FIT5032 - Internet Applications Development

Modern JavaScript Development

Prepared by - Jian Liew

Last Updated - 30th May 2018

Monash University (Caulfield Campus)

MONASH University

Outline

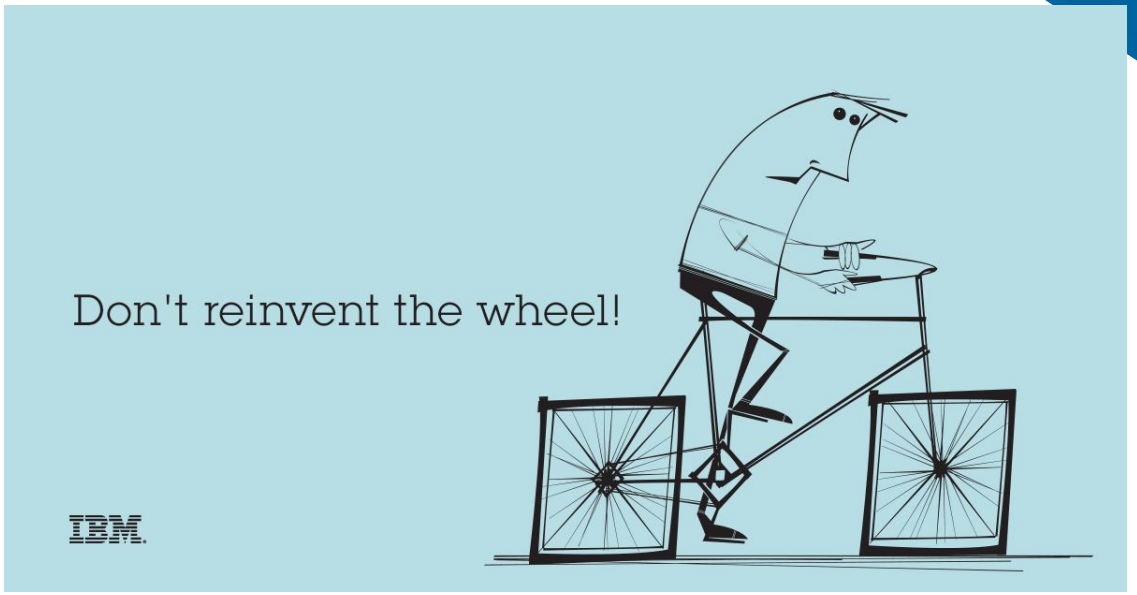
- The need for change.
- Architecture Difference
- Role of Web Server and Client
- Web API
- REST vs SOAP
- API Design

General Information

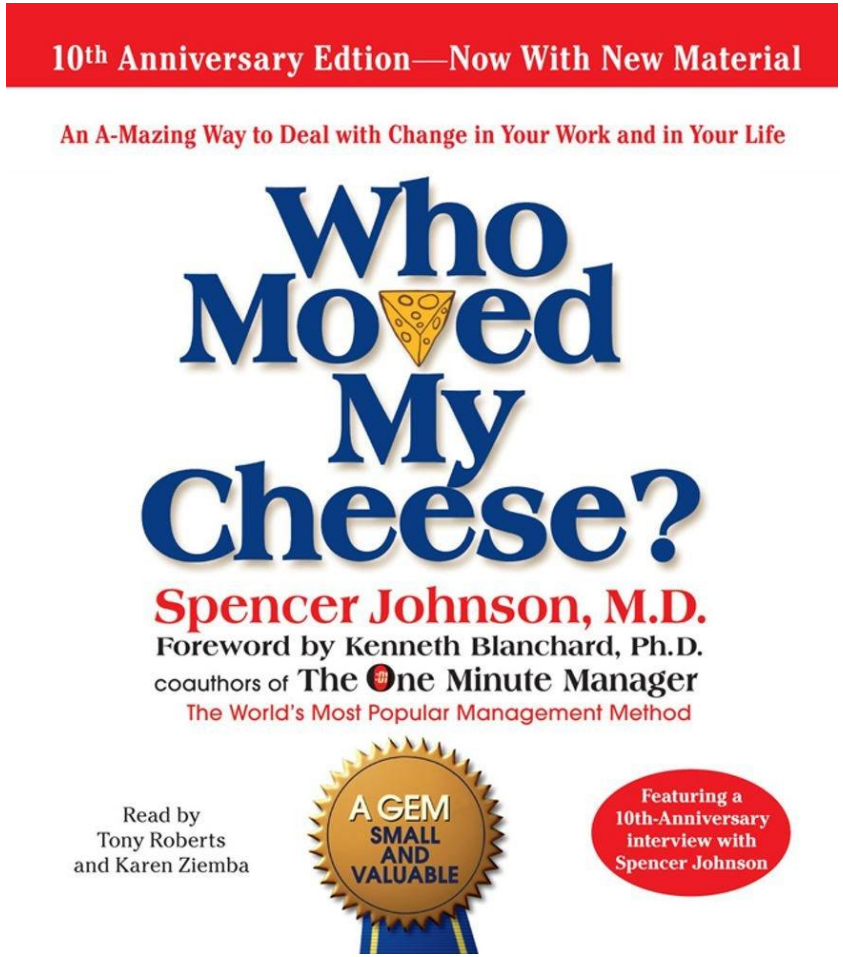
- These days, modern JavaScript development can very complicated.
- It is almost impossible to cover everything that has to do with modern JavaScript development in one lecture and one week itself.
- The objective of this week is to give **a minimalistic introduction to the world of modern JavaScript development.**

Amount of programming needed

- The amount of programming needed for this subject is actually **considerably smaller in comparison to other subjects.**
- In this subject, we will be using tools which are ready made for us to simplify the development process, but it is still important for us to understand how to properly utilise the tools.
- In other words, we will try “not reinvent the wheel”.
- However, at the end of the day everything depends on the context. This subject is designed so that we try to avoid “reinventing the wheel”. In real life, however often times there are times when reinventing the wheel is needed.
- That being said, it is important to understand the basics of what we will be using.



