Introduction

Instructor: Vatanak Vong

Resources

- **Reference Material Location**
  - https://github.com/v-vong3/csulb/tree/master/cecs_491

Who am I?

- **Background**
  - Graduated with a BS in Computer Science
  - Over 9 years developing software for various industries
  - Specialize in delivering web solutions

- **Full-time**
  - Architect / Lead Engineer for a Fortune 300 company

- **Part-time**
  - Freelance Developer
  - Computer Science Lecturer

Pop Quiz

Prompt:

Create a file called test.txt that contains the phrase “Hello World”.

Create a file called test2.txt that contains the phrase “Foobar”. Copy the contents of test.txt to a new line at the end of test2.txt.

Answer:

echo “Hello World” > test.txt

echo “Foobar" > test2.txt

cat test.txt >> test2.txt

Food for Thought

How would you describe software engineering?

“Software engineering is a world of tangents”
- Vatanak Vong

Course Objectives

- **Overview of modern technologies for delivering web solutions**
- **Reinforce understanding of SDLC**
- **Experience an Agile methodology**
- **Produce a tangible “real-world” system**

***Practical skills* for an impractical world**

Class for Career

- **The course is meant to provide you insight in a career as a software developer, as such, it is fast-paced.**
- **Time won't be spent on "syntax", since they can be easily web searched. Instead, the focus of lectures will be a *layman's* approach on core web concepts and practical applications**
- **Your effort will directly correlate with how much you can apply topics taught in class to a professional setting**
- **Homework is always to review all topics discussed in lecture & lab and material for the next class meeting in addition to assignments**

Demos?

- **Pre-built demos typically results in a “missing piece to the puzzle” feeling**
- **Most demos will be shown from scratch to show and present ALL steps in the process. It’s best to take notes during the demo then practice/ask questions during lab**

App of Substance

- Register
- Login / Logout
- Application content (Requirements)

- Logging
- Error Handling
- Security
- Data store access
- UI / UX
- Documentation

Review SDLC

- What are the phases of the SDLC?
- What techniques are used when designing software?
- What are the methodologies for development?

Project Criteria

- **Registration**
- **User Management**
- **Login / Logout**
- **User Access Control**
- **Usage Analysis Dashboard**
- **Logging / Archiving**
- **Error Handling**
- **Data store access**
- **Network communication**
- **Documentation**

- UI / UX
- Application content (Requirements)

Project Deliverables

- Project Plan
- Test Plan
- BRD
- Design Doc/FRD
- Site Map
- Tech Spec

Recommended Dev Environment

- Windows PC
- Local Admin
- Install
  1. Chrome or Chrome Canary
  2. .NET Framework **4.7.1**
  3. Visual Studio Community
  4. SQL Server 2016 Developer Edition (Database Engine)
  5. SQL Server Management Studio (Database Client)
  6. Local IIS (Microsoft's Web Server)

SDLC Review

Instructor: Vatanak Vong

Main Phases

- **Value Identification**
  - Market research or end-user pain point
- **Requirements**
  - Elicitation (user stories), Analysis, High-level modeling (Abstractions)
- **Planning**
  - Project, resource, timeline planning
- **Design**
  - Low-level modeling (detailed diagrams)
- **Construction**
  - Execution of design
- **Testing**
  - Unit, functional, regression, business validation, user acceptance
- **Release**
  - Deployment, maintenance, enhancement, retirement

Main Roles

- **Client**
  - Provide the “problem”
- **Business Analysts (BA)**
  - Convert problems into quantifiable requirements
- **Project Sponsor**
  - Provides funding for project
- **Project Manager (PM)**
  - Focus on making sure the project is delivered on time, on budget and to specification
- **Development Lead / Lead Developer**
  - Liaison between business with IT; works closely with PM
  - Lead the development of solution
- **Developer**
  - Implement functionality
- **Quality Assurance (QA)**
  - Verify and validate functionality

Core Artifacts

- **Business Requirements Document (BRD)**
  - Explains client's pain points and conveys desired functionalities
  - Focus on the business needs
  - Should contain quantifiable constraints the solution needs to adhere to
  - Should outline key use cases / user stories to provide context to development team

- **Project Plan**
  - Schedule of milestones, deliverables and timelines of activities
  - Requires capacity and resource planning

- **Technical Design Document**
  - Low-level design of solution
  - Contains technologies that will be used and design decisions

- **Test Plan**
  - Details the process for verifying and validating solution
  - Enumerates both pass and fail scenarios as well as criteria for each

- **Project Road Map**
  - Details direction of solution such as future enhancements and upgrades

Methodologies

- **Waterfall**
  - Linear progression
    (Requirements, Planning, Design, Construction, Testing, Release)

- **Evolutionary**
  - Iterative progression
  - Examples: Prototyping, Spiral model

- **Agile**
  - Frequent client feedback
  - Iterative releases
  - Emphasizes a working system over process
  - Examples: Scrum, Extreme Programming, PSP/TSP

Scrum

- **Scrum Master**
  - Assigned individual to ensure developers needs are met (NOT the project manager)
- **Project Backlog**
  - Enumeration of features/functionalities/user stories
  - High level estimations (relatively or complexity)
- **Sprint Planning**
  - Break down of features into tasks
  - Task estimations (8 - 16 hours)
  - Task assignment/delegation
- **Sprint**
  - Development & QA
  - Daily stand-ups (15 minutes max) to review what was done, what will be done and road-blocks
- **Artifacts**
  - Deployable system
- **Retrospective**
  - Review sprint and ways to improve

Scrum II

- **Example Scenario**

  - John is the owner of a very popular restaurant. On most days his guests spend ~2 hours in the restaurant. John wants to improve customer turn over, but needs your help to do so.

- **Paint Points**
  - Lack of waiters
  - Lack of tables
  - Lack of chefs
  - Slow check processing

Scrum III

- **User Story**
  - Format: <Subject> <Action> <Outcome/Reason>

- **Build project backlog (backlog grooming)**

- **Sprint Planning**
  - Tasking of work items
  - Task estimation
  - Task assignment/delegation

- **Sprint Retrospective**

Modeling & Analysis

Instructor: Vatanak Vong

Glossary

- **Analysis**
  - Studying a domain to gain insight or comprehension
- **Modeling**
  - Creation of a representation of a domain

Techniques

- **Abstractions**
- **User stories**
  - <Descriptive Subject> <Action> <Outcome/Reason>
- **Diagrams**
  - Use case diagrams
  - Entity-Relation (ER) diagram
  - Workflow/activity diagrams
  - Swim-lane diagrams
  - Network diagrams
  - Site map
  - Class diagrams
  - Sequence diagrams
  - Responsibility Matrix
  - CRC Cards

Techniques II

- **High Level**
  - Workflow diagrams
  - Swim-lane diagrams
  - Site Map
  - Network diagrams
  - CRC Cards

- **Low Level**
  - Use case diagrams
  - Responsibility matrix
  - Entity-Relation (ER) diagram
  - Class diagrams
  - Sequence diagrams

References

- **UML 2.5**
  - http://www.omg.org/spec/UML/2.5/

- **UML Tools**
  - https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

Open Source

Instructor: Vatanak Vong

Disclaimer

This section contains information that should **not** be construed as legal advice. The instructor is not liable for any outcome that may arise from the student's interpretation and usage of the presented material. It is the student's responsibility to consult with certified legal representation prior to engaging in any and all activity if legal ramification is a concern.

Glossary

- **License**
  - Legal terms of use of a software and all of it's components

- **Open Source Software (OSS) / FOSS / FLOSS / Software Libre**
  - Software (source code or otherwise) with a **permissive license** that grants others the right to study, change or distribute any part of the original or altered versions for any purpose.

- **Proprietary Software / Closed Source**
  - Software (source code or otherwise) with a **restrictive license**

Glossary II

- **Public Domain**

  - Anything published where the exclusive intellectual property rights have been waived, expired, rescinded or not applicable.

- **Copyright**

  - Exclusive intellectual property rights to a specific work/entity allowing the owner to dictate terms of use and other criteria.

  - Copyright term is usually the life of the author plus X years after

  - Certain entities cannot be copyrighted such as physics, gravity, formulas, etc.

