

• Время хранения ограничено

- отсчёт стартует заново с момента последней активности(последнего запроса клиента например)

- в какой-то момент сессия очищается, если бы не очищалось, то мы бы выработали всю память, которая выделяется под наше веб-приложение

Хранение сессий.

• Может стать весельем в случае веб-фермы(когда есть несколько идентичных приложений, которые работают параллельно и обрабатывают запросы клиентов, в зависимости от того насколько они сильно нагружены(некая балансировка нагрузки))

Веселье потому что у каждого приложения будут свои локальные хранилища и может возникнуть проблема

• Типичные подходы к хранению сессий:

* Локальное хранилище, которое внутри нашего приложения(про которое было сказано выше)
* В отдельной службе(решит проблемы с веб-фермой)
* В базе данных

Пользователи.

• С помощью сессии можно проверить, был ли авторизован пользователь

• Можно в словаре сессии проверить наличие значения по ключу (например, currentUser)

* В зависимости от результата направить redirect к обработчику запроса или к форме авторизации

• 3 этапа Авторизации

- Идентификация – представление пользователя, сообщение о том, кто он такой

- Аутентификация – проверка доказательств (факторов) подтверждения личности(проверка сервера)

Идентификация – часть аутентификации

- Авторизация – проверка разрешений действий аутентифицированной персоне.

Redirect.

• Redirect – автоматический перевод на другую страницу, инициируемый сервером.

• Содержимое адресной строки меняется без участия пользователя

В HttpListenerResponce:

context.Responce.Redirect(“http://ya.ru”);

Клиенту возвращается response с кодом 302, в заголовках содержится Location с адресом перехода. Можно прописать и напрямую в заголовки

Варианты перенаправлений.

1. Перманентные (301, 308)
2. Временные (302, 303, 307)
3. Специальные (300, 304)

Еще варианты:

• С помощью мета-тегов html

<meta http-equiv=”Refresh” content=”0; URL=https://example.com/”>

• JavaScript

window.location = “https://example.com/”;

Cookie. Содержимое. Как передавать и принимать.

У сервера есть много клиентов => Как сервер определяет клиента работая именно с его сессией?

• У браузера есть кое-что, что позволяет серверу узнавать его

HTTP Cookie

• Пара строк “ключ-значение”, сохранённая в браузере для кокретного ресурса (домен + путь)

• Посылается с любым запросом к этому ресурсу

• Это они сохраняются, когда отмечаем галочку “запомнить меня”

- Чтобы реализовать это нужно в куках сохранить логин и зашифрованную строку, которую мы сохраним на сервере

• Сервер, приняв cookie, проводит автоматическую авторизацию пользователя

Сервер может:

• Обрабатывать запрос, учитывая пришедшие cookies от клиента

• Добавлять к ответу новые cookies или изменять/удалять существующие.

Содержимое Cookie:

• Имя, значение (строки) 1P\_JAR=2021-10-12-16;

• Expires – время жизни(GMT time)

• Domain, Path – область действия (к какой части доменного имени и path применяется) имеется в виду, что данные куки будут отправляться только когда запрос проходит по соответствующему домену и пути(path)

• Secure – передача только по HTTPS

• SameSite – некоторая доп.защита от CSRF-атак

• HttpOnly – запрет доступа из JS

Генерация ответа – шаблоны.

Проблема генерации ответа – спагетти код при попытке сделать динамику в html

Шаблонизаторы

• Использование html-шаблонов с вкраплениями кода для генерации конечных html-страниц

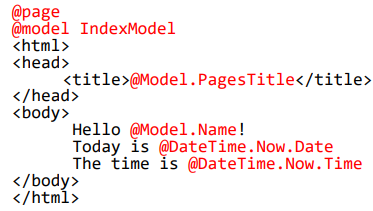
• Основная цель – отделить насколько это возможно представление от исходного кода

Razor, синтаксис.

• Cтандартный шаблонизатор для ASP .NET Core

• Подключение: services.AddRazorPages();

• Синтаксис:



Передача информации в шаблон.

• ViewData(словарь)

• ViewBag(объект)

• Модель представления(пример выше)

Мастер-страницы.