The Need For Change



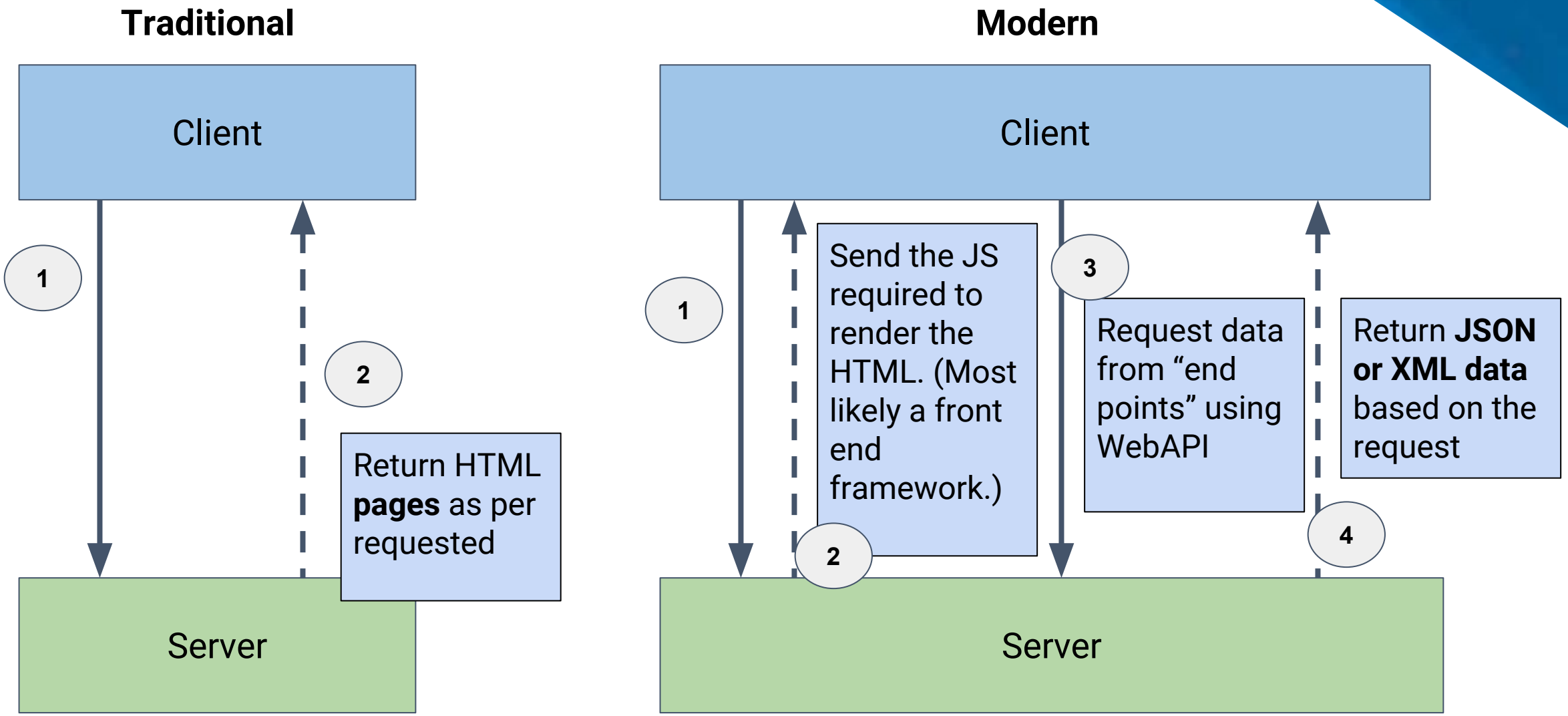
Over time, everything changes. The objective of this week's lecture is to give an introduction to **how modern web development works.**

It will attempt to give a brief introduction to how we are moving away from the monolith architecture of web development to the new “microservices” architecture. (It is due to this the way we construct web applications differs significantly especially for application that focus on enterprises and have more users” (Scalability is always a main concern)

We then then showcase, how modern JavaScript plays a role in this.

Role of Web Server and Client

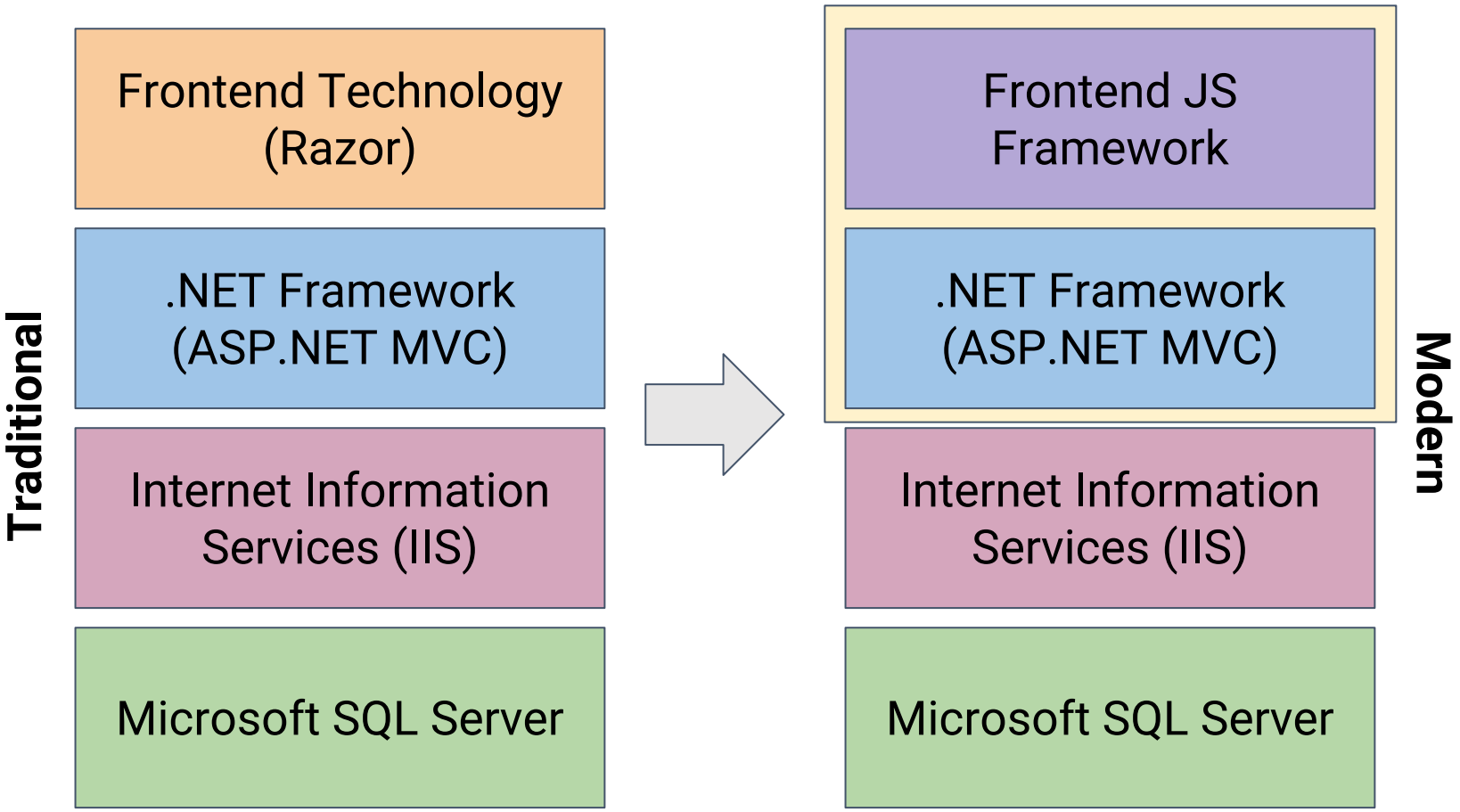
Rendering of HTML is offloaded to the client. This is done via a JS Framework.



