# What we have been doing so far

* Monolithic architecture (ASP.NET MVC)
* Meaning the view is tightly coupled to the logic and anything you change in the view might heavily affect the logic or any change you did with the logic might heavily affect the view.

## Cons

* Client will have to wait for its request to be processed and also for the view that the server will eventually return
* Too much code dependency between the view and the logic
  + Makes it so any changes to the view might break your logic or vice versa

## The solution

* Decoupling the logic that process the data (your backend) from the logic that presents your data (your frontend)
* Basically, separate the two things as backend and frontend

# SOA Introduction

* Stands for Service Oriented Architecture
* A style of software design where the services are provided to the other components using some sort of a communication protocol over the network
* They separated backend and frontend and establish a form of language that allows them to communicate with each.

## Services

* They are responsible for sending and receiving data between your backend and frontend
* Usually, services are deployed as web services, so they are independent of platforms and programming language.

## SOA principles

* Standard Service Contract
  + You must have a description on what the service is about.
  + This makes it easy for client/end user to understand what the service can do.
* Loose Coupling
  + Less dependency between backend and frontend
  + So, if the service functionality changes at any point in time, it should not break the client application or stop it from working
* Service Abstraction
  + Services hide the logic they encapsulate from the outside world
* Service Reusability
  + Logic is divided into services with the intent of maximizing reuse.
* Service Autonomy
  + Should have control over the logic they encapsulate
* Service Statelessness
  + Service should be stateless
  + Service should not withhold information from one state to the other

## Pros

* Frontend and backend are decoupled
  + Meaning you can work on them separated without breaking the other one.
* It is easier to scale and expand
  + Since you can have multiple teams solely on backend or solely on frontend
* Platform independent

## Cons

* High cost since now you need another set of servers to host your services and frontend.
* To maximize the efficient of SOA, you also need separate team developing each server.

# SOAP

* Stands for Simple Object Access Protocol
* A messaging protocol specification for exchanging structured information in the implementation of web services.
* Another standardized way to communicate between computers
* It is protocol independent meaning it can be sent over HTTP, HTTPS, SMTP, etc.
* Well documented
  + You can easily figure out what endpoints you can use, what they do, what they require, etc.

## WSDL

* Stands for Web Service Description Language
* XML based file that tells the client what exactly the web service will do
* This is essentially what makes SOAP a well-documented SOA
* Think of it as the instruction manual of the web service that you get on most products you buy.

## Has 3 main components (ABCs)

* Address
  + URL address used to connect to the soap service
* Binding
  + Describes how the service is bound to the SOAP messaging protocol
  + Essentially, it gives the details require for the client and service to communicate with each other
* Contract
  + Gives what input the method takes and what output you should expect

## SOAP Messaging

* Another XML file that contains the actual information or message you get as the client using the service

## Has 4 main components

* Envelope – Identifies the xml document as a soup message
* Header – contains more information on what exactly the server is receiving from the client or vice versa
* Body – contains the response information
* Fault – contains error and if the request was successful

# Contract First

* Contract needs to be created first so that means WSDL needs to be created first without writing the actual code.

# Contract Last

* Contract will be created last so that means create the actual API first and then write a documentation detailing what it does, need, so on.

# REST

* Stands for Representational State Transfer
* Another Architectural style to design your web service

## Guiding Principles of Rest

* Uniform Interfaces
  + Your service would need to have an interface that will be defined by four constraints:
    - Identification of resources
      * Able to identity the resource you are trying to access
      * Ex: an endpoint that ends with /pokemon/pikachu will give you all the information about pikachu (from poke api)
    - Manipulation of resources through representation
      * Using appropriate action verb to do some sort of operation of your api
      * Ex: Get methods should only retrieve data from your database connected of your api
    - Self-description message
      * Any other information needed to process the data
      * Ex: format the data will come (JSON, XML, HTML, etc.)
    - Hypermedia as the engine of the application state (HATEOAS)
      * It gives other potential actions the user can take after doing one action
      * Not implemented as much now a days
* Client-Server
  + Client app will evolve separately from the server app without any dependencies on each other
  + Same concept of when we change our DL from file system to database and it didn’t break anything in our code
* Layered System
  + Constraining the interaction of your components to the ones in the next layer
  + The service should only access components it is dependent on
* Stateless
  + Server isn’t responsible for storing client state
  + It will just treat every http request as a completely different person and will not store any history of client using their api in some shape of form
  + The client is the one responsible for storing their own state
* Cacheable
  + Resource from the server can be stored
* Code on demand (optional)
  + Allows client functionality to be extended by downloading and executing code in a form of applets