• \_Layout.cshtml (Основная вёрстка нашей страницы)

• \_ViewStart.cshtml (Шаблон, что будет везде подключен)

• \_ViewImports.cshtml (Шаблон, который содержит в себе все юзинги которые будут подключены для любого шаблона, который реализован в нашем приложение)

\_Layout.chtml.

• Находится в Views/Shared

• Главное что содержит - @RenderBody(), куда подставляется содержимое других страниц

Секции.

@RenderSection(“Footer”)

//на мастер странице. После чего на странице:

@section Footer {

Все права защищены. Site Corp. 2016.

}

**4. UI, event-driven, клиенты.**

UI.

User Interface(UI) – пользовательский интерфейс

• Обеспечивает взаимодействие пользователя-человека с программно-аппаратными компонентами компьютерной системы

Пример: командная строка(всё начиналось с неё)

GUI с пользовательской и программисткой точек зрения.

Промежуточным этапом между консольными приложениями и GUI выступили программы файловых менеджеров(Norton Commander, FAR Manager)

Graphic User Interface(GUI)

• Первый графический интерфейс появился у ксероксов

• Windows, Unix(Linux, Free BSD)

Gui с пользовательской точки зрения:

• Приложение – набор окон(если говорим об ОС)

• В каждом окне – стандартные графический элементы

- формы, кнопки, меню, списки, поля, …

• Клавиатурой, мышью управляем приложением (кликаем, перетаскиваем, печатаем…), ну или тач-скрин

GUI Developers viewpoint

• Приложение – набор окон(window, frame, form)

• Окно – набор графических элементов (компонентов, виджетов), имеющих определенно расположение (layout)

• Каждый графический элемент имеет свойства(координаты, текст, ширина)

Библиотеки разработки приложений с GUI

• Низкоуровневые:

- WinAPI, Xlib, Carbon(До 2012)

• Высокоуровневые

- QT, GTK+, WinForms, WPF, Swing, Cocoa, …

Преимущества web-приложений над desctopными

• Не нужна установка

- только браузер и интернет

• Легко поддерживать

- Обновил на сервере и готово

• Кросс-платформенность

- Нужна кроссбраузерность, но её обеспечить проще

Недостаток: нагрузка на пользовательские компьютеры из-за того что браузер пользователя воспринимается как часть нашего приложения

UI в браузерах.

• Браузер можно вопспринимать как любое другое оконное графическое приложение

- Есть меню, полде ввода (адресная строка, поиск) и др.

• С помощью браузера работаем с web-приложениями(браузер по сути это интерфейс для нашего веб-приложения)

- По аналогии ОС и десктоп приложений

JS

Есть разные события. Есть те которые инициируют запрос к серверу(формы), есть те которые должны отрабатывать на текущем компьютере пользователя, без запроса к серверу. Тоесть на стороне браузера тоже должен исполняться код => JavaScript

• JavaScript ≈ ECMAScript(ядро, базовая часть языка) + DOM + BOM

DOM и BOM

• DOM – Document Object Model

* Позволяет получать доступ к структуре HTML (в виде дерева) и менять его содержимое
* HTML-элементы – объекты со свойствами. DOM даёт JS возможность изменять эти свойства

• BOM – Browser Object Model – с помощью него можно управлять самим браузером

* Надмножество DOM
* Плохо стандартизировано(разные браузеры поддерживают по разному)
* Управление окнами (windwo) / управление фреймами (frames)
* Системные диалоги (alert(), promt(), confirm())
* Управление адресом открытой страницы(location)

AJAX. Преимущества и недостатки.

Ajax (Asynchronous Javascript and XML)

- Это набор методов веб-разработки, которые позволяют веб-приложениям работать асинхронно — обрабатывать любые запросы к серверу в фоновом режиме

• Фоновый (без перезагрузки страницы) обмен информацией с сервером + динамическое изменение содержимого страницы

Пример использования AJAX:

Что если при этом данные нужно брать с сервера?

- Читаем новости VK, добавилась еще одна

- Переписываемся, пришло сообщение

Преимущества AJAX

• Экономия трафика

• Интерфейс остается отзывчивым во время выполнения длительной операции(асинхроность)

• Возможность реализации динамического графического интерфейса

Недостатки AJAX

• История посещений

• Поисковики(до определенного момента)

• Статистика

• Усложнение проекта

• Включенный JS

• Кодировки

• Плохое поведение на нестабильных соединениях

JS-библиотеки

•jQuery(оболочка над DOMом, позволяет коротко обращаться с элементам, выполнять AJAX-запросы)

• AngularJS

• React

• Vue.js

• Node.js

**5. Взаимодействие сервер-клиент**.

ASP.NET Core Razor Pages.