Architecture Difference

Before we start, there is a need to understand in there is a **sheer architecture** different in the development of modern web applications.

This diagram shows an example of a modern web application architecture, it **should not be taken** as a representation of all modern web applications. In fact, this is an oversimplification.



React, AngularJS and VueJS are examples of modern front end frameworks.

Instead of returning views in the form of HTML, the server will open “end points” that will return data in the format of **JSON or XML.**

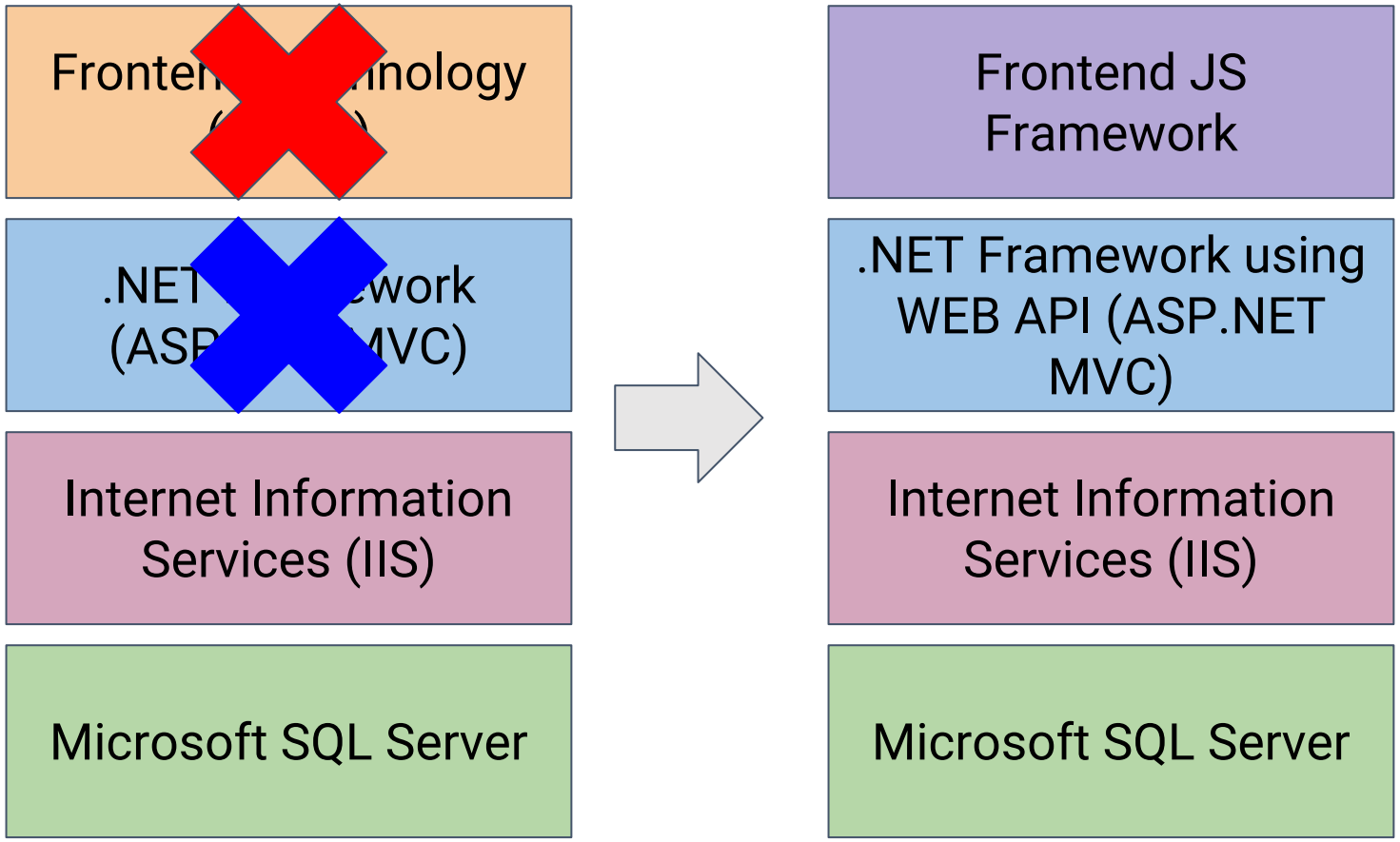
Most of the View will be rendered by the client side with data populated from the server.

This example, is specific for this subject.

Modern JavaScript Development

- Before we talk about more development using JavaScript, it is important to understand that JavaScript is now used for both front end and back end.
- Famous examples of modern JavaScript frameworks are like React, AngularJS and more recently VueJS. (These are frameworks for front-end)
- These frameworks aim to achieve Single Page Applications.
- Most if not all modern JavaScript framework heavily relies on using WEB APIs.
- This is the shift from **server side rendering of HTML pages to client side rendering.**
- **The role of the server is significantly different in comparison to traditional development.**

The difference



In order to use a modern JavaScript framework, there will be a difference in the roles that these two layers will play. (For this subject)

The objective of the server is now to create a WEB API which will allow the client request resources from it via HTTP request. (In real life the server can be created with any technology)

The front end is now a JavaScript framework like React or VueJS.

Server Side vs Client Side Rendering

The conventional method for getting your HTML up onto a screen was by using server-side rendering.

Fast-forward to today and that’s no longer the case. Websites these days are more like applications pretending to be websites. You can use them to send messages, update online information, shop, and so much more. The web is just a whole lot more advanced than it used to be.

