# 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.