# Differences between REST and SOAP

## Rest

* HTTP and HTTPS only
* Any format can be used to send information (JSON, XML, HTML, etc.)
* Very easy to setup and get it running because you don’t have to deal with the amount of documentation needed from SOAP
* Uses status code to tell if the request was successful or not

## SOAP

* Can use any protocol
* Just uses XML format to send information
* Well documented with WSDL
* Uses fault to tell if the request was successful or not

# Security

## CORS

* Stands for Cross Origin Resource Sharing
* It is just a way for you to share your resource (api that connects to your database) to other people
* It is a mechanism that checks the current origin of where that request came from and see if they have the permission to even access your resource
* The user will make a preflight request to just check if they can even make normal request. Once verified then the user will make an actual normal http request to the server

## Same Origin Policy

* Browser security prevents a web page from making a get request from a different domain than the one that is served by the web page
* What is an origin?
  + It is the url that you use to navigate in a webpage
* What counts as the same origin
  + When the protocol, host, and port (optional) is the same
    - Ex:
      * <https://google.com> and <https://google.com/search>?.... Same origin
      * <https://google.com> and <http://google.com> .. are the not the same origin
* So, the main idea is you don’t want a completely different website running some sort of javascript in their website to check your cookies and obtain crucial information and store it in their database without you knowing.

# CSRF

* Stand cross-site request forgery and it is a web security vulnerability of Same Origin that attackers use
* It basically bypasses the same origin policy by using you as the user to ask the request and you just didn’t know what you did

# Security Tokens

* Your best defense against CSRF
* They are personalizing (autogenerate) value that will be given to you as a suer so that every request will require this special token and if you don’t have it, the request will be invalid
* Nowadays every time you login to your bank account, a new token is made for you just for that session and if you notice, once you become inactive in their website, they auto logout and you have login over again and that establishes a completely new token
* So, you can see how hard it is stealing that information when it keeps dynamically changing.

## JWT

* Stands for Json Web Tokens
* It encodes json objects so you can share information between client and server freely

## Authorization vs Authentication

* Authentication
  + It is determining what exactly is the role of the user
  + You determine what you say is what you are
  + Ex:
    - Creating an account for amazon will give us a role as a customer
    - Creating an account for amazon to distribute your product will be given a role as Amazon Seller
    - Etc.
* Authorization
  + Determines what exactly does a role have access to
  + Determines what you have access to
    - Customer role can buy items but they cannot list a product they want to sell in Amazon
    - Amazon Seller role can buy items and can sell a product they want in Amazon

# Introduction to HTML

* Stands for Hypertext Markup Language
* Note that this is not a programming language but a markup language
* Unlike XML that is used to send information, HTML is used to define the structure of our webpage so that our browser will interpret that HTML file and construct that website for you to view
* It uses tags to describe the elements of your webpage

## So how does HTML read by your browser?

1. The client will request a certain webpage using the URL (address of the webpage)
2. The server that is connected to that URL will see the request and give an appropriate response
   1. http response will give information of how to display the webpage
3. The browser will use that html file to display that webpage for you
   1. Browser does not compile the html it just interprets
   2. It just gives the browser instructions to follow to display the webpage

# Semantic Elements

* They are elements that will tell you as a developer what that section in the html mean
  + Ex:
    - Table tag
    - You know whatever is put inside of that table tag will be used to create some sort of a table in your webpage
    - Another one is article and anything you put inside the article tag will be used to create an article of some sort
* They may or may not also change how it will look in the browser, that depends on the semantic element

# HTML attributes

* They are used to provide extra information that the tag can use
* All HTML elements have an attribute
* You can find what kind of attribute you can use for the tag by referencing the documentation

## Global attributes

* They are attributes that every single element has
* Ex:
  + Id – Used to uniquely identify an element
  + Class – Used to add CSS
  + Style – Used to add CSS

# Introduction to DOM

* Stands for Document Object Model
* When the browser finishes creating the website using HTML, it will create a DOM of the page
* This is how we can select one or more elements in the browser (remember about css selectors)
* It treats html elements as objects
* So essentially, we will use the DOM to get, change, add, or delete elements
  + It is the way to make our website more dynamic

