Logols Learning

WEEKEND WEB DEVELOPMENT BOOT CAMP

TRAINING: REVIEW

Select Statements

- Components in Logical Order
  - SELECT
  - FROM
  - WHERE
  - GROUP BY
  - HAVING
  - ORDER BY

- Example:

```
SELECT FirstName,
LastName
FROM Person
WHERE personId = 2
```

Insert Statements

- Components
  - INSERT
  - INTO
  - VALUES
- Example:

  INSERT INTO Person (FirstName, LastName)

  VALUES ('Joe', 'Mackie')

Update Statements

- Components
  - UPDATE
  - SET
  - WHERE
- Example:

```
UPDATE Person
  SET FirstName = 'Joe'
WHERE LastName =
'Mackie'
```

Delete Statements

- Components
  - DELETE
  - FROM
  - WHERE
- Example:

```
DELETE
FROM Person
WHERE LastName =
'Mackie'
```

MySql Data Types

- int, smallint, tinyint, bigint
- bit
- numeric, decimal
- date, datetime
- char, text, varchar

Create Table

- Components
  - CREATE TABLE
  - (
    - Column1 data type,
    - Column2 data type
  -);

```
CREATE TABLE People
(
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Primary Key & Auto Increment

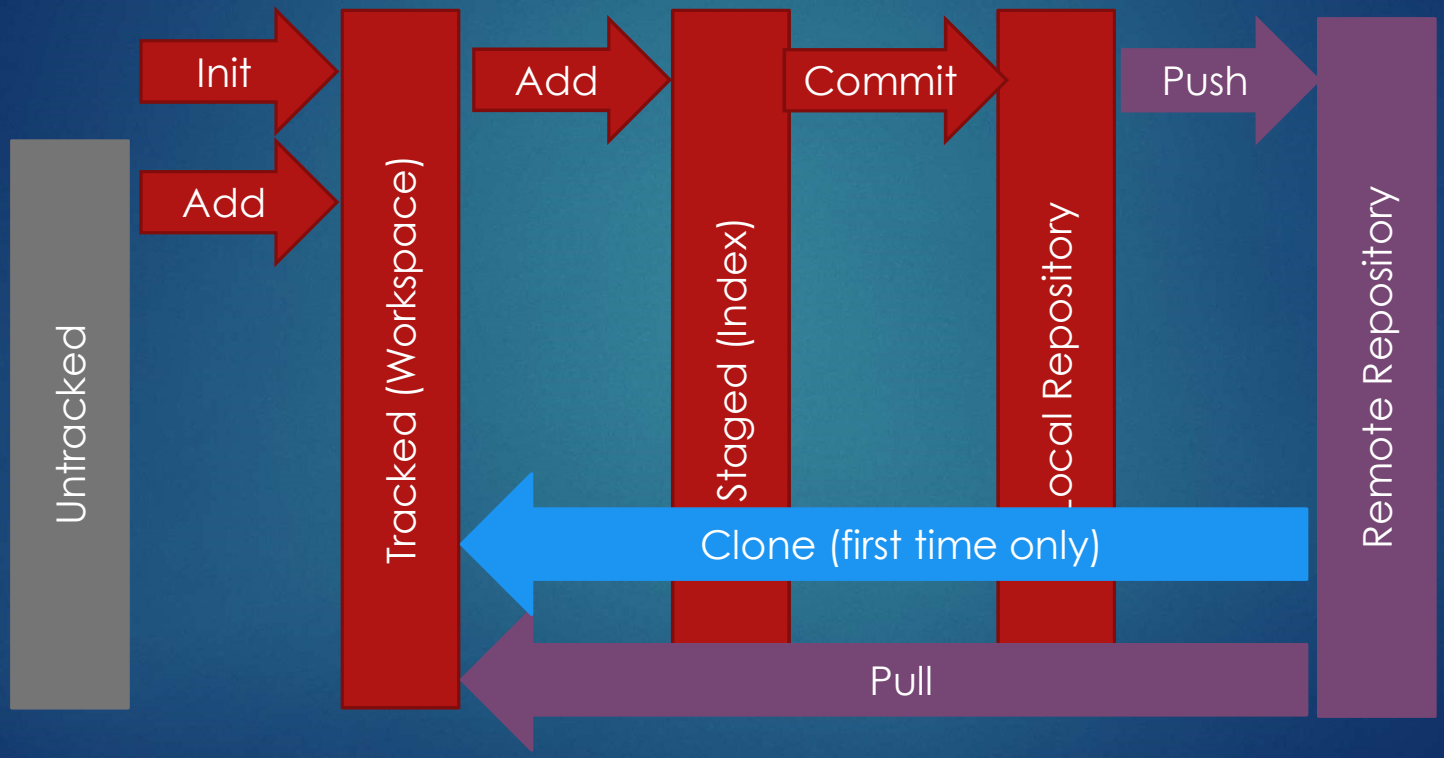
```
CREATE TABLE People
(
    PersonID int not null AUTO_INCREMENT,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    PRIMARY KEY (PersonID)
);
```

Inner Join

- Every row from the first table will be matched with every row from the second table based upon the on conditions specified.