- **“Copyleft”**

  - Not a legal term

  - A concept developed by the open source community to specify that all derivative works shall share the same license as the original

Organizations

- **Apache Software Foundation**
  - Provides support for OSS projects
  - Apache HTTP Server, Lucene, Cordova
- **Linux Foundation**
  - Support open source communities with resources
  - Maintains Linux, Collaborate with Let’s Encrypt (letsencrypt.org)
- **Free Software Foundation**
  - Sponsors GNU
- **Open Source Initiative**
  - Champions OSS, defines “open source”, review & approve licenses as OSD-compliant
- **JS Foundation**
  - Promotes and supports open source JavaScript

Licenses

- **Apache License v2.0**
  - Requires copyright notice and disclaimer to be present

- **GNU General Public License (GPL) v3**
  - Requires derivative works to be free software

- **Lesser GNU General Public License v3 (LGPL)**
  - Can use free software to make proprietary software, but functionality should have an alternative available

- **MIT**
  - Requires copyright notice in proprietary work

References

- **OSS Licenses Overview**
  - https://opensource.org/licenses
- **GNU License Overview**
  - https://www.gnu.org/licenses

Web Basics

Instructor: Vatanak Vong

Glossary

- **Internet -** Public network
- **Intranet -** Private network
- **Internet hostname**
  - Human readable alias for IP address
  - 253 total ASCII characters [a-z][0-9][.][-]
- **URL**
  - Modern browsers have a limit of 2000 characters

  https://csulb-csm.symplicity.com/employers/index.php?signin_tab=0&js_disabled=0*#someID*

  Top-Level Domain - .com
  Domain - symplicity.com
  Subdomain - csulb-csm
  Scheme (protocol) - https
  Hostname - csulb-csm.symplicity.com
  Path - employers
  Resource/file - index.php
  Query string - signin_tab=0&js_disabled=0
  Hash Identifier - someID

Glossary II

- **HTTP (Hypertext Transfer Protocol)**
  - Dictates communication with servers
- **TCP/IP**
  - Dictates communication with computers in a network
- **Handshake**
  - Negotiation between two or more parties regarding how connection is established and constraints of communication
- **Web Browser / HTTP Client / Client**
  - Initiator of a HTTP request
- **Web Server / HTTP Server / Server**
  - Crafter of HTTP response
- **Front-end / Client-side**
  - Code on the client
- **Back-end / Server-side**
  - Code on the server

Glossary III

- **Ever-green**
  - Adjective for software that is auto-updated to the latest version
- **Full-Stack Developer**
  - Jack of all trades, master of none
- **Authentication / AuthN**
  - Is this person Bob?
- **Authorization / AuthR**
  - Can Bob borrow your car?
- **Session**
  - The duration of Bob using your car until he returns it to you.
- **HTTP Cookie / Web Cookie / Browser Cookie / Cookie**
  - Small piece of plain-text stored by web client connected to a domain
  - Browsers automatically attach cookies as part of every request
  - chrome://settings/content/cookies

File Extensions

- **Basic File Extensions**
  - Example.html
  - Example.js
  - Example.css

- **Proprietary File Extensions**
  - ASP.NET (C#, VB) - Example.aspx
  - Apache Structs (Java) - Example.do
  - Other - Example.php, Example.rb

Network Basics

- **IP address**
  - IPv4: 4 octets totaling 2^{32} (over 4 billion) unique addresses
    [0-255].[0-255].[0-255].[0-255]
  - IPv6: 16 octets totaling 2^{128} (over 3 sextillion) unique addresses
  - Common IPs:
    Router - 192.168.1.1
    Private Network - 192.168.x.x, 10.x.x.x, 172.16.x.x
    Localhost - 127.0.0.1

- **Port**
  - 2^{16} (0-65535)
  - First 1024 are reserved
  - Common Ports
    HTTP - 80
    HTTPS - 443
    EMAIL (SMTP) - 25
    FTP - 20 + 21

Network Basics II

- **Host file**
  - Plain-text file used by local OS to map a hostname to an IP address
    * Windows: c:\windows\system32\drivers\etc\hosts
    * MacOS: /private/etc/hosts
    * Linux: /etc/hosts

- **DNS (Domain Name System)**
  - A distributed collection of servers that's responsible for translating hostnames to IP addresses
  - Corporations can have both internal and external name servers
  - Record changes in name servers can take 4-24 hours to propagate throughout the internet

- **Firewall**
  - Hardware or software that restricts network traffic (by domain, IP, port, etc.)

Network Basics III

- **Static vs Dynamic IP Addresses**
  - Usually web servers, email servers and VPN/RDP need static IPs

- **DHCP (Dynamic Host Configuration Protocol) Server**
  - A server that either assigns an IP address to a newly added computer or reclaims an IP address from a removed computer

- **Load Balancer**
  - Hardware that routes network traffic by various criteria

- **TLS/SSL Certificate / Digital Certificate / X.509 Certificate**
  - Digital proof of a site's identity by a third party (Certificate Authority)
  - Utilizes public key cryptography mechanism to enable secure network traffic (HTTPS)

Internet Flow

1. Browser: Hey ISP, fetch me the content http://github.com
2. ISP: I don't know where that is, but let me ask my friend DNS. Hey DNS, where is github.com?
3. DNS: github.com is a nickname for 192.30.255.113
4. ISP: Gracias DNS. Hey, 192.30.255.113 on Port 80, can you send me your contents?
5. GitHub: Hey Internet stranger! I only do business on Port 443. I'll pretend you want Port 443 so let me elevate the request to HTTPS instead. You agree?
6. ISP: Hey Browser, you want HTTPS instead?
7. Browser: Yeah, whatever.
8. ISP: Do what you got to do GitHub.
9. GitHub: Thank you for your compliance. Here is the content you request.
10. ISP: 謝謝 GitHub. Hey Browser, I got the stuff.
11. Browser: Thanks ISP. Totally worth the wait. Now to render this mo…..(buffering)

HTTP Basics

- **HTTP/1.1 vs HTTP/2**
  - 1997 vs 2015
  - Improved performance through header compression and better equipped to handle asynchronous operations
  - Port 80

- **HTTP Request**
  - Header
  - Body

- **HTTP Response**
  - Header
  - Body

HTTP Basics II

- **HTTP Verbs**
  - GET
  - POST
  - HEAD
  - PUT
  - DELETE
  - OPTIONS
  - CONNECT
  - PATCH

- **HTTP Status Codes**
  - 1XX: Information
  - 2XX: Success
  - 3XX: Moved
  - 4XX: Request Error
  - 5XX: Server Error

- **HTTPS**
  - Requires TLS/SSL Certificate on the web server/network appliance
  - The body and header of the request/response is encrypted. Generally, the URL remain unencrypted.

Types
Of
Web Solutions

Instructor: Vatanak Vong

Categories

- **Website**
- **Web Application (Web App)**
- **Web Service**
- **Webhook**

Website

- **Definition:**

  One or many web pages accessible through a network (public or private)

- **Purpose:**

  To publish resources (images, documents, videos, etc.); typically static content

- **Technologies:**

  1. HTML
  2. CSS
  3. JavaScript
  4. Web Server

Web Application

- **Definition:**

  Software that adheres to the **Client-Server** architecture in which the "Client" is a web browser.

- **Purpose:**

  To provide a solution to a need; typically dynamic content

- **Technologies:**

  1. HTML
  2. CSS
  3. JavaScript
  4. Server-side Stack (.NET, Rails, PHP, etc.)
  5. Web Server

Types of Web Application

- **“Classic” Web Application**
  - Majority of content is rendered on the server then served to the client
  - Application consists of multiple web pages
  - New content requires full page loads aka the browser “flickers”
  - Form submissions mostly consists of **postbacks**

- **Single Page Application (SPA)**
  - Consist of only a single web page (the initial page load is the only full page load experienced by user)
  - URL routing
  - **AJAX** calls delivers content

- **Progressive Web Application (PWA)**
  - Works offline (HTML 5)
  - Add additional “Apps” without an App Store
  - “Delivering an installed app experience”

Web Service

- **Definition:**

  Network accessible APIs that adheres to the Request-Response architecture

- **Purpose:**

  Provide distributed functionality

- **Technologies:**

  1. Server-side Stack (.NET, Rails, PHP, etc.)
  2. Web Server

Webhook

- **Definition:**

  HTTP **callback** (trigger)

- **Purpose:**

  Provide PubSub mechanism for web solutions

- **Technologies:**

  1. Server-side Stack (.NET, Rails, PHP, etc.)
  2. Web Server

Coding Standard

Instructor: Vatanak Vong

Naming Convention

- **Classes / Interfaces**
  - Descriptive nouns

```
class Teacher { }
Interface Lecturer { }
Teacher scienceTeacher = new Teacher();
```

- **Methods**
  - Descriptive verbs; First character should be capitalized unless conflicts with standards of language

```
void UpdateAge(Person person, int age);
```

- **Variables**
  - Descriptive nouns/state so that the data type can be inferred
  - Only variables meant to keep track of iteration can be a single letter, but preferably should not be the case if more than two counters are required

```
var age = 0;
var i = 0;
var _privateData = new Data();
```

- **Collections**
  - All collections or data structures that contains multiple entries should be in plural form as much as possible

```
var apples = new Apple[10];
```

- **Unit Tests**
  - ClassName_Method_Scenario

```
ShoppingCart_Checkout_SuccessfulPayment
```

Coding Style

- **Casing**
  - PascalCase : UpdateAge
  - camelCase: updateAge
  - kebab-case: update-age
  - Hungarian notation

```
Person tempPerson = new Person();
int intCounter = 0;
```

- **JavaScript**
  - All beginning braces should be on the same line as the previous statement
  - All single expression code blocks (e.g. if, while, for, etc.) should have beginning and ending braces
  - Always include a semicolon at the end of an expression or statement
  - Classes should be PascalCase
  - Functions/Methods should be camelCase
  - Variables should be camelCase
  - Single line comments start with a space then a capitalized letter
  - Commented-out code does not have a space in between the code and the comment operator

- **.NET**
  - All beginning braces should be on it's own line
  - All single expression code blocks (e.g. if, while, for, etc.) should have beginning and ending braces
  - Classes/Interfaces should be PascalCase
  - Functions/Methods should be PascalCase
  - Variables should be camelCase
  - Single line comments start with a space then a capitalized letter
  - Commented-out code does not have a space in between the code and the comment operator

Coding Style II

- **Developer Notes/Comments**

  - Use common “tags” for adding developer notes to your source code

    1. // TODO: reason
    2. // HACK: reason
    3. // REVIEW: reason
    4. // IMPORTANT: reason
    5. // SECURITY: reason