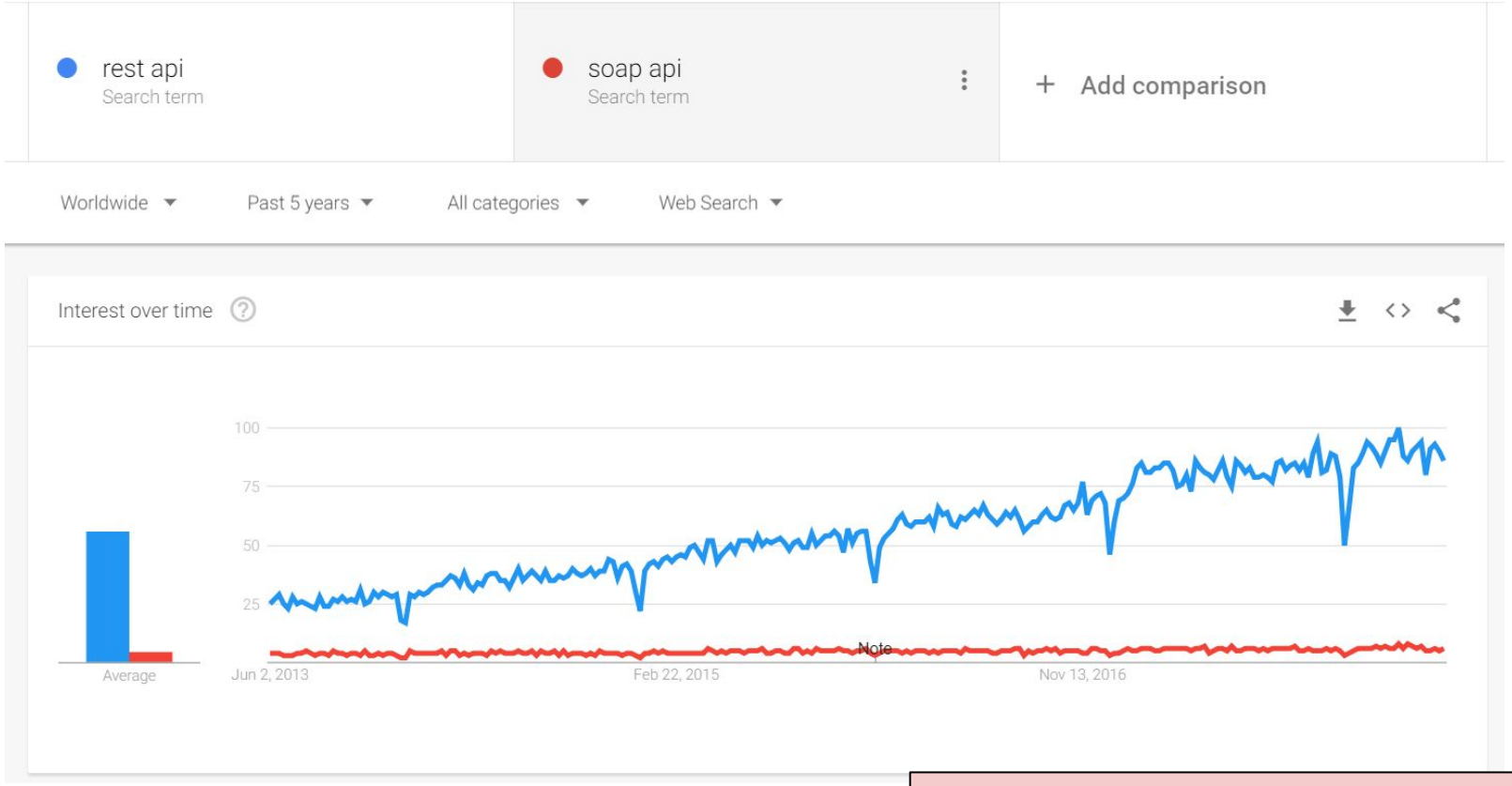
Server-side rendering is slowly beginning to take a backseat to the ever-growing method of rendering web pages on the client side.

However, with client side rendering there is a **need for a WEB API to be created.**

Web API

- A Web API is a unique type of interface where the communication takes place using the Internet and Web-specific protocols.
- Much like remote APIs make remote resources appear as local, Web APIs do the same thing for resources available on the Web.
- In fact, Web APIs started to become popular with the advent of internet services that let users store content online.
- In general, you serve Web APIs through an HTTP interface.
- The API itself **defines a set of endpoints, request messages and response structures.**
- It is a standard approach also to identify the supported response media types. **XML and JSON** are two favorite examples of response media types that can be easily interpreted by API consumers.
- While initially Web APIs were also called Web services, nowadays the use of the latter form signals that the API is RESTful, as opposed to following the SOAP standard.

REST vs SOAP



This is a Google Trend search comparing REST API vs SOAP.

It is quite clear that REST has gained significantly more interest in comparison to SOAP. **Due to this reason, we will not cover SOAP.**

SOAP is actually harder to use but a lot of companies are there still use it. It really depends on how fast things adapt.

API Design

Most modern web applications expose APIs that clients can use to interact with the application. A well-designed web API should aim to support:

- **Platform independence.** Any client should be able to call the API, regardless of how the API is implemented internally. This requires using standard protocols, and having a mechanism whereby the client and the web service can agree on the format of the data to exchange.

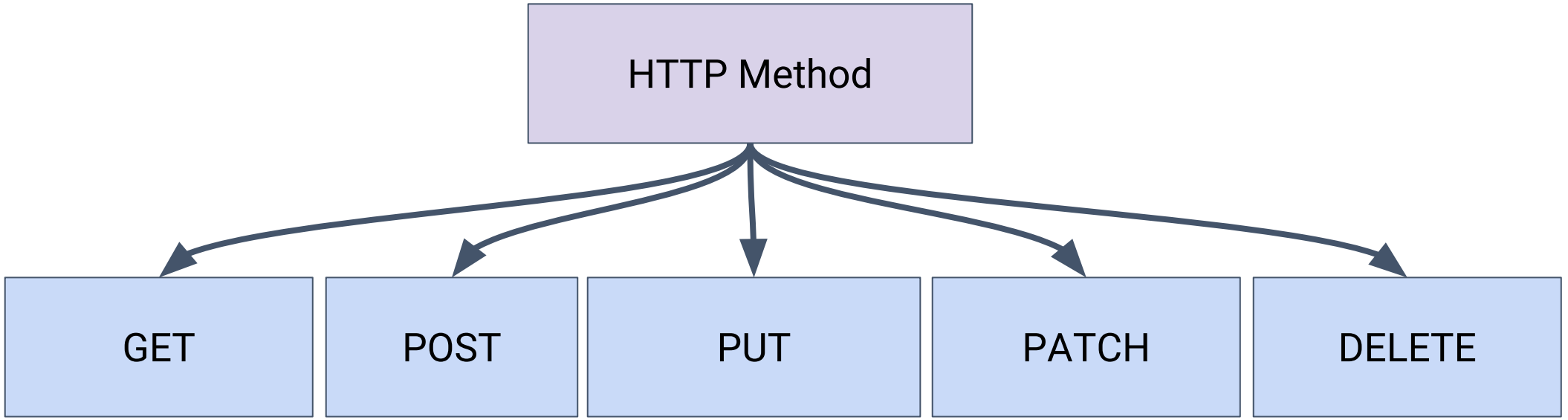
- **Service evolution**. The web API should be able to evolve and add functionality independently from client applications. As the API evolves, existing client applications should continue to function without modification. All functionality should be discoverable, so that client applications can fully utilize it.