```
SELECT c.ClassId, c.ClassName, t.TeacherId,
t.FirstName, t.LastName
FROM Class c
INNER JOIN Teacher t
ON c.TeacherId = t.TeacherId
```

Git Workflow



CLI new Examples

- mkdir – create directory
- cd – change directory
- Console project:
  - dotnet new console
- Class Library project:
  - dotnet new classlib
- Web API project:
  - dotnet new webapi

Statements

- Made up of:
  - Keywords
  - Expressions
  - Operators
- Statements end with a Semicolon ;
- Statements can span multiple lines
- Statement blocks contain multiple statements
  - Surrounded by curly braces { }
  - Can have blocks within blocks

C# & JavaScript Comments

- // this is a comment
  - Single line comments
- /* this is a multi line comment */
  - Multi-line comments

Types

- Basic Built-In Types
  - bool
  - int
  - decimal
  - string
  - array

C# & JavaScript Declaring Variables

C#

- Declaring Variables
  - string myString;
  - string myString = “test string”;
- Using Variables
  - Console.WriteLine(myString);

JavaScript

- let [name];
- let [name] = [value];

Example:

```
let message = “hello”;
alert(message);
```

C# and JavaScript Logical Operators

- && Conditional And
- || Conditional Or

C# & JavaScript If-Else Statement

- C#:

```
bool myVariable = true;
if (myVariable)
{
   console.writeLine("true");
}
else
{
 console.writeLine("false");
}
```

- JavaScript:

```
let myVariable = true;
if (myVariable) {
   console.log("true");
}
else {
 console.log("false");
}
```

C# & JavaScript Switch Statement

- Example

```
switch(myVariable)
{
   case 1:
      Console.WriteLine("1");
      break;
   case 2:
   case 3:
      Console.WriteLine("2 or 3");
      break;
   default:
      Console.WriteLine("default");
      break;
}
```

C# Value and Reference Types

- Type System
  - Value Types
    - Contain data within it's own memory location.
    - int, decimal, bool
  - Reference Types
    - Contain a pointer to a memory location.
    - Require a new instance of an object.
    - Are null if no instance of an object has been provided.
    - string, array, class

Default Values for C#

- Value Types
  - 0 for int or decimal
  - false for bool
- Reference Types
  - null
  - This means lack of a value
  - To check for null
    - If (variable == null)

C# & JavaScript Declaring Arrays

- C#
  - int[] myArray;
  - myArray = new int [5];
  - myArray = new int[] {0, 1, 2, 3};
  - int[] myArray = new int[] {0, 1, 2, 3};
  - int[] myArray = {0, 1, 2, 3};
- Using Variables
  - myArray[5] = 6;
  - Console.WriteLine(myArray[5]

- JavaScript
  - let myArray = [0, 1, 2, 3];
- Using Variables
  - myArray[5] = 6;
  - console.log(myArray[5]);
  - myArray.length

for Loop

C#