Web Ecosystem

Instructor: Vatanak Vong

References

- **HTML**
  - https://developer.mozilla.org/en-US/docs/Web/HTML
  - https://www.w3.org/standards/webdesign/htmlcss.html
  - https://validator.w3.org/

- **JavaScript**
  - PPK on JavaScript (https://www.quirksmode.org/js/contents.html)
  - eloquentjavascript.net
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript

- **Cascading Stylesheets (CSS)**
  - https://developer.mozilla.org/en-US/docs/Web/CSS
  - https://css-tricks.com/guides/beginner/

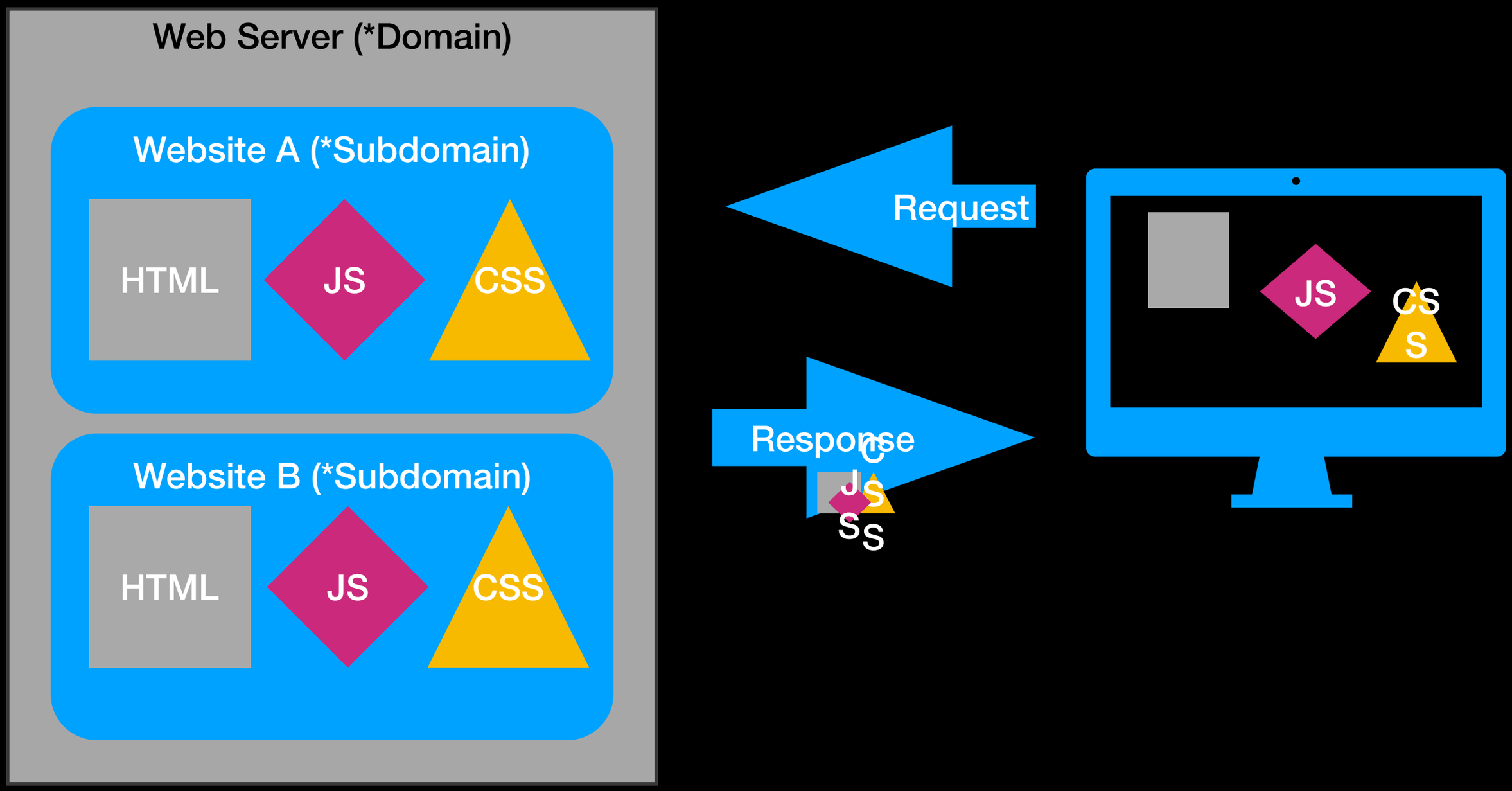
Fundamental Topics

- **Browser**
- **Web Server**
- **Hypertext Markup Language (HTML)**
- **Cascading Stylesheets (CSS)**
- **ECMAScript (JavaScript)**

Expanded Topics

- **Server-side Stack**
- **JavaScript Libraries/Frameworks**
- **Tooling / Tool chain**
  - Builds
  - Transpilers
  - Preprocessors
  - Minifiers/Uglifiers
  - Optimizers

Big Picture



*** The actual host configuration depends on the network topology/security structure**

Separation of Concerns

- **HTML**
  - Drives the structure of the content
- **CSS**
  - Drives the presentation of the content
- **JavaScript**
  - Drives the behavior of the content

HTML Bascis

Instructor: Vatanak Vong

HTML

- **What is it?**

  - “Loose” mark up language for describing structure of documents with links to other documents or more simply the format for a web page

  - .html or .htm

  - Most web servers will automatically serve up a web page named default.html/index.html if found in a directory

- **Components**

  - Elements/tags

    <foo></foo>

    <foo />

  - Attributes/properties

    <foo bar=“rar”></foo>

    <foo bar=“rar” />

- **Basic HTML Structure**

```
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

HTML II

- **HTML 4.01 (Strict, Transitional, Frameset) vs HTML 5**

  - 1998 vs 2014

  - HTML 5 added new elements like <canvas>, <video>, etc. that better aligns with mobile devices and modern applications

- **Basic HTML 5 Structure**

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

HTML III

- **Important Elements**
  - <head>: Resources to load before the content
  - <body>: The content
  - <script>: Inline or external JavaScript
  - : Container
  - <style>: Inline CSS
  - <input>: User data entry

- **Not so important element**
  - <hr />, <b>, <table>, <frame>, <frameset>

HTML IV

- **HTML Entities**

  - Special notation for characters that are difficult to type with a keyboard or values that you don't want to be interpreted as HTML code

    < = <

    & = &

    © = ©

    @ = @

    Δ = Δ

- **Events**

  - Event Types

    * Mouse, Keyboard, Form, Media, etc.

```
<button onclick="alert('Hi')">
    Say Hi
</button>
```

  - Common Events

    * onclick, onblur, onsubmit, onkeydown, onload, onunload, onmouseover

  - Event Propagation Models

    * Event Bubbling - starts from the target element and goes up the ancestor tree (default model)

    * Event Capturing - starts at the ancestor root element and goes down the children tree

    * Both are part of the W3C specification

Browser Basics

Instructor: Vatanak Vong

Browser

- **Key components**
  - HTML Parser
  - Rendering/Layout Engine
  - JavaScript Engine
- **Main Web Engines**
  - WebKit: Chrome, Safari, Opera
  - Chakra: Firefox
  - Trident: Internet Explorer

Browser II

- **Original Intent**
  - Platform to serve static documents with hyperlinks
- **Current Intent**
  - Platform for providing rich, iterative and complex applications
- **Browser Wars**
  - Competition for market dominance resulted in browser vendors being very lax on malformed HTML and non-standard JavaScript
  - Race for what browser can render the worst HTML and execute the worst JS

Browser III

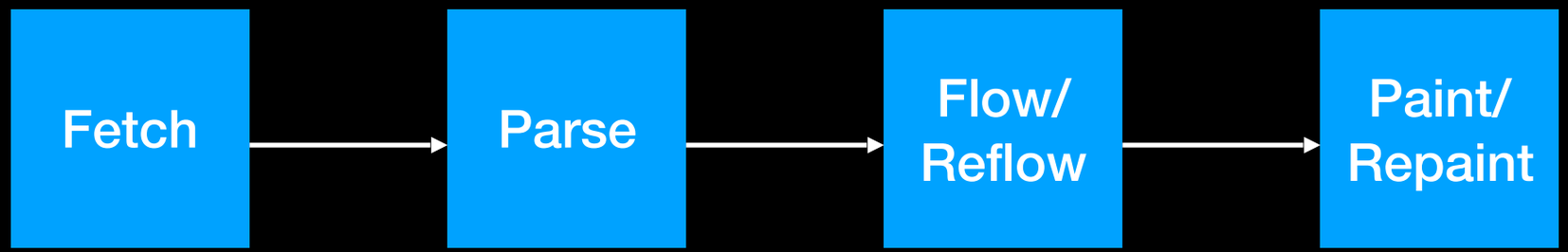
- **Browser Object Model (BOM)**
  - The APIs that allows access to the browser window, cookies and other windows.

- **Document Object Model (DOM)**
  - Internal representation of the HTML document's structure
  - In JavaScript, it is the **document** object
  - Programmatically alter a web page by manipulating the DOM

- **DOM Levels**
  - 0 (Netscape 3 Standard): All browsers support; limited HTML element access
  - 1 (W3C 1998 Standard): Access to manipulate all elements
  - 2 (W3C 2000 Standard): Updated interfaces, CSS support
  - 3 (W3C 2004 Standard): XML Support, Keyboard events
  - 4 (W3C 2015 Standard): WIP

Browser IV

- **How Browsers Work**



- Fetch: Get resources

- Parse: Generates the DOM and render tree

- Flow/Reflow: Use render tree/DOM to calculate exact screen position.

- Paint/Repaint: Draw content to the screen by traversing render tree/DOM

Browser V

- **HTML Render Flow**

1. **Browser Sniffing**
2. **HTML Parsing**
3. **HTML Head Processing**
   A. **Resource loading**
   B. **Resource execution**
4. **HTML Body Processing**
   C. **Resource loading**
   D. **Resource execution**
   E. **Painting/Reflow**

Browser VI

- **Browser Connection Pools**

  - All browsers have a limited number of concurrent requests that can be made to a **single hostname** (subdomains don't count).

  - Most modern browser have a limit of 6 while older browsers were capped at 2

CSS Basics

Instructor: Vatanak Vong

CSS

- **What is it?**
  - Language for styling and altering the layout of web pages
  - Define rules that contains a set of styling

- **Types of Selectors**
  - ID
  - Type/Tag
  - Universal
  - Attribute

- **Components**
  - Selectors: HTML element to target
  - Properties: Attributes you want to apply
  - Values: integers, float, strings
  - Units: pt, px, em, %
  - Pseudo-classes: Style element based on state (div:hover, input:checked)
  - Pseudo-elements: Allows you to target a specific part of an element (a::after,)

CSS II