Introduction to REST

- In 2000, Roy Fielding proposed Representational State Transfer (REST) as an architectural approach to designing web services.
- REST is an architectural style for building distributed systems based on hypermedia. REST is independent of any underlying protocol and is not necessarily tied to HTTP.
- However, most common REST implementations use HTTP as the application protocol, and this guide focuses on designing REST APIs for HTTP.
- A primary advantage of REST over HTTP is that it uses open standards, and does not bind the implementation of the API or the client applications any specific implementation.
- For example, **a REST web service could be written in ASP.NET**, and client applications can use any language or toolset that can generate HTTP requests and parse HTTP responses. (JS for example).

Operations in terms of HTTP Methods

The common HTTP methods used by most RESTful web APIs are:



There are actually more HTTP method than the ones listed here, but the ones here are the most common ones. (The ones we care about)

Common HTTP Methods used by RESTful APIs

Request	Description
`GET`	Retrieves a representation of the resource at the specified URI. The body of the response message contains the details of the requested resource.
`POST`	Creates a new resource at the specified URI. The body of the request message provides the details of the new resource. Note that POST can also be used to trigger operations that don't actually create resources.
`PUT`	Either creates or replaces the resource at the specified URI. The body of the request message specifies the resource to be created or updated
`PATCH`	Performs a partial update of a resource. The request body specifies the set of changes to apply to the resource
`DELETE`	Removes the resource at the specified URI.

Example

Resource	POST	GET	PUT	DELETE
`/customers`	Create a new customer	Retrieve all customers	Bulk update of customers	Remove all customers
`/customers/1`	Error	Retrieve the details for customer 1	Update the details of customer 1 if it exist	Remove customer 1
`/customers/1/orders`	Create a new order for customer 1	Retrieve all orders for customer 1	Bulk update of orders for customer 1	Remove all orders for customer 1

Differences Between POST, PUT & PATCH

Resource	POST	PUT	PATCH
`/customers/1`	Error	Update the details of customer 1 if it exists. (This only works if the server supports it). Depending on implementation, it might create new customer.	This can be more efficient than PATCH as it performs a partial update.

Please note that these are just examples, at the end of the day, it would depend on the implementation of the other party. Different companies, would implement these end points differently.

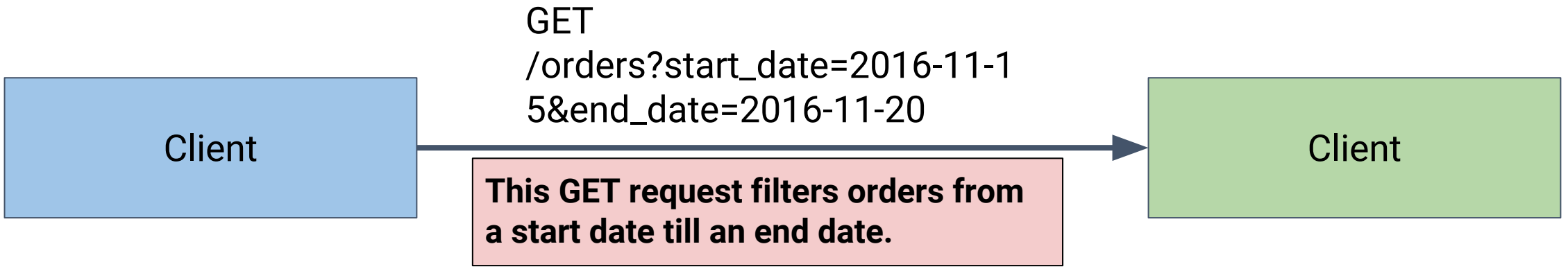
Besides that, it is important to note that PUT request must be idempotent. If a client submits the same PUT request multiple times, the results should always be the same (the same resource will be modified with the same values). POST and PATCH requests are not guaranteed to be idempotent.

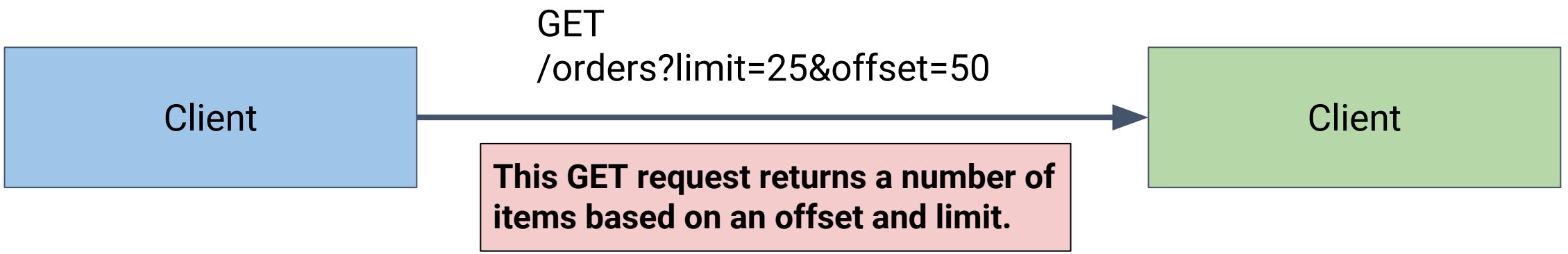
Filter & Paginate Data

- Exposing a collection of resources through a single URI can lead to applications fetching large amounts of data when only a subset of the information is required.
- For example, suppose a client application needs to find all orders with a cost over a specific value.
- It might retrieve all orders from the /orders URI and then filter these orders on the client side. Clearly this process is highly inefficient. (Due to the volume of data too)
- It wastes network bandwidth and processing power on the server hosting the web API.
- A good web API should limit the amount of data returned by any single request.