Polling. Long polling.

Что если нужно обновлять данные на клиента по событиям на сервере?(новое сообщение в переписке, новость в ленте) => polling

* Polling

• Опрос сервера на наличие событий с заданной периодичностью

- Long Polling

• Клиент посылает запрос, сервер держит соединение открытым, посылает

данные, закрылось – открывается новое соедиенение

Минусы: открытие любого http-соединения это затратная операция, поэтому такой вариант не подходит

Нужен активный сервер имеющий возможность генерации событий, чтобы пользователь слушал и обрабатывал события сервера => есть несколько реализаций (2), о них ниже

Server Sent Events.

Отличие от Long Polling в том что соединение не обрывается каждый раз после получения ответа

• SSE – однонаправленная связь от сервера к клиенту

• Появился в HTML5

• Позволяет web-странице автоматически получать обновления с сервера (push-сообщения)

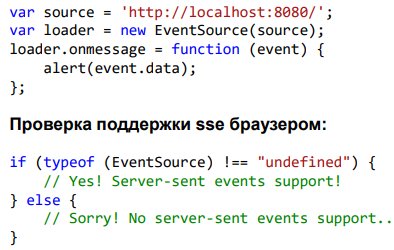
• Использует HTTP/HTTPS

• Работает как одностороння модель издатель-подписчик

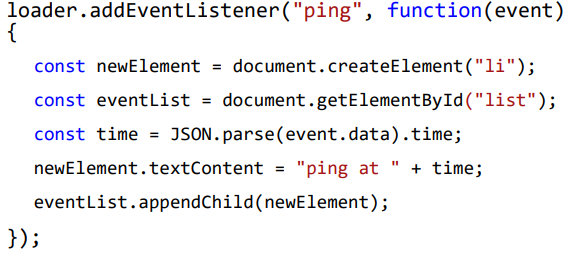
• Реализуется с помощью объекта EventSource и события onmessage

EventSource.

Подписка на события сервера происходит благодаря классу EventSource из System.Diagnostics.Tracing



Подписка на конкретное событие сервера:



SSE на сервере.

• Сервер при подписке клиента на его события получает Request с заголовком

Accept: text/exent-stream(тут будет потоковая передача текста)

• Сервер должен генерировать ответы в этом случае

- с соответствующим ContentType

- С кодом 200 Ok

- без указания размеров передаваемых данных

- не закрыть потом, использовать Flush

• Сервер держит соединение открытым!

• Сервер может генерировать многострочный текст(используя \n)

• Для завершения строки пишется \n\n

WebSockets. Схема подключения.

Если нужно сделать связь двусторонней, то используется WebSocket.

• Протокол двунаправленной связи поверх TCP

• Для установления соединения, участники по HTTP устанавливают соединение(рукопожатие – handshake)

• Поддерживается всеми современными популярными веб-браузерами

Handshake – обычный get запрос с заголовком Upgrade: websocket в ответ приходит 101 код ответа от сервера(переключение на другой протокол(на вебсокет))

Недостатки ws.

• Сложный, rfc содержит 71 страницу

• Сложность поддержки при работае с прокси

• Молчаливый отвал соединения

• Смена сети клиентом

• Повышенная нагрузка на серверы

В ASP .Net Core используется SignalR, которое позволяет писать асинхронные веб-приложения с использованием

- AJAX

- WebSockets

- SSE

(по сути является фасадом на этими технологиями)

ASP.Net Core.

• Кроссплатформенный фреймворк для создания web-приложений и API.

• Имеет открытый исходный код, доступный на GitHub

• Компактный, модульный фреймворк

• По умолчанию не устанавливается ничего (нужное докачивается через NuGet пакеты)

• Нет привязки к конкретному веб-серверу, в особенности к IIS, как было в ASP.NET

Паттерн Builder.