```
int[] myArray = {0, 1, 2, 3};

for(int counter = 0;  counter < myArray.Length; counter++)
{
   Console.WriteLine(myArray[counter]);
}
```

JavaScript:

```
let myArray = [0, 1, 2, 3];

for(let counter = 0;  counter < myArray.length; counter++)
{
   console.log(myArray[counter]);
}
```

foreach / let of Loop

```
int[] myArray = {0, 1, 2, 3};
foreach(int value in myArray)
{
   Console.WriteLine(value);
}
```

```
let myArray = [0, 1, 2, 3];
for(let value of myArray)
{
   console.log(value);
}
```

C# Function Syntax

```
[access modifier] [return type] [name]([type1] [parameter1], [type2] [parameter2])
{
    Statements...;
}
```

- Example:

```
private int AddNumbers(int num1, int num2)
{
    Statements...;
}
```

JavaScript function Syntax

- Function performs an action
- Can also be a type in Javascript

Example:

```
function square(num) {
  return num * num;
}
alert(square(4));
```

C# Class Syntax

```
[access modifier] class [name] : [base class], [interface1], [interface2]
{
    Statements…
}

public class Car : Automobile, IPositionWriter
{
  Statements…
}
```

C# Namespace Syntax

```
namespace [name]
{
   Statements…
}

namespace Logols.Assessment.Entities.Subjects
{
   Statements…
}
```

C# Constructor

- Method called when a class is instantiated
- Method Name = Class Name
- Return type or void is not used
- Can be overloaded

- Example:

```
public class Car
{
   public Car()
   {
      Statements...;
   }
}
```

TypeScript Syntax

- Typescript is written in .ts files that are transpiled to .js files
- Also option to create .d.ts declaration files for intellisense
- Variable Declaration: let[name] :[type] = [value];

function [name] ([param1]:[type], [param2]:[type]) : [return type] {}

Ex.

```
class [name] {
  name:string;

  constructor(name:string) {
    this.name = name;
  }

  private write():void {
    console.log("Name is " + this.name);
  }
}
```

C# Property Syntax

General Syntax

```
[access modifier] [type] [name]
{
  get
  {
    Statements…
  }
  set
  {
    Statements…
  }
}
```

Fully Implemented Example

```
public int Count
{
  get
  {
    return _count;
  }
  set
  {
    _count = value;
  }
}
```

Auto Implemented Example

```
public int Count { get; set; }
```

Instantiating and Using Objects

- A class needs to be instantiated to be used
- A class can be instantiated many times
- One instance of a class does not effect another

C# Car car = new Car(); or JS: let car = new Car();