## DOM Events

* Events (like C# events) will give a response whenever a certain event happens such as clicking a button, scrolling your mouse wheel, or pressing a key on your keyboard, etc.
* What we will do is tie a button event to a specific javascript function that will do something on the website

# Introduction to CSS

* Stands for Cascading Style Sheets
* Applies styling in webpage using cascading algorithm
* Essentially, CSS is the way to make your website not look like it came from the 90s

## CSS Selectors

* They allow you to select certain HTML elements
* They are great for applying multiple rules that an element should follow
* There are four main selectors
  + Element selector – it will select every element that you specify in that HTML file
  + Class selector – it will select every element that has the same class attribute
  + Id selector – It will select HTML element based on their Id
  + Pseudo selector – They will apply styling when a speciate state is met

# Different ways to include CSS

* Inline CSS
  + Applies CSS to a single element
  + It uses the style attribute to add styling to that html to apply
  + It has the highest priority
* Internal CSS
  + Applies CSS by using the style tag (usually located within the head tag)
  + Used to apply css to one or more elements
  + Second priority
* External CSS
  + Applies CSS from an external CSS file
  + You must use link tag to reference that external css file
  + Least priority

## Specificity

* The order from most specific to least specific
* This is how CSS determines which style to apply especially on rules that conflicts with each

# CSS Box model

* It is a box that wraps around every html element
* It is made up of 4 different boxes
* The order from inner to outside

1. Content
2. Padding
3. Border
4. Margin

# Bootstrap

* A CSS framework
* Has multiple already made CSS files that you can use to make beautiful websites
* They focus mobile web dev

# Introduction to JavaScript

* JavaScript has no correlation with the java language
  + They just named it the same since Java is a popular language
  + Saying JavaScript is Java is like saying a hamster is a ham.
* It is a functional language
  + We cannot use OOP pillars unless we do some backwards coding to emulate it
* Loosely typed
  + Convenient but debugging is hell
* JavaScript is both compiled and **interpreted** 
  + Back then javascript is purely interpreted language but ever since the addition of Node.js it will dynamically compile certain javascript code if it keeps being used in a website

# Datatypes

* Number
* String
* Object
* Boolean
* Null
* Undefined
* Symbol

# Scopes in JS

* The scope of a variable is defined depending on where you used it
* Function
  + Each function you create is a new scope
  + Like methods in C# you cannot use any variables made in a function in another function
* Block
  + Cannot be access from outside {}
  + Ex:

{

Let x =2;

}

//Anything outside cannot see that x variable

* Global
  + Can be access anywhere in JS
  + Var keyword will give that variable a global scope
  + Let keyword limits the scope of the variable to where it is declared
    - Advisable to just stick with let

## Other variable declaration

* Const
  + Cannot change the reference value of the variable once it is declared
* {}
  + Declaring an object you use curly brackets
  + Ex: const stephen = {name: “Stephen”, money:”$10”}
    - We use const for objects for the most part

# Different Function in JS

* Basic Function
* Callback function
  + A function that has a function in its parameter to be used.
* Arrow function
  + Very similar to what we have been doing with =>
* IIFE
  + Stands for Immediately Invoked Function Expression
  + Functions that run the moment it is defined
  + It is useful for limiting the amount of global variables you want in your files, if you expect to only use that variable only once and not reuse it again, it is best to use IIFE
  + Also useful for emulating access modifiers for variables… because JS doesn’t have access modifiers and in this way we can emulate encapsulation

# Truthy and Falsey

* In JS, all values have a Boolean equivalent to it
* Meaning you can do “hello” == 96.7 perfectly fine in JS

## What counts as false values?

* Fun0ne
* False
* Undefined
* Null
* 0 (-0 and +0 because zero can be negative or positive in JS)
* NaN (Not a number)
* Empty string

## What counts as true?

* Everything else

# How to get information from an API

## AJAX

* Stands for Asynchrnous Javascript and XML
* Web dev techniques used for updating your page dynamically after receiving the data
* Used to follow SOAP principles by using XML exclusively but now they also use JSON too
* XMLHttpRequest
  + The main object that AJAX uses to communicate with an API.
  + XML is in the name because AJAX used to do just XML

## Fetch API

* Similar Ajax in that you get info from api
* It is more inline with the REST principles
* It uses promises to achieve asynchronous operations
  + Promises represents either a completion or failure of an asynchronous operation