Краткая идея(цитирую): Собираем веб-приложение из блоков отдельных и у нас получается из таких кирпичиков созданное веб-приложение

• “Предоставляет способ создания составного объекта”

• WebHostBuilder развёртывает веб-приложение согласно этому паттерну

• Использует Fluent-синтаксис(цепочки)

• Позволяет изменять процесс конструирования, выбирать части

Класс StartUp.

• С помощью метода UseStartUp<StartUp>() можно указать какой класс будет использоваться в качестве настройки приложения

• Класс должен быть открытым

• Должны быть методы

- Configure – настройки Middleware(связующего ПО)

- ConfigureServices – настройки используемых сервисов

Middleware.

• C помощью Use’ов подключается Middleware

• Middleware асинхронно обрабатывает HttpContext и либо вызывает следующего, либо сам обрабатывает запрос

• С ASP.NET Core используется любое связующее ПО, основанное на OWIN

– Статические файлы, роутинг, аутентификация

– Можно писать своё

RazorPages.

• Более простой вариант ASP .NET Core MVC приложения

• Архитектура Model-View-Controller

* Разделение ответственности
* Модель – бизнес логика
* Представление – отображение контекста в UI, использующее шаблонизатор Razor
* Контроллер – обработка запросов, работа с моделью, выбор представления для отображения контента

• В Razor Pages модель и контроллер объединены в один cs-файл

• Представление (\*.cshtml – Razor-шаблон) тоже тесно связано с моделью

Маршрутизация.

• URL – Mapping, сопоставления пути с обработчиком

• Позволяет создавать приложения с понятными и четкими URL

* Производится автоматически, после добавления новой страницы

Связывание свойств моделей.

• Автоматическая привязка и присваивание параметров Request’а в свойства модели с помощью атрибута [BindProperty]

[BindProperty]

public string Name { get; set; }

• Валидация моделей – использование атрибутов-аннотаций для автоматической валидации данных.

Передача модели во View.

Специально ничего передавать не надо, достаточно использовать @model IndexModel внутри .cshtml скрипта

После этого к свойствам и методами модели можно обращаться @Model.Property

Обработка запросов

• В соответствии с методом OnGet/OnPost/OnPut

• Если нужно несколько обработчиков на один метод, то можно использовать параметр handler и называть методы соответствующим образом, например, OnGetUsers

Сессии и куки.

• Microsoft.AspNetCore.Session пакет NuGet

• Добавляется к сервисам AddSession()

• Регистрируется Middleware UseSession()

• Доступны в через свойство HttpContext.Session

• Cookies стандартно доступны через HttpContext.Request

**6. Базы данных.**

Модели данных

Данные в БД хранятся и обрабатываются согласно модели данных

• Концептуальной

• Логической

• Физической (специфическая для СУБД)

Классификация БД по модели:

• Навигационные (ссылки)

– Иерархические

– Сетевые

– Графовые

• Реляционные

• Объектные (объектно-ориентированные)

• Документо-ориентированные

Реляционные БД.

• 1969, Эдгар Франк Кодд

• Реляционная модель данных

• БД – связанные таблицы

• Наиболее популярная модель БД

Виды связей.

– Один-к-одному

• Студент-зачетка

– Один-ко-многим

• Группа-студенты

– Многие-ко-многим

• Предметы-преподаватели

Реализация связей.

1) Первичный ключ (Primary Key, PK)

• У каждой таблицы должен быть уникальный набор полей для однозначной идентификации записей

– Части PK не обязаны быть уникальными

• Очень часто – целочисленное поле ID

– Как правило, генерируемый автоматически

2) Внешний ключ (Foreign key, FK)

• Набор полей, по которым однозначно идентифицируется ДРУГАЯ строка в ДРУГОЙ таблице

• **Ссылка** на Primary Key или Unique-поля в другой таблице

Важные моменты в реализации БД.

• Всегда устанавливайте межтабличные связи

• Выбирайте правильным образом типы данных для столбцов – Например, ip-адрес – это не строка

• Не забывайте о Check Constraints

• (!) Не пренебрегайте индексами

Аномалии.

• Модификации – изменение данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей

• Удаления – при удалении записи из таблицы может пропасть информация, напрямую не связанная с удаляемой записью

• Добавления – невозможность помещения неполной информации, необходимость дополнительного просмотра таблицы

Нормальные формы.

