# 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 Definition 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

# 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 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