```
 /* Element with ID=test */
#test {
  color: blue;
}

/*
All elements with “warning”
in class attribute
*/
.warning {
  font-weight: bold;
  background-color: orange;
}

 /* All DIV elements */
div {
  background-color: red;
}
```

```
/* All elements */
*  {
      font-style: italic;
}

/* All links that ends in .edu */
a[href$=“.edu”]:hover {
   color: green;
}

/* Listing */
body, p, div#test, img.warning { }
```

CSS III

- **Selector Performance**
    1. ID
    2. ID with tag qualifier (div#test)
    3. Class
    4. Class with tag qualifier (li.current)
    5. Type/Tag
    6. Multi-tag (ul li a)
    7. Attribute (a[title=“home”], #content[title=“home”])
    8. Universal (* {})

CSS IV

- **Combinators**

```
div img {} /* Descendant (most expensive) */

div > img {} /* Child (one level)*/

img + a { } /* Adjacent sibling (immediately after) */

div ~ img { }  /* General sibling */
```

CSS V

- **The Box Model**
  - HTML elements are represented as rectangle boxes
    * Areas
      - Content
      - Padding
      - Border
      - Margin

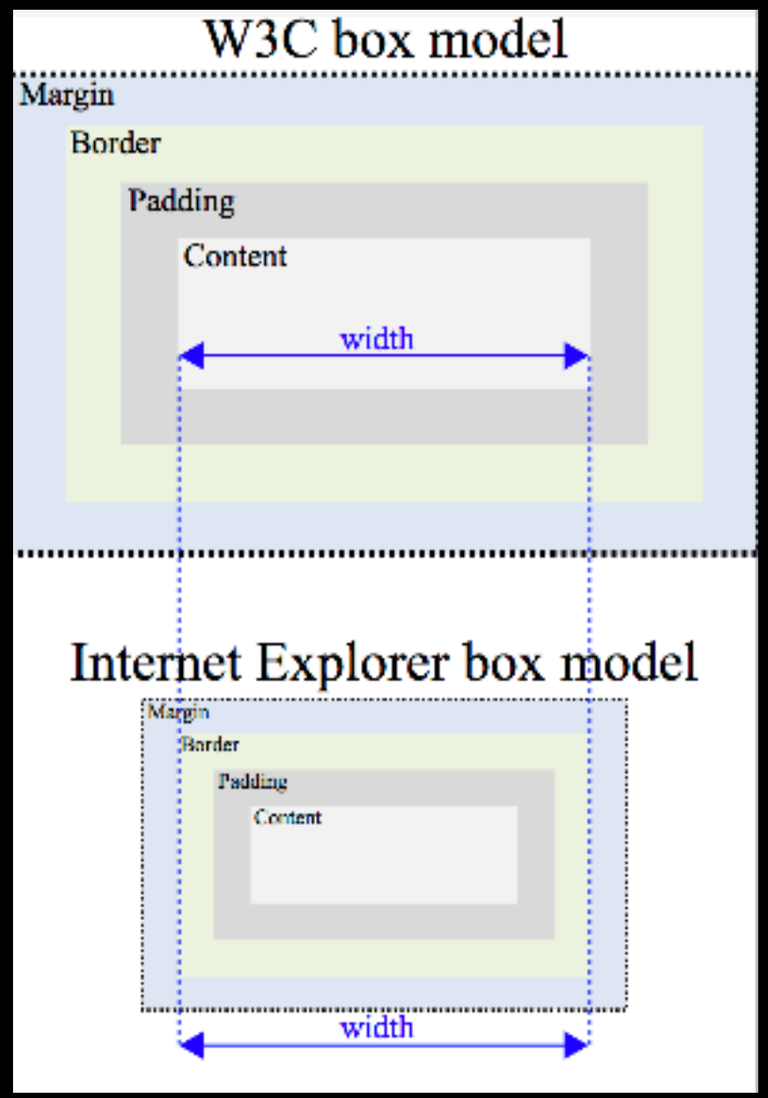


*Image from
https://commons.wikimedia.org/wiki/File:W3C_and_Internet_Explorer_box_models.svg

CSS VI

- **Less**
  - lesscss.org
- **Syntactically Awesome Stylesheet (Sass/SCSS)**
  - sass-lang.com
- **Benefits**
  - Variables (@themeColor, $themeColor)
  - Nesting
  - Operators
  - Mixins

Fundamental JavaScript

Instructor: Vatanak Vong

Version History

Year	ECMAScript Version	Details
1997	ECMAScript 1	
1998	ECMAScript 2	
1999	ECMAScript 3	RegEx Try/Catch
2009	ECMASscript 5	Full DOM support Strict mode JSON support
2011	ECMAScript 5.1	
2015	ECMAScript 6	Template literals Classes Modules Others*
2016	ECMAScript 7	Array.prototype.includes ** operator (exponential)

Same Origin Policy

- **What is it?**

  - Thou shall not execute scripts from another domain

  - Default browser security policy to protect against malicious client-side code

- **Workarounds**

  - Set document.domain of page to match origin of script

  - JSON Padding (JSONP) callback

  - Use an iFrame to load a page with malicious scripts that auto-executes

Warning

With few exceptions, ALL properties of all objects can be overridden by any code (local or external)

Core

- **Declarations**
  - var a, b, c;
  - var a = 1, b = 2, c; // Inferred and mutable
- **Operators**
  - +, -, *, /, %, ++, - -, **
  - &&, ||, ==, <, >, >=, <=, ?, !=
  - <<, >>, <<<, >>>, &, |
  - ===, !===
- **Comments**
  - //
  - /* */

Core II

- **Primitives**
  - Number:
    var age = 10;
    var price = 2.99;
  - String:
    var first = “Bob”;
    var last = ‘Smith’;
  - Object
    var bob = Object.create();
    var person = { }; // Object literal
  - Array:
    var grades = [‘A’, ‘B’, ‘C’];
    var data = [10, 11.3, “Apple”];
  - boolean: true/false
  - null
  - undefined, NaN (arithmetic with undefined)

Core III

- **Loops**
  - for
  - for..in
  - while
  - do..while

- **Automatic Semicolon Insertion**

```
function test() {
    return
      5;
}
```

Core IV

- **Functions**

```
- function add(a, b) { // function declaration
    return a + b;
}
add(1, 2); // 3
```

```
- var add = function (a, b) { // function expression, anonymous function
    return a + b;
};
add(1, 2); // 3

- var add = (a, b) => { return a + b; } // arrow function
add(1, 2); // 3
```

Closure

- **Function Scope**
  - High-level C-based languages have bracket-level scoping
  - JavaScript uses function-level scoping
  - Functions executes under the scope in which they were defined in

```
if (true)
{
    var name = “Bob”;
}
name = “John”;  // Works in JS, fail in Java/C/C#

function test (index) {
    var storage = [ ];
    for (var i = 0; i < 10; i++) {
        storage[i] = function ( ) {
            console.log(i);
        };
    }
    storage[index]( );
} // Output for test(5); ?
```

Closure II

- ***this* keyword**
  - Reference to the containing (top-level) function
  - In a browser, the global object is **window**
  - In nodeJS, it is **undefined** or the nodeJS module
  - In an event handler, it is the HTML element

```
function test(obj) {
    console.log(obj);
    console.log(this);
}

<div onclick=“test” />
<div onclick=“test()” />
<div onclick=“test(this)” />
```

Global vs Local

```
var a = 10;

function test() {
  var b = 25;
}

function quiz() {
  c = 22;
}

function iGiveUp() {
  var a = 5;
}
```

Global vs Local

```
var a = 10; // Global

function test() {
  var b = 25; // Local
}
```

```
function quiz() {
  c = 22; // Global
}

function iGiveUp() {
  var a = 5; // Local - variable masking
}
```

JSON

- **JavaScript Object Notation**
  - Officially recognized format for JavaScript objects
  - http://www.json.org/

```
{
  “Name”: “Bob”
  “age”: 10,
  “friends”: [
      { “Name”: “Bob”, “age”: 10.5, friends: [“Bob”] },
      { “Name”: “Peter”, “age”: 9, friends: [“Bob”] },
  ]
}
```

Hoisting