• Для устранения наиболее частых (но не всех) ошибок проектирования БД (аномалий)

• Один из самых популярных вопросов на собеседованиях

• Строго формализованный научный подход

Метод нормальных форм

• Анализ полей отношения, декомпозиция отношения на несколько взаимосвязанных отношений на основе процедур нормализации

1НФ.

• В ячейке одно значение (каждое поле должно быть атомарным)

– Простое, цельное, неделимое

• Не должно быть повторений строк

• Пример нарушения:

– Список значений в ячейке

Решение – декомпозиция на несколько записей

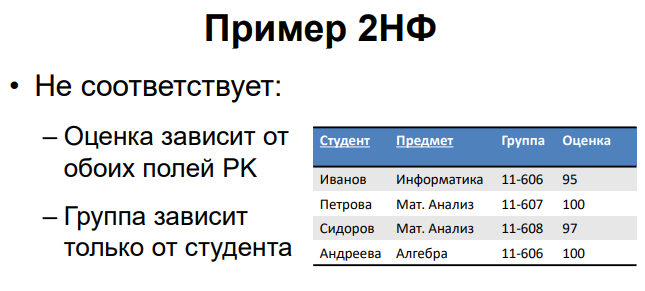


2НФ.

• 1НФ

• Каждый неключевой атрибут *неприводимо* зависит от первичного ключа Неприводимость – в составе первичного ключа отсутствует подмножество полей, от которых можно вывести данную функциональную зависимость

– Не должно быть зависимостей только от части ключа



3НФ.

• 2НФ

• Ни один неключевой атрибут не находится в транзитивной функциональной

зависимости от первичного ключа

Простым языком: если есть неключевой атрибут который зависит от другого атрибута, который в свою очередь зависит от первичного ключа, значит такое отношение можно разбить на части

Проще: Выносить в отдельные таблицы все

неключевые поля, содержимое которых может относиться к нескольким записям

таблицы





SQL.

Structured Query Language (SQL)

• Декларативный

• Основан на исчислении кортежей (см. теорема Кодда о равномощности реляционной алгебры, исчисления кортежей и SQL)

• Имеет диалекты (PL/SQL, Transact-SQL,…­)

• Data Definition Language (DDL)

create, alter, drop

• Data Manipulation Language (DML)

select, insert, update, delete

• Data Control Language (DCL)

grant, revoke

• Transaction Control Language (TCL)

commit, rollback

ADO.NET.

Работа с БД из приложения

ADO.NET – набор классов, предоставляющих службы для доступа к данным

• Для взаимодействия приложения с БД необходим провайдер данных (для разных СУБД

разные)

• Также провайдеры разные в зависимости от технологии работы с БД

ADO.NET

• Унифицированный интерфейс для работы с различными СУБД (за счёт провайдеров)

• Поддержка разных диалектов SQL

• Основные объекты:

– Connection, Command, DataReader, DataAdapter (реализуются каждым провайдером по

своему)

– DataSet, DataTable

SqlClient. Работа с параметрами. ORM. EF Core. Виды взаимодействий с БД. Классы DbContext, ModelBuilder и DbSet. Варианты настроек. Механизм миграций.

**7. MVC.**

Архитектура приложений.

• Метафора архитектуры зданий

• Высокоуровневая структура системы

– Структура, ответственности, интерфейсы, взаимодействия

– Обладающая атрибутами качества

Принципы хорошей архитектуры.

• Эффективность системы

• Гибкость системы. Откладывание ключевых решений

• Расширяемость системы. YAGNI

– Open-Closed Principle

• Масштабируемость процесса разработки

• Тестируемость (TDD)

• Возможность повторного использования

• Хорошо структурированный код. Сопровождаемость

Паттерны.

• Для удовлетворения атрибутам качества применяются архитектурные паттерны (шаблоны

– Паттерн (англ. pattern — образец, шаблон, система) — закономерная

регулярность, встречающаяся в природе и в человеческом дизайне, а также

повторяющийся шаблон,образец (возможность переиспользования)

• Некоторые из них:

– Layered pattern

• Независимые модули, взаимодействие только со смежными.

– Broker pattern

• Посредник (шина) для общения модулей

– Client-Server pattern

• Доступ большого числа клиентов к ограниченному числу ресурсов

Smart UI.

• UI содержит всю логику и работает с данными напрямую

• Логика в обработчиках событий UI

Бизнес логика.