Filter & Paginate Data

A well designed RESTful API should have the ability for pagination as well as filtering.

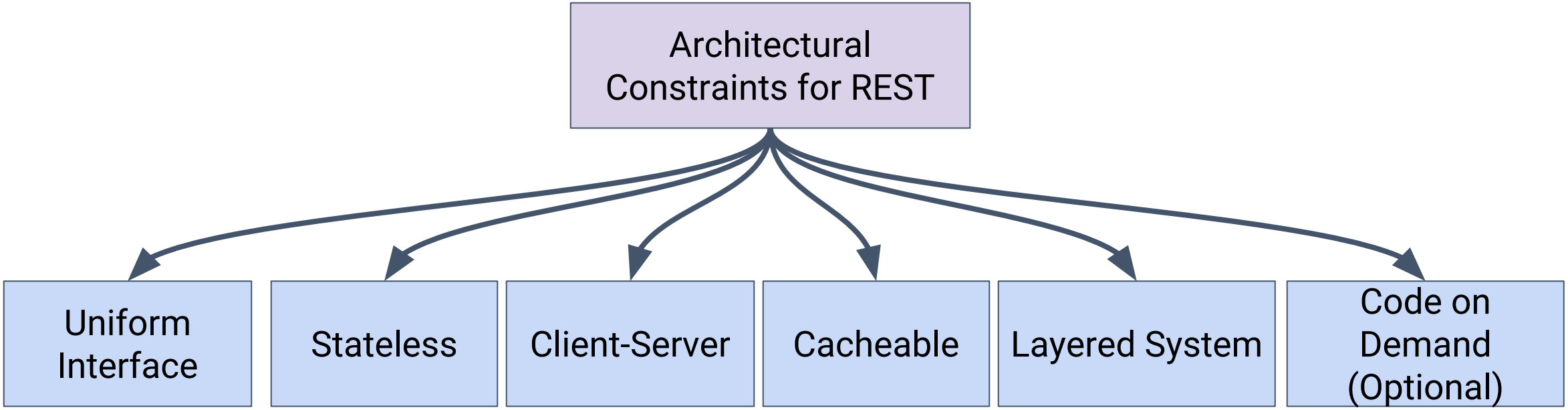




Importance of Web API

- APIs are especially important because they dictate how developers can create new apps that tap into big Web services—social networks like Facebook or Pinterest, for instance, or utilities like Google Maps or Dropbox.
- The developer of a game app, for instance, can use the Dropbox API to let users store their saved games in the Dropbox cloud instead of working out some other cloud-storage option from scratch.
- More important, by using a Web API, you can share information between companies.

Architectural Constraints for REST



Uniform Interface

- A resource in system should have only one logical URI, and then should provide way to fetch related or additional data. It’s always better to synonymise a resource with a web page.
- Any single resource should not be too large and contain each and everything in it’s representation. Whenever relevant, a resource should contain links (HATEOAS) pointing to relative URIs to fetch related information.
- Also, the resource representations across system should follow certain guidelines such as naming conventions, link formats or data format (xml or/and json).
- All resources should be accessible through a common approach such as HTTP GET, and similarly modified using consistent approach.

Client–server

- This essentially means that client application and server application MUST be able to evolve separately without any dependency on each other.
- Client should know only resource URIs and that’s all. Today, this is normal practice in web development so nothing fancy is required from your side. Keep it simple.
- Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.

Stateless

- Make all client-server interaction stateless. Server will not store anything about latest HTTP request client made.
- It will treat each and every request as new. No session, no history.
- If client application need to be a stateful application for end user, where user logs in once and do other authorized operations thereafter, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details.
- No client context shall be stored on the server between requests. Client is responsible for managing the state of application.

Cacheable

- Caching of data and responses is of utmost important wherever they are applicable/possible. The webpage you are reading here is also a cached version of HTML page.
- Caching brings performance improvement for client side, and better scope for scalability for server because load has reduced.
- In REST, caching shall be applied on resources when applicable and then these resources MUST declare themselves cacheable.
- Caching can be implemented on server or client side.
- Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

Layered System

- REST allow you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example.
- A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.

Best Practices in API Design

In general, an effective API design will have the following characteristics (SwaggerHub, 2016):

1. **Easy to read and work with**: A well designed API will be easy to work with, and its resources and associated operations can quickly be memorized by developers who work with it constantly.

2. **Hard to misuse**: Implementing and integrating with an API with good design will be a straightforward process, and writing incorrect code will be a less likely outcome. It has informative feedback, and doesn't enforce strict guidelines on the API's end consumer.

3. **Complete and concise**: Finally, a complete API will make it possible for developers to make full- fledged applications against the data you expose. Completeness happens over time usually, and most API designers and developers incrementally build on top of existing APIs. It is an ideal which every engineer or company with an API must strive towards.

Responses

- Providing good feedback to developers on how well they are using your product goes a long way in improving adoption and retention.
- Every client request and server side response is a message and, in an ideal RESTful ecosystem, these messages must be self descriptive.
- In general, there are three possible outcomes when using your API: –
  - The client application behaved erroneously (client error – 4xx response code)
  - The API behaved erroneously (server error – 5xx response code)
  - The client and API worked (success – 2xx response code)

Benefits of RESTful API

1. **Separation between the client and the server**: the REST protocol totally separates the user interface from the server and the data storage. This has some advantages when making developments. It allows the different components of the developments to be evolved independently.

2. **Visibility, reliability and scalability.** The separation between client and server has one evident advantage, and that is that each development team can scale the product without too much problem. They can migrate to other servers or make all kinds of changes in the database, provided the data from each request is sent correctly.

3. **The REST API is always independent of the type of platform or languages**: the REST API always adapts to the type of syntax or platforms being used, which gives considerable freedom when changing or testing new environments within the development

Versioning

Versioning enables a web API to indicate the features and resources that it exposes, and a client application can submit requests that are directed to a specific version of a feature or resource.

Here are the general approaches

1. **No versioning.**
2. **URI versioning.**
3. **Query String versioning.**

External Development Tools

These days most companies provide a RESTful API so that their consumers can interact with it

Here are some examples

- Facebook API
- Google API
- Twitter API
- Amazon API
- SalesForce API
- YouTube API

Some of these APIs are behind paywalls where the consumers of these API needs to be authenticated.