```
d = 2;
add();

function add() {  // function hoisting
  return c + d;
}

var realAdd = function(c) {
  return c + d;
};

var d; // Variable hoisting
var c = 1;
```

Hoisting

```
d = 2;
add();

function add() {  // function hoisting
  return c + d;
}

var realAdd = function(c) {
  return c + d;
};

var d; // variable hoisting
var c = 1;
```

Quirks

var person = { age: 10 };
person.age
person[“age”]

10 == 10?
10 == “10”?
10 === 10?
10 === “10”?

var a = “a” + [1]
var error = ‘You need $’ + 1 + 55 + ‘ dollars’

Quirks

```
var person = { age: 10 };
person.age // 10
person[“age”]  // 10
```

```
10 == 10? // true
10 == “10”? // true
10 === 10? // true
10 === “10”? // false

var a = “a” + [1]? // “a1”
var msg = ‘Need $’ + 1 + 55 + ‘ dollars’ // “Need $155 dollars”
```

Intermediate JavaScript

Instructor: Vatanak Vong

OO JavaScript

- **Constructor**
  - Object.constructor
    * All objects inherit a **constructor** property
    * Makes it possible to create instances of objects with the same properties and methods by using the **new** operator (“use strict” to prevent calls without the new keyword)
    * Only use the **instanceof** operator to check type instead of constructor property since it can be overridden

- **Inheritance**
  - JavaScript uses prototypical inheritance instead of classical inheritance
  - Object.prototype
    * Define members that are shared across all instances (saves memory)

- **Prototype Chain**
  - Member lookup first happens on the “current” object level
  - Chained search ends when member is found or **null** is reached

Callbacks

```
function longRunningCode ( msg, callback ) {

    // Code to simulate long running code
    var pattern = /A(B|C+)*D?/;
    pattern.test("ACCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC");

    // Useful for executing code exactly after long running code
    // or allowing custom code executing in a framework/library
    callback( );
  }

longRunningCode ( 'A test message',  function () {
  console.log('Callback running');
});
```

Currying

```
function test(msg) {

    . . .
}

test(“a”)(“b”);
```

Currying

```
function test(msg) {
    console.log(msg);

    return function (anotherMsg) {
        console.log(anotherMsg);
    };
}

test(“a”)(“b”);
```

Chaining

```
function Car( ) {  }
Car.prototype.drive = function ( ) {

}
Car.prototype.brake = function ( ) {
}

var myCar = new Car( );

myCar.drive( ).brake( ).drive( ).drive( ).brake( );
```

Chaining

```
function Car( ) {  }
Car.prototype.drive = function ( ) {
    console.log(“Driving”)

    return this;
}
Car.prototype.brake = function ( ) {
    console.log(“Braking”)

    return this;
}

var myCar = new Car( );

myCar.drive( ).brake( ).drive( ).drive( ).brake( );
```

~~Advance~~
~~JavaScript~~
TypeScript

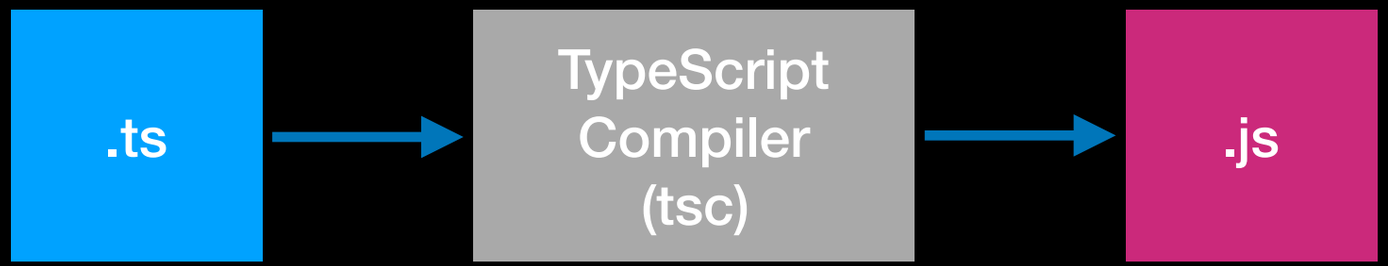
Instructor: Vatanak Vong

Why

- **Reduces common JS errors**
  - Type system
    * Parameter types
    * return types
  - namespaces and modules
  - “strict” mode by default
  - Built-in mechanism for API documentation
    * Type declaration file

- **Object-oriented**
  - Easier to compose objects
    * Java/C# style classes & interfaces
  - Proper data encapsulation (private/protect/public)

Transpilation



Type Annotation

- **JavaScript**

```
var firstName = “Johnny”;
let lastName = “Appleseed”;

function toSchoolFormat(first, last) {
    return `$(last),  ${first}`;  // ES2015 template string
}
```

- **TypeScript**

```
let firstName: string = “Johnny”;
let lastName = “Appleseed”;

function toSchoolFormat(first: string, last: string): string {
    return `$(last),  ${first}`;  // ES2015 template string
}
```

Types

- boolean
- number
- string
- array
- Symbol
- enum
- void
- null
- undefined
- never - return never;
- any

Interface & Classes

- **Interface**

```
interface Person {
    name: string;
    age: number;

    // Optional
    jobTitle?: string;
}

let bob = {
  name: “Bob",
  age: 25,
  jobTitle: “Teacher”,
  Salary: 40000
};

let teacher: Person = bob;
```

- **Class**

```
class Teacher {
    // public by default
    readonly name: string;
    private age: number;
    protected title: string;

    constructor(person: Person) {
        this.name = person.name;
        this.age = person.age;
        this.title = “Teacher”;
    }
}

let bob: Teacher = new Teacher( {…} );
```

Deployment

Instructor: Vatanak Vong

Types of Web Hosting

- **Local**
  - Your server/network/infrastructure/security/etc.
    * Windows - Internet Information Services (IIS)
    * Linux - Apache Http Server (httpd), Nginx
    * Node - Express (https://expressjs.com)
    * Java - Apache Tomcat

- **Cloud**
  - Someone else's server/network/infrastructure/security/etc.
    * Platform as a Service (PaaS)
    * Infrastructure as a Service (IaaS)
    * Software as a Service (SaaS)

Cloud Options

- **Azure**
  - Hosting Options
    https://docs.microsoft.com/en-us/azure/app-service-web/choose-web-site-cloud-service-vm

    https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal

- **Amazon Web Services (AWS)**
  - Amazon Electric Compute Cloud (Amazon EC2)
  - Amazon Simple Storage Service (Amazon S3)
  - Amazon Relational Database Service (Amazon RDS) or Amazon DynamoDB

- **Google Cloud**
  - Firebase
  - App Engine (PaaS)
  - Compute Engine (VM)

IIS

- **Version History**
  - Current Version: IIS 10 (Windows Server 2016)
  - Previous Version: IIS 8.5 (Windows Server 2012)

- **IIS is a component of the Windows OS**
  - Installation through “Turn on/off Windows feature” option

- **Administration (Requires Local Admin permissions)**
  - Command-Line: %windir%\system32\inetsrv\Appcmd.exe
  - GUI: inetmgr

- **Application Pool (App Pool)**
  - IIS worker process (w3wp.exe): Handles requests for specific app pool
  - App Pool Pipeline
    * Classic - Only configured requests (i.e. .aspx) are routed to the ASP.NET pipeline
    * Integrated - ASP.NET's pipeline handles all requests

IIS Demo

- **Review Installation**
- **App Pool creation**
- **App Pool Identity**
- **Site creation**
- **Binding**

Network Communication

Instructor: Vatanak Vong

Message Protocol

- **SOAP**
  - XML-based
    - Message Envelope
      1) Header
      2) Body*
      3) Fault
  - Secured by Web Services Security (WS-*)
    - SAML
  - Designed to be agnostic to network protocol
    - Through HTTP
      1) HTTP request with *verbs* in the URL dictating action and HTTP request body containing the SOAP message.
      2) HTTP response with HTTP response body containing the SOAP message

- **REST**
  - HTTP-based
    - HTTP Verbs
    - HTTP Status codes
    - HTTP Request/Response
    - **Stateless**
  - Secured by TLS/SSL and custom*
  - JSON over HTTP
  - Invoked as a standard HTTP request and HTTP response

Message Protocol II

- **SOAP**

```
<envelope>
    <header>
        <saml>
            …
        </saml
    </header>
    <body>
        <car year="2017">
            <make>Honda</make>
            <model>Civic</model>
            <vin>usa123</vin>
        </car>
    </body>
</envelope>
```

- **REST**

```
{
    "make": "Honda",
    "model": "Civic"
    "vin": "usa123",
    "year": 2017
}
```

Message Protocol III

- **SOAP Caveats**
  - May require XML namespace or XML Schema definition (XSD)
  - Susceptible to XML buffer overflow attacks
  - Verbose (a lot of info)

- **JSON over HTTP Caveats**
  - Succinct (to the point)
  - Susceptible to encoding and XSS attacks
  - Limited to acceptable characters (unicode + other)
  - Reliant on developers following REST standards

Client-side

- **AJAX/Callbacks**
  - Cancellable
  - Request to any resource over the web
  - Invoke code when AJAX request completes
  - Leads to the pyramid of doom and lackluster error handling

- **AJAX/Promises**
  - ES spec does not allow cancellation, but a lot of libraries do
  - Request to any resource over the web
  - Invoke code when any promise completes
  - Composable with granular consolidated error handler

- **AJAX/Observables**
  - Described as “Promises (for events) with cancellation”
  - Direction of the web ecosystem
  - Reactive programming

Client-side II

- **XMLHttpRequest (XHR)**