• Enterprise-данные и правила работы с ними

• Синоним – «логика предметной области» (domain logic)

• Поведение компоненты

• Реализация модели процессов реального мира в программном виде

Model-View.

• Отделение модели предметной области от представления (UI)

– данных и действий от отображения

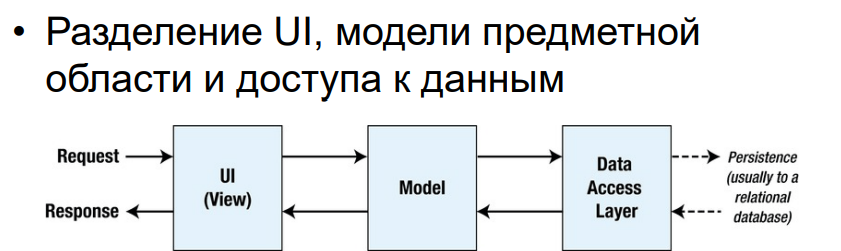
DAL.

• Слой для упрощенного доступа к данным, хранимым в хранилище, например, БД

• Дополнительный уровень абстракции

• ORM (обсуждали ранее)

Three-Tier.



MV\*.

– По сути, MV\* (MVC) – парадигма – MVC, MVP, MVVM – архитектурные паттерны

Идеи MVC:

• Разделить модель от представления (view, вид, визуальное представление)

• Модель независима ни от кого и ничего ни о ком не знает, знает только о себе

• Представление отображает модель

Роль представления.

• Активное (Active view)

– Знает о модели, самостоятельно обращается к ней за данными

• Пассивное (Passive view)

– Не имеет прямой связи с моделью, общается через посредника – Контроллера(Controller)

Контроллер.

• Самый неоднозначный компонент

• В любом случае знает о модели и может её изменять (в ответ на действия пользователя)

• Может управлять представлениями и знать о них, но это необязательно

Модель и варианты её реализации.

Неадекватность ранней реализации MVC в вебе.

MVC != доменной модели.

• Модель MVC – интерфейс и фасад для модели предметной области, которая может быть сколь угодно сложной и состоять из множества объектов

• Представление и контроллер взаимодействуют с интерфейсом и объектом-фасадом его реализующим

Интерфейс для модели (фасад).

Архитектура микросервисов. Типичные ошибки в MV\*. MVP. MVVM.

**8. Сетевое программирование.**

HTTP клиенты.

-HttpWebRequest

- WebCient

- HttpClient(Microsoft советует использовать его)

• Все из пространства имён System.Net

• Существуют много сторонних

HttpWebRequest.

• Самый старый из троицы(с .NET6 уже устаревший)

• Даёт полный контроль над объектами запроса/ответа

- Заголовками, таймаутами, куки…

• Работает с отдельным потоком

*Сложности:*

• Богатые возможности и полный контроль привядт к сложности работы с HttpWebClient

• Это может привести к ошибкам во множестве сценариев

• Лучше не работать с ним, если вам не нужны низкоуровневые возможности и полный контроль

*Использование:*

• Для отправки запроса и приёма ответа используются классы WebRequest и WebResponse

* Это абстрактные базовые классы для HttpWebRequest и HttpWebResponce соответственно

• Также WebRequest – фабрика HttpWebRequest`oв (метод CreateHttp)

HTTP-аутентификация.

WebClient.

• Простой инструмент

- более высокого уровня абстракции

• Простая обёртка над HttpWebRequest(более дружелюбный с пользовательской точки зрения)

• Упрощает решение большинства типовых задач

• Требует меньше кода

• Доступен с .NET Core 2.0

• Появилась поддержка *асинхронных методов* (раньше не было)

• Есть отчёты загрузок

• Так как базирууется на HttpWebRequest тоже стал устаревшим

HttpClient.

• System.Net.Http

• Функциональность + чистота + многопоточность (потокобезопасность)

• Один клиент – много запросов

• Рекомендован к использованию вместо HttpWebRequest и WebClient

Тонкости. TCP и UDP. System.Net.Dns и System.Net.IPAddress. Socket. Основные методы. TCP-сокеты. UDP-сокеты. TcpListener и TcpClient. UdpClient. UDP multicast. SmtpClient. FtpClient