Modern Front End JavaScript Framework

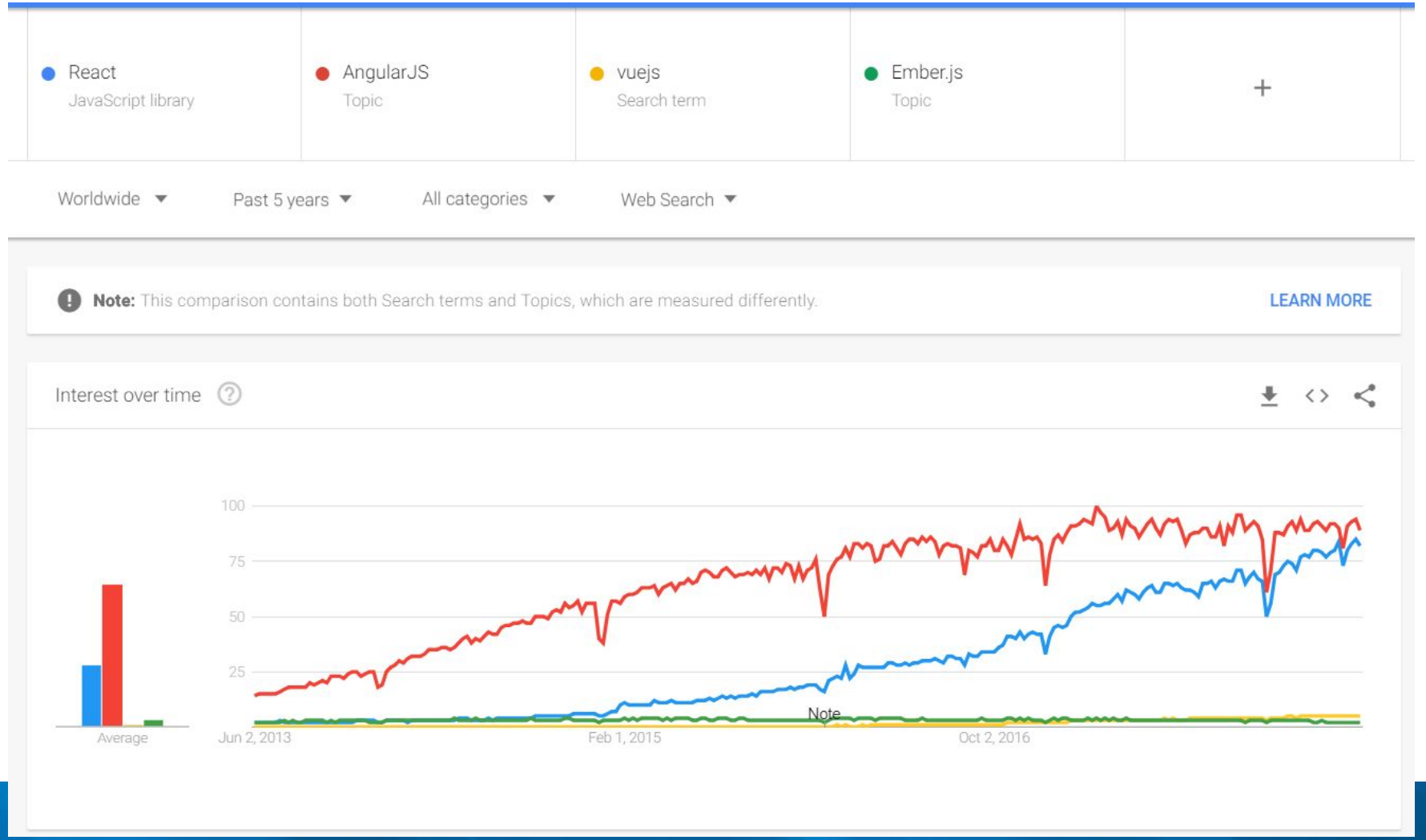
Most modern JavaScript framework are client side rendering frameworks.

Some examples of these are like

- ReactJS (Facebook)
- AngularJS (Google)
- VueJS

As mentioned, these framework heavily rely on WEB APIs provided as without a web API, there will not be data driven applications.

Framework Popularity



AngularJS

- AngularJS is a structural framework for dynamic web apps.
- It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly.
- AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write.
- And it all happens within the browser, making it an ideal partner with any server technology.

The Zen of AngularJS

- Declarative code is better than imperative when it comes to building UIs and wiring software components together, while imperative code is excellent for expressing business logic.

- It is a very good idea to decouple DOM manipulation from app logic. This dramatically improves the testability of the code.

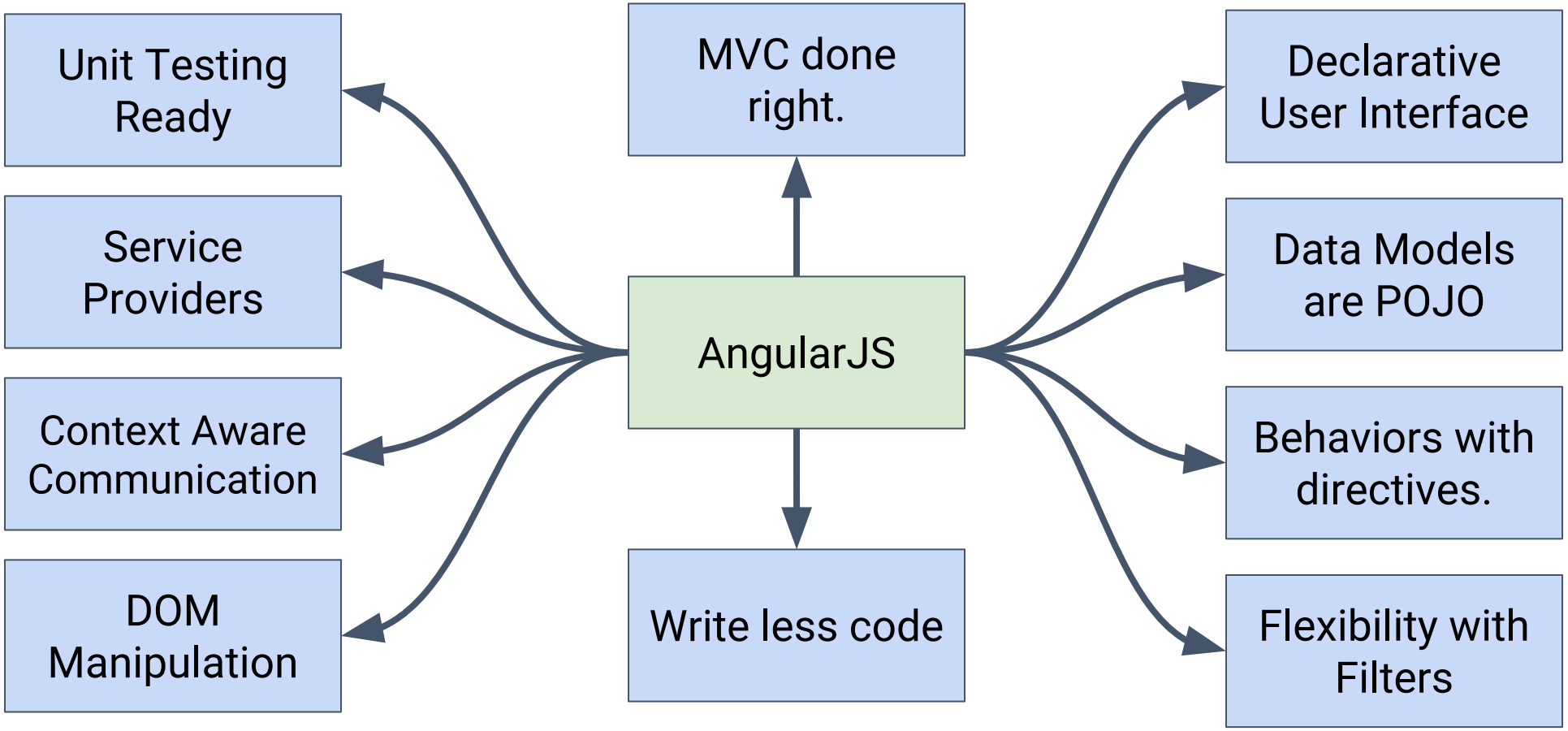
- It is a really, really good idea to regard app testing as equal in importance to app writing. Testing difficulty is dramatically affected by the way the code is structured.