```
// Declare XHR object
var xhr = new XMLHttpRequest();

// Registering a callback for when XHR completes
xhr.addEventListener(“load”, eventHandlerFunc);  // Event types: progress, load, error, abort

// Configure as asynchronous request; default is asynchronous
xhr.open(“GET”, “http://www.example.com/data.txt”, true);

// Invoke
xhr.send();
```

- **Fetch API**

```
var request = new Request(“http://www.example.com/data.txt”, { method: “GET” });

fetch(request)
        .then(function (response) {
                …
        })
        .catch(function (error) {
                …
        });
```

Libraries

- **axios**
  - https://github.com/axios/axios
  - HTTP client for browser and NodeJS

- **Angularjs/Angular**
  - $http module

- **Fetch polyfill**
  - https://github.com/github/fetch

- **jQuery**
  - $.ajax()

Design

Instructor: Vatanak Vong

Tips & Tricks

- **Volatility Analysis**
  - Start with purpose/value
  - Helps identify areas that are insulated to change
  - Develop a decision tree based on components that governs other components
    * Laptop => weight governs other components (i.e. battery size, dimensions, screen size, etc.)

- **Abstraction**
  - Dividing concerns of solution into separate layers
  - Identify concerns that crosses layer boundaries (aka “aspects”)
  - Determine how data and errors will flow from one layer boundary to the next and vice versa

- **Technology**
  - Factor in technology into your high-level design to determine components that you get for “free” and components that you have to implement

- **API Usage**
  - Identify potential/pseudo method signatures and how you or how you intend other developers to use your code

Architecture

- **N-tiered Architecture**
  - Solution components organized into deployable units

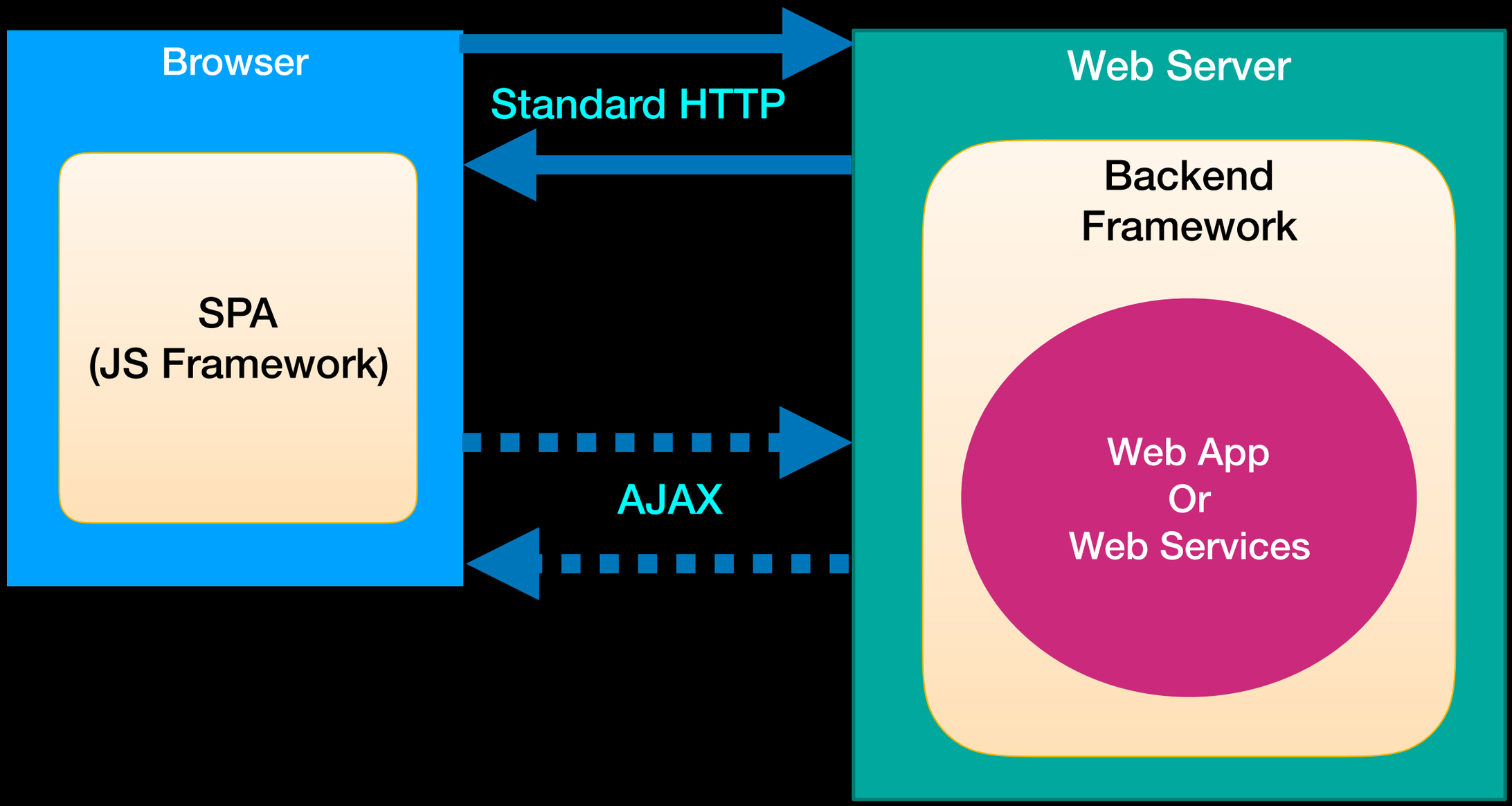
- **Layered Architecture**
  - Separation of concerns
  - **Model-View-Controller (MVC)**
    * Model - Domain/business object, validation logic
    * View - UI, no business logic
    * Controller - Satisfies requesting (routing, DB call, serve view, etc.)
  - **Model-View-ViewModel (MVVM)**
    * Model - Domain/business object, validation logic
    * View - UI, no business logic
    * ViewModel - Handles binding of model data to view, view logic

- **Service-Oriented Architecture (SOA)**
  - From monolithic system to composable, individual services
  - Chain-able services
  - **Microservices**
    * **Autonomous**, single “purpose” services
    * Avoids chaining

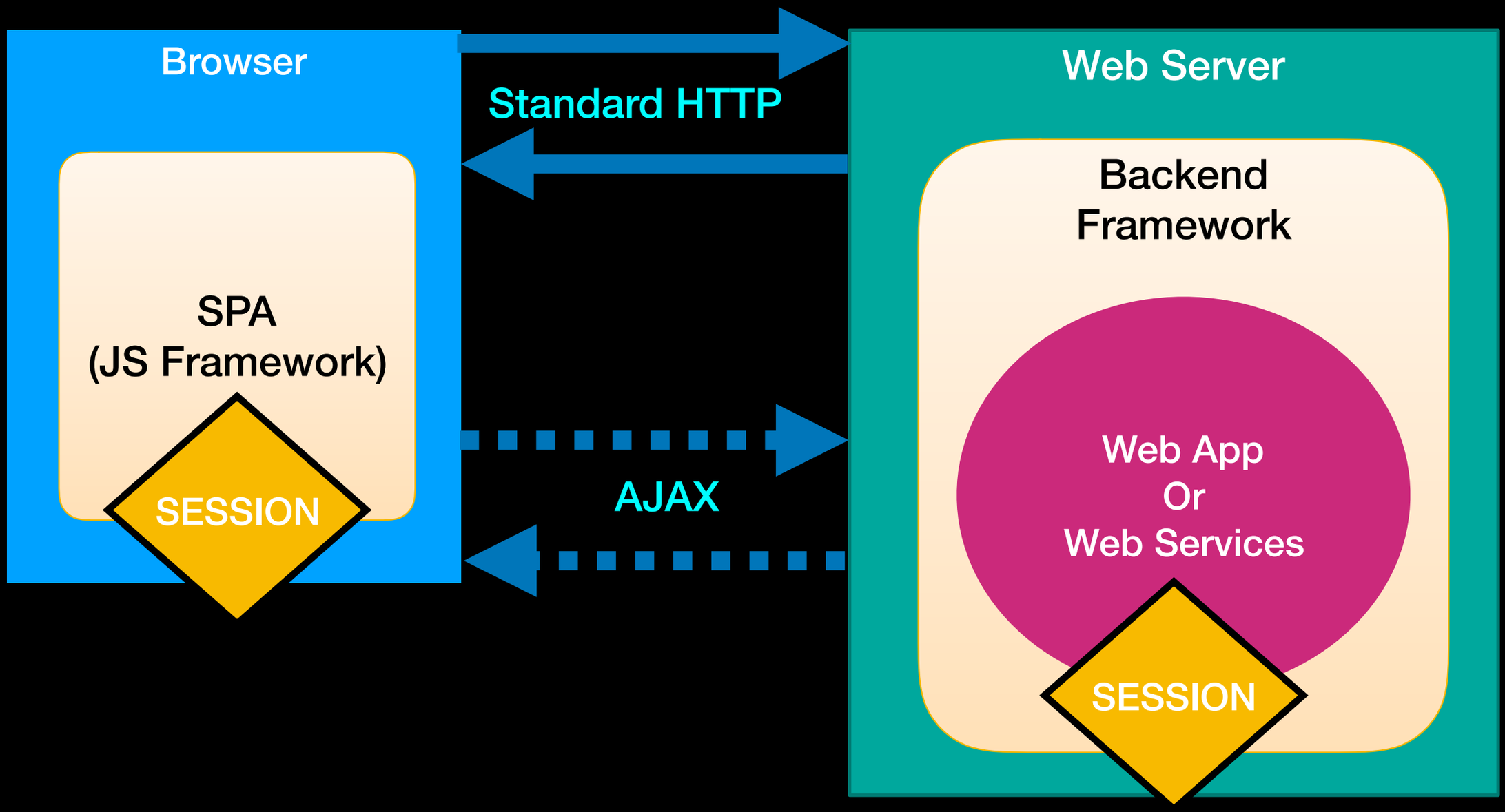
SPA Architecture

- **MVC/MVVM**
  - Frontend
    - index.html (JS, CSS)
    - Client URL Routing
    - Client validation logic
    - XMLHttpRequest (XHR)/Fetch requests aka AJAX
  - Backend
    - Server validation logic
    - Business logic
    - Server URL routing
    - Handles server errors
    - Serve views/JSON data
- **SOA**
  - Frontend
    - index.html (JS, CSS)
    - URL Routing
    - Client validation logic
    - XMLHttpRequest (XHR) requests aka AJAX
  - Backend
    - Server validation logic
    - Business logic
    - Handles server errors
    - Serve views/JSON data

SPA High Level



SPA High Level



SOLID Principles

- **Single Responsibility Principle**
  - Objects should be only responsible for a single concern, domain, etc.
  - Prevents monolithic objects that handle everything (e.g. the “god” object)

- **Open/Close Principle**
  - A system should be open to extension, but closed to modification
  - Discourage modifying existing code, except when original requirements are no longer valid
  - Achieved through inheritance and polymorphism

- **Liskov Substitution Principle**
  - Substituting the parent types with their corresponding derived types should not break the system.
  - Achieved when both a parent and derived object adheres to a behavioral contract

- **Interface Segregation Principle**
  - Many, smaller specific interfaces are preferred over larger, general ones
  - Reduces having objects “inheriting” functionalities that it does not need

- **Dependency Inversion Principle**
  - High level components should interact with low level components through abstractions
  - Achieved further through Inversion of Control

Common Patterns

Instructor: Vatanak Vong

Patterns vs Anti-Patterns

- **Design Patterns**
  - Tried and true solutions (i.e. object structure, code organization, naming, etc.) to common scenarios found in software development
  - Solutions are recognized as patterns by the community over time
  - Example: Data Access Object

- **Anti-Patterns**
  - Common solutions that serve a purpose at time of implementation, but leads to negative consequences
  - Example: “God” object

Facades

- **Facade**
  - Objects that provide simpler APIs to other objects/components

- **Advantages**
  - Insulates the impact when the underlying component undergoes a minor change such as a new optional parameter is added
  - Enhances comprehension/usage of system due to simpler interface

- **Disadvantages**
  - An additional object needs to be updated when major changes occur on the underlying component

Facades

Underlying Component

```
function addString(a, b) {
  return [a, b].join(‘’);
}

function addNumeric(a, b) {
  return a + b;
}
```

Facade Component

```
function add(a, b) {
  if(isString(a) && isString(b)) {
    return addString(a, b);
  }

  return addNumeric(a, b);
}
```

* Assumes isString( ) is a function that returns true if the argument is of type String else false otherwise

Adapters

- **Adapters**
  - A tertiary objects that allows two existing objects to be able to work together.
  - Adapters are needed when you can’t alter the source of existing objects
  - Also called “wrappers”

- **Advantages**
  - Allows for interoperability between objects/systems that was not possible before

- **Disadvantages**
  - An additional object needs to be updated when any change occur on either of the underlying objects/system

Adapters

Separate Component

```
string ModuleA(string data)
{
   …
}

byte[ ] ModuleB(byte[ ] data)
{
   …
}
```

Adapter

```
byte[ ] DataAdapter(string data) {
   …
}
```

Usage

```
string data = “data”;
string modA = ModuleA(data);
byte[ ] adapted = DataAdapter(modA);
byte[ ] modB = ModuleB(adapted);
```

Data Access Object

- **DAOs**
  - Objects dedicated to interacting with a data storage (volatile or persistent)

- **Repository vs Data Access Object (DAO)**
  - A Repository is a specific implementation of a data access object
  - Repositories have a narrow set of public APIs that allows interactions with a data store
  - Most repositories are implemented as CRUD Repositories: Repositories the only allow Create, Read, Update and Delete operations

- **Advantages**
  - Hides the implementation details of your data layer from the rest of your application
  - Insulates the impact of changing the data storage in an application

- **Disadvantages**
  - Can result in abstraction leakage if repository is too strict
  - Design is most affected by technology restrictions (from ORM or from data storage)

Data Transfer Object

- **DTOs**
  - Simple POCOs for moving data across layers/boundaries

- **Advantages**
  - Protects against sensitive data leakage
  - Prevents abstraction leakage
  - Insulates against changes in integration points

- **Disadvantages**
  - Increases number of objects/code files in solution
  - May lead to duplication of validation/business logic
  - Increased complexity

CQRS

- **Command-Query Responsibility Segregation**
  - Separation of read requests from write requests

- **Advantages**
  - Reduces security complexity for read and write operations
  - Prevents unwanted data leakage

- **Disadvantages**
  - Increases number of objects/code files in solution
  - Increased complexity, especially when implemented as two separate data stores
  - Typically implemented with Event Sourcing Pattern

Factories

- **Flavors**
  - Simple Factory / Factory method
  - Factory object
  - Abstract Factory

- **Advantages**
  - Normalize object creation throughout system by centralizing creational logic
  - Reduce duplication of code

- **Disadvantages**
  - Need to manage “types” to feed into factories
  - Increased complexity

Builders

- **Builders**
  - Objects that are responsible for constructing other objects through configurations

- **Advantages**
  - Dynamically construct objects at run-time based on state or settings

- **Disadvantages**
  - Limited to exposed configurations
  - Increased complexity

Strategies

- **Strategy**
  - Object representation of a single workflow (encapsulation of a workflow)

- **Advantages**
  - Abstracting out flows allows for easily swapping logic that needs to be run under specific scenarios
  - Limits impact of changing business rules

- **Disadvantages**
  - Changes in the structure of inputs can require code updates to all strategy objects
  - Increased complexity

Security

Instructor: Vatanak Vong

OWASP

- **What is it?**
  - Consortium of security experts that consolidate security vulnerabilities

- **Reports (Top 10)**
  - Typically 3-year frequency
  - Privacy Risks
    https://www.owasp.org/index.php/OWASP_Top_10_Privacy_Risks_Project
  - Web Security
    https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet
  - Mobile Security

Common

- **Lack of proper Authentication/Authorization**
  - Implemented incorrectly
  - Identify user and manage their permissions

- **Unlimited Sessions**
  - Not revoking access after a time limit
  - Leads to easy replay attacks

- **Data Exposure**
  - Allows client to view/alter more data than it should
  - Easiest method of allowing attackers insight into system

Main Web Vulnerabilities

- **Input Validation & Sanitization**
  - Consortium of security experts that consolidate security vulnerabilities
  - Leads to injection based attacks (i.e. SQL Injection, XSS, DoS)

- **JSON Hijacking**
  - Invoking JavaScript after a JSON over HTTP request

- **Clickjacking**
  - Fooling users into clicking malicious areas (DOM element/iframe) without knowing

- **Cross-Site Request Forgery (CSRF/XSRF)**
  - Executing requests on behalf of active, authenticated user

- **Cross-Site Scripting (XSS)**
  - Executing code that was not built for system

NodeJS

- **Passport**
  - http://www.passportjs.org/
  - Authentication middleware

- **Express Session**
  - https://github.com/expressjs/session
  - Server-side session data management (**development ONLY**)
  - Use a compatible session store
    http://expressjs.com/en/resources/middleware/session.html#compatible-session-stores

- **CSURF**
  - https://github.com/expressjs/csurf
  - CSRF protection middleware

- **Helmet**
  - https://helmetjs.github.io/
  - Adds security related HTTP headers

ASP.NET

- **Action Filters**
  - ValidateAntiForgeryToken
  - Authorize

- **Web.config**
  - <authentication mode="…">

- **SessionAuthenticationModule**

- **IIS Application Pool Identity**

Claims-based Identity

- **Claim**
  - A fact
  - Claim type & claim value pair
  - Role:Admin

- **Identity Context**
  - Who a user is

- **Principal Context**
  - What a user can do

Server-side

Instructor: Vatanak Vong

MEAN Stack

- **MongoDB**
  - NoSQL data store

- **Express (https://expressjs.com)**
  - Minimalistic, web application framework that sits on top of NodeJS
  - Latest version is 4.16.1 (4.16 contains important security updates)
  - Majority of your backend code
    * Custom code should be developed as Express middleware for modularity and composability

- **Angular/Ember/etc.**
  - Frontend JavaScript application framework
  - The frontend framework can be swapped with any framework of your choice

- **NodeJS**
  - Server-side runtime that executes **JavaScript** code
  - Simple web server capabilities

- **[[[Production]]]**
  - Use **Nginx** or **Apache HTTP Server** as a reverse proxy to MEAN

NodeJS

- **What is it?**
  - A C/C++ library

- **Main Components**
  - **V8**
  - **libuv** (event loop)
  - C/C++ add-ons
  - **npm** (Node Packs Manager)
    * CommonJS formats (others: AMD, UMD or requireJS)

- **Purpose**
  - Server-side run-time (NodeJS specific APIs)
  - Allows developers to run JavaScript everywhere

Microsoft Stack

- **.NET Framework (v4.7.1)**
  - Server-side framework and runtime
  - Supported languages are C#/VB.NET

- **ASP.NET (Version depends on .NET version)**
  - Full web application framework built on top of .NET Framework

- **IIS (Version depends on Windows version)**
  - Windows only web server
  - Integrated into all “professional” versions of Windows

- **Angular/Ember/Etc.**
  - Any front-end JavaScript framework

- **SQL Server (2016)**
  - Relational database management system
  - SQL-based data store
  - Data access typically is done using **Entity Framework (v6.2)** or raw **ADO.NET**

ASP.NET

- **What is it?**
  - A web framework that is built on top of .NET developed by Microsoft

- **Main component**
  - HTTP Pipeline
    * HTTP Modules
    * HTTP Handler
    * Global.asax

- **Flavors**
  - WebForms
    * **Page** lifecycle (.aspx - SILVR U)
    * Custom User Controls (.ascx)
  - MVC
    * URL routing
    * Views (.cshtml - Razor)
  - Web API
    * URL routing
    * HTTP Verbs (**RESTful**)
    * HTTPMessageHandler
  - Web Pages
    * Views (.cshtml - Razor)

Microsoft Core Stack

- **.NET Core Framework (v2.0.4)**