- It is an excellent idea to decouple the client side of an app from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.

- It is very helpful indeed if the framework guides developers through the entire journey of building an app: From designing the UI, through writing the business logic, to testing.
  It is always good to make common tasks trivial and difficult tasks possible.

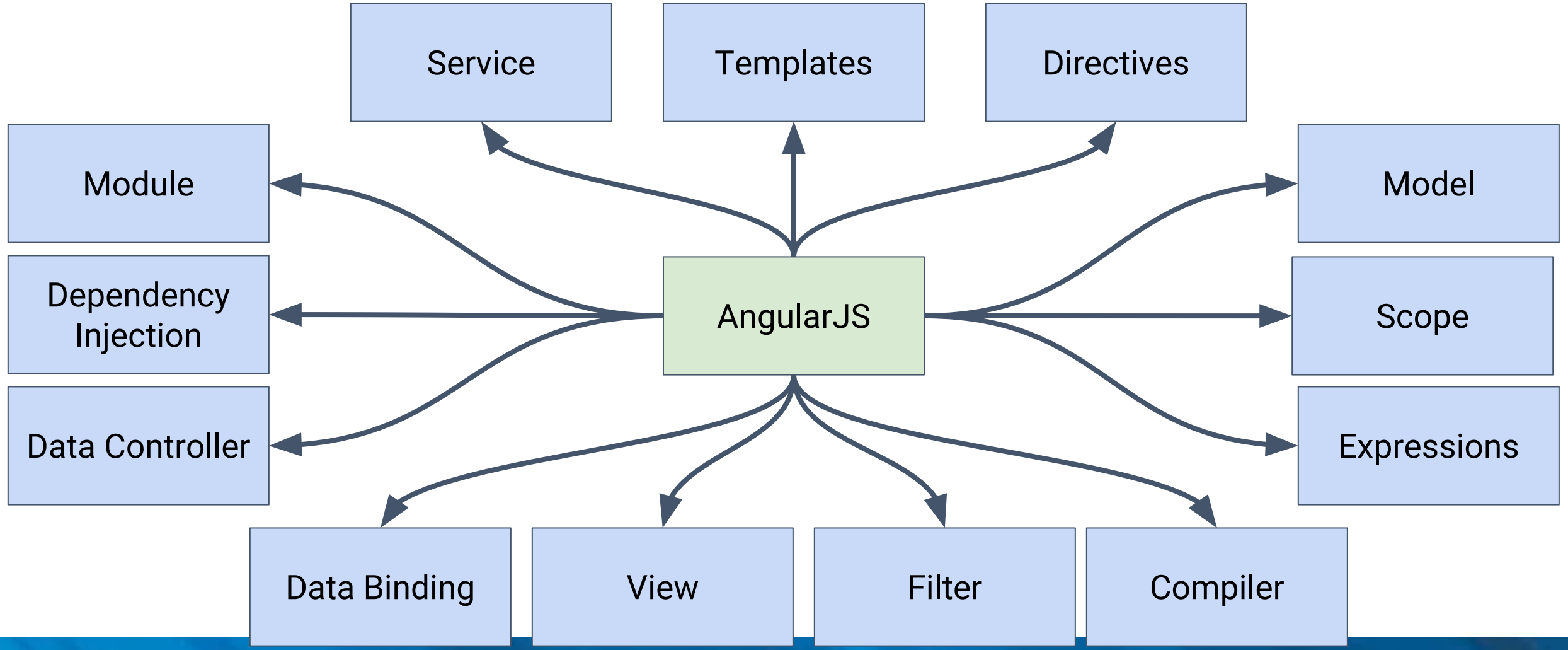
Reasons to use AngularJs

Your will learn more about AngularJS in the labs. You do not need to remember all of this.



Conceptual Overview

Your will learn more about AngularJS in the labs. You do not need to remember this.



ReactJS

Please note that there is a difference between React Native and React.

- A JavaScript library for building user interfaces.
- ReactJS coined the phrase “Learn Once, Write Anywhere”
- The main concept of React is based on reusable components.
- React components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- In comparison to AngularJS, React is vastly difference in terms of the way these components are created.
- If you like to know more about React, please read the feature article “Thinking in React”.

In the labs

You will learn how to

- Create your own WEB API using .NET MVC Framework
- Interact with your newly created API
- Learn how build a simple AngularJS application

Summary

At the end of this lecture, you would gain an understanding of

- the architecture difference between a modern web application vs the traditional
- what is a WEB API and how it is useful for development
- the proper good practices in the design of WEB APIs
- the different HTTP methods used in WEB APIs
- the general architecture of a WEB API