  - Open-sourced server-side framework and runtime
  - Supported languages are C#

- **ASP.NET Core (v2.0.4)**
  - Open-sourced Full web application framework built on top of .NET Core

- **IIS/Any web server of choice**
  - No longer restricted to using IIS

- **Angular/Ember/Etc.**
  - Any front-end JavaScript framework

- **SQL Server (2016)/Any data store of choice (NoSQL included)**
  — Data access typically is done using open-sourced **Entity Framework Core (v2.0.4)**

.NET Standard

- **Version vs Standard**
  - Versions may not indicate API surface area across platforms
  - Standards indicate API surface area across platforms

- **Standards**
  - .NET Framework 4.6.1 => .NET Standard 2
  - .NET Framework 4.7.1 => .NET Standard 2
  - Other => .NET Standard 1.5/1.6
  - Lower .NET Standard == higher cross-platform compatibility
  - Higher .NET Standard == more API surface area to use

ASP.NET Core

- **What is it?**
  - Open-sourced web framework that is built on top of .NET Core developed by Microsoft

- **Main component**
  - HTTP Middleware
    * Custom HTTP pipeline

- **Flavors**
  - MVC

Data Stores

Instructor: Vatanak Vong

Main Contenders

- **Relational Database Management System (RDBMS)**
  - Relational (SQL-based/Set theory-based) databases
  - Examples: SQL Server, MySQL, Oracle

- **NoSQL**
  - Umbrella term for any data store that does not rely on relational database system
  - Examples: Cassandra, MongoDB, RavenDB, Redis

Relational

- **Column**
  - Represent the actual data

- **Row**
  - A collection of columns

- **Table**
  - A collection of rows

- **Database**
  - A collection of tables

Relational II

- **ACID Compliance**
  - Atomicity, Consistency, Isolation, Durability
  - A set of constraints to ensure that a successful transaction is valid & permanent and that no change is made for an unsuccessful transaction

NoSQL

- **Document**
  - Stores complex schema-less data structure (typically JSON format)
  - Examples: MongoDB, CouchDB, DocumentDB

- **Column-based (Column-Family or Wide Column)**
  - Stores data by columns instead of by rows
  - Uses a row key to point to a collection of column key and value pairs
  - Easier to retrieve data
  - Examples: Cassandra, Apache Hadoop, Google BigTable

- **Key-Value**
  - Utilizes one to one mapping of key to value. Data type is ignored
  - Also stores metadata needed for indexing data
  - Examples: Memcached, Redis, Riak

- **Graph**
  - The same as document, but add relationship metadata (graph) to data.
  - Improves node traversal.
  - Examples: Neo4j, Giraph

Data Design

Instructor: Vatanak Vong

References

- **Relational Database Design**
  - http://www.tomjewett.com/dbdesign/dbdesign.php

Relational

- **Primary Key**
  - Stores complex schema-less data structure (typically JSON format)
  - Examples: MongoDB, CouchDB, DocumentDB

- **Foreign Key**
  - Stores data by columns instead of by rows
  - Uses a row key to point to a collection of column key and value pairs
  - Easier to retrieve data
  - Examples: Cassandra, Apache Hadoop, Google BigTable

  - Utilizes one to one mapping of key to value. Data type is ignored
  - Also stores metadata needed for indexing data
  - Examples: Memcached, Redis, Riak

- **Graph**
  - The same as document, but add relationship metadata (graph) to data.
  - Improves node traversal.
  - Examples: Neo4j, Giraph

General NoSQL

- **Denormalization**
  - A denormalized data set makes querying data much faster and easier
  - Instead of having two sets for Person and Address, just have both attributes in Person

- **Nesting**
  - Entities should rely on nesting to denote relationship, especially for static collections
  - Improves functional search techniques

Key-Value

- **Composite Key**
  - For ordered key stores, use keys that contains a specific format to create multi-dimensional indexes
  - Example:
    FacultyType:LastNameChar
  - Improves searches

Web Services

Instructor: Vatanak Vong

WS Basics

- **What is it?**
  - Method that can be executed over a public or private network
  - Public networks being the internet and private networks being the LAN/WAN

- **Terms**
  - **Web Service**: The general term for executable code accessible through a network
  - **End-point**: The actual resource to invoke a web service, which is typically the URL. For REST services it is also the HTTP Verb.

Technology

- **MEAN**
  - Native NodeJS
  - ExpressJS (Preferred)

- **Microsoft**
  - ASP.NET MVC (General web services)
  - ASP.NET Web API (REST-based Services)
  - WCF (SOAP-based Services)
  - ASP.NET ASMX (Depreciated)

Debugging

- **Testing/Debugging**

  - Since web services are URL-based, you just need an HTTP client to send requests to the web service end point

- **HTTP Clients**

  - MacOS/Linux
    * cURL
  - Windows
    * Postman: A Google Chrome Extension
    * Fiddler: An all purpose network proxy made by Eric Lawrence
      ❖ Fiddler was bought by Telerik (a subsidiary of Progress)

Examples

- **cURL**
  - https://ec.haxx.se/http.html

  - GET: default

    curl http://someurl

  - POST: -d or -F

    curl -d ‘{ “data”: “value” }’ -H “Content-Type: application/json” -X POST http://someurl

DEMO
Instructions

Instructor: Vatanak Vong

Web Dev Environment

1. Install latest LTS version of NodeJS (nodejs.org)
    - NPM comes installed with NodeJS, but ensure that it's the latest version by running "npm update"
2. Install VS Code (code.visualstudio.com)
3. Launch VS Code
4. Open "Integrated Terminal" in VS Code
    1. Navigate to a working directory (create a directory if desired)
    2. Run "npm init -y" to create the package.json file in working directory
    3. Run "npm install lite-server --save"
5. Open package.json
    1. Manually add "debug": "lite-server" to the "scripts" section of the package.json
    2. Save the package.json
6. Create a javascript file (e.g. app.js) at the root of your working directory
7. Create an index.html file with standard HTML5 structure; include reference to your JavaScript file
8. In "Integrated Terminal", run "npm debug" to begin debugging

ASP.NET Projects

- **Production**
  - Start with “Empty” template project instead of pre-made ones to avoid having unnecessary project dependencies
  - Use the WebApiConfig.cs to setup JSON formatting in Web API
  - Use global.asax to configure custom event handling in the ASP.NET pipeline
  - SPA
    * Use Web API to build stateless web services for your SPA to consume
    * Use MVC if you need to handle serving views, data, or other advance server-side scenarios
    * Use a JavaScript library to handle UI/UX
    * If using MVC, the initial SPA view is served up by MVC “default” controller
  - All inputs must be re-validated on the backend
    * Your system should not trust the client
  - Security
    * Use [Authorize] authorization filter to deny unauthenticated users from accessing Web API & MVC endpoints

TypeScript

1. Setup working directory

2. npm init -y

3. npm install --save typescript

4. npm install --save-dev http-server

5. Add a new “start” command with “http-server” in the “scripts” property of the package.json file

6. Create a tsconfig.json file at the working directory to configure TypeScript with the following properties using “tsc —init" on the terminal

```
{
  "compilerOptions": {

    "target": "es2015",         /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', or 'ESNEXT'. */
    "module": "amd",      /* Specify module code generation: 'none', commonjs', 'amd', 'system', 'umd', 'es2015', or 'ESNext'. */
    "outFile": "./js/app.js",  /* Concatenate and emit output to single file. */
    "outDir": "./js",          /* Redirect output structure to the directory. */
    /* Strict Type-Checking Options */
    "strict": true                           /* Enable all strict type-checking options. */
  },
  "files": [  // Files to include in compilation
    "app/app.ts"
  ],
  "include": [              // A glob specifying files to include based off of a pattern
    "./**/*.ts",   // Look for TS files in the current directory and all sub directories
    "./**/*"        // Look for all files in the current directory and all subdir that TS can compile
  ]
}
```