```
Console.WriteLine(car.DistanceTraveled);
car.Drive(15);
```

Scope Access Modifiers

- public – accessible to everyone, not restricted
- Internal – access limited to current assembly
- private – access limited to defined class
- protected – access limited to derived classes

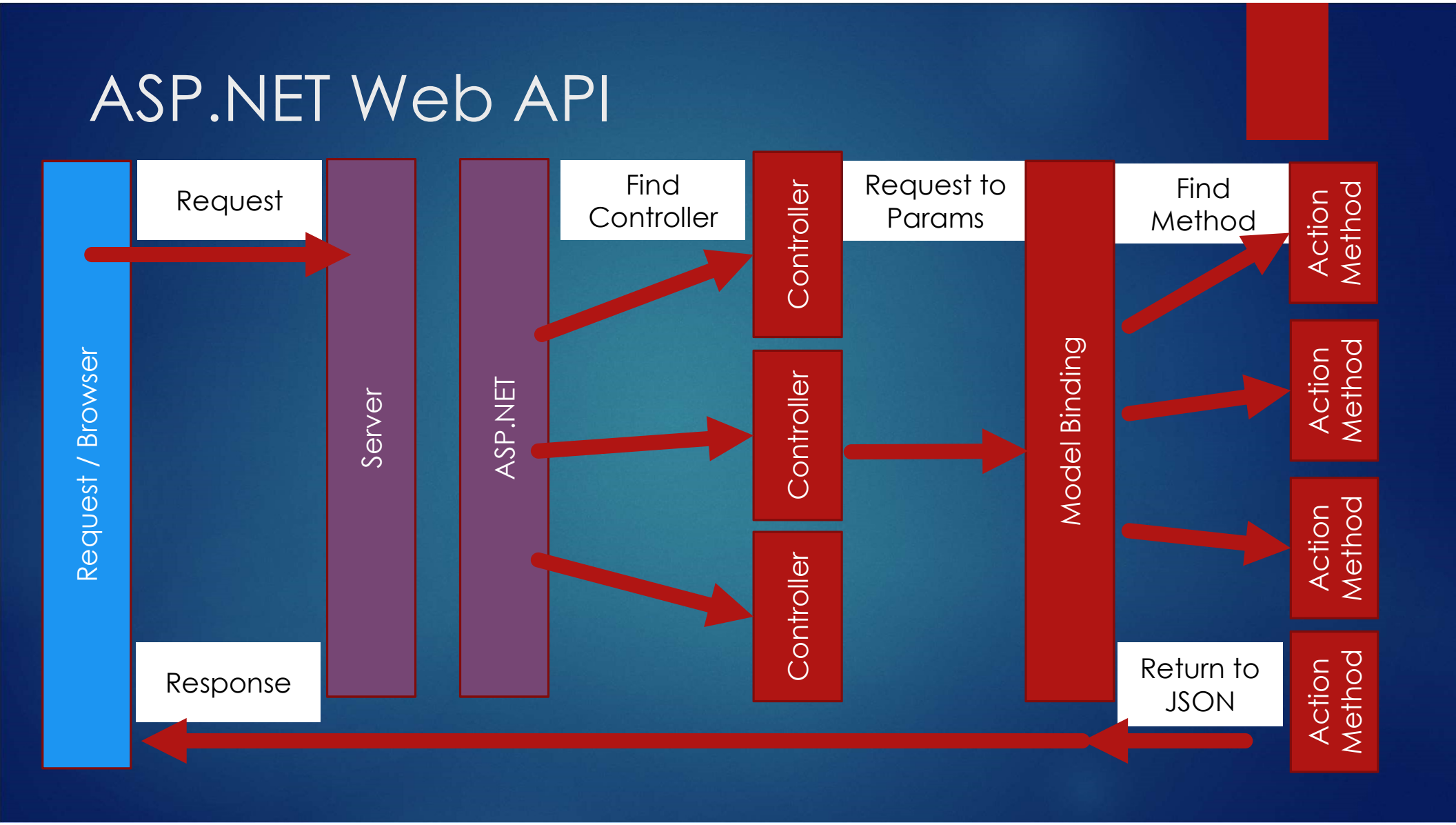
Base Repository

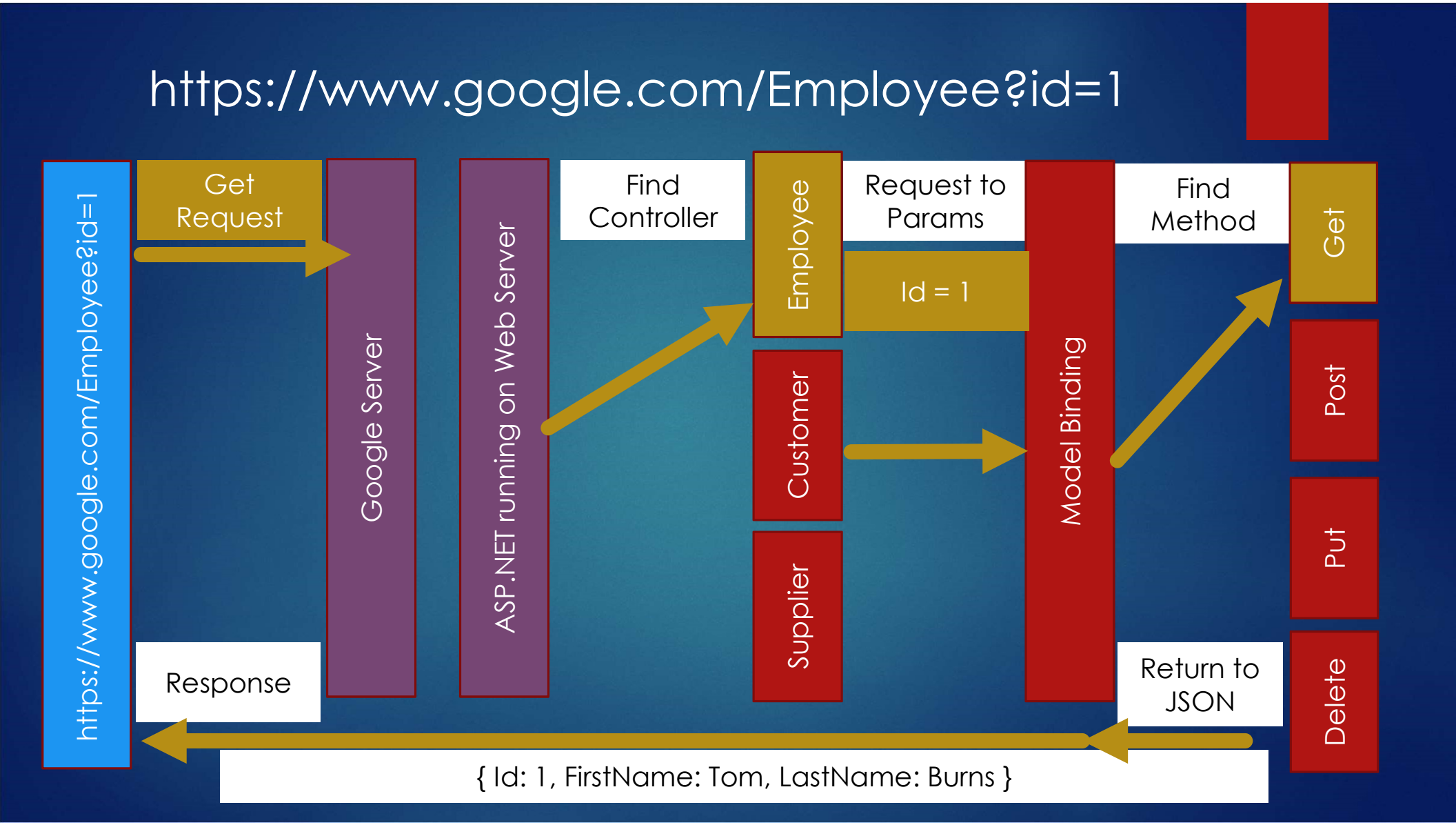
```
private string connectionString;
public Repository()
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json");
    var connectionStringConfig = builder.Build();
    connectionString = connectionStringConfig.GetConnectionString("DefaultConnection");
}
public IDbConnection Connection
{
    get { return new MySqlConnection(connectionString); }
}
```

Repository

```
public Subject Get(int subjectID)
{
   using (IDbConnection dbConnection = Connection)
   {
      dbConnection.Open();
       string sql = "Select SubjectId, Name, Description From Subject Where SubjectId = @SubjectId“;
      return dbConnection.Query<Subject>(sql, new { SubjectId = subjectID }, commandType: CommandType.Text).FirstOrDefault();
   }
}
```

ASP.NET Web API




https://www.google.com/Employee?id=1
https://www.google.com/Employee?id=1
Get Request
Google Server
ASP.NET running on Web Server
Find Controller
Employee
Customer
Supplier
Request to Params
Id = 1
Model Binding
Find Method
Get
Post
Put
Delete
Return to JSON
Response
{ Id: 1, FirstName: Tom, LastName: Burns }

Response Binding

- Return value to JSON Data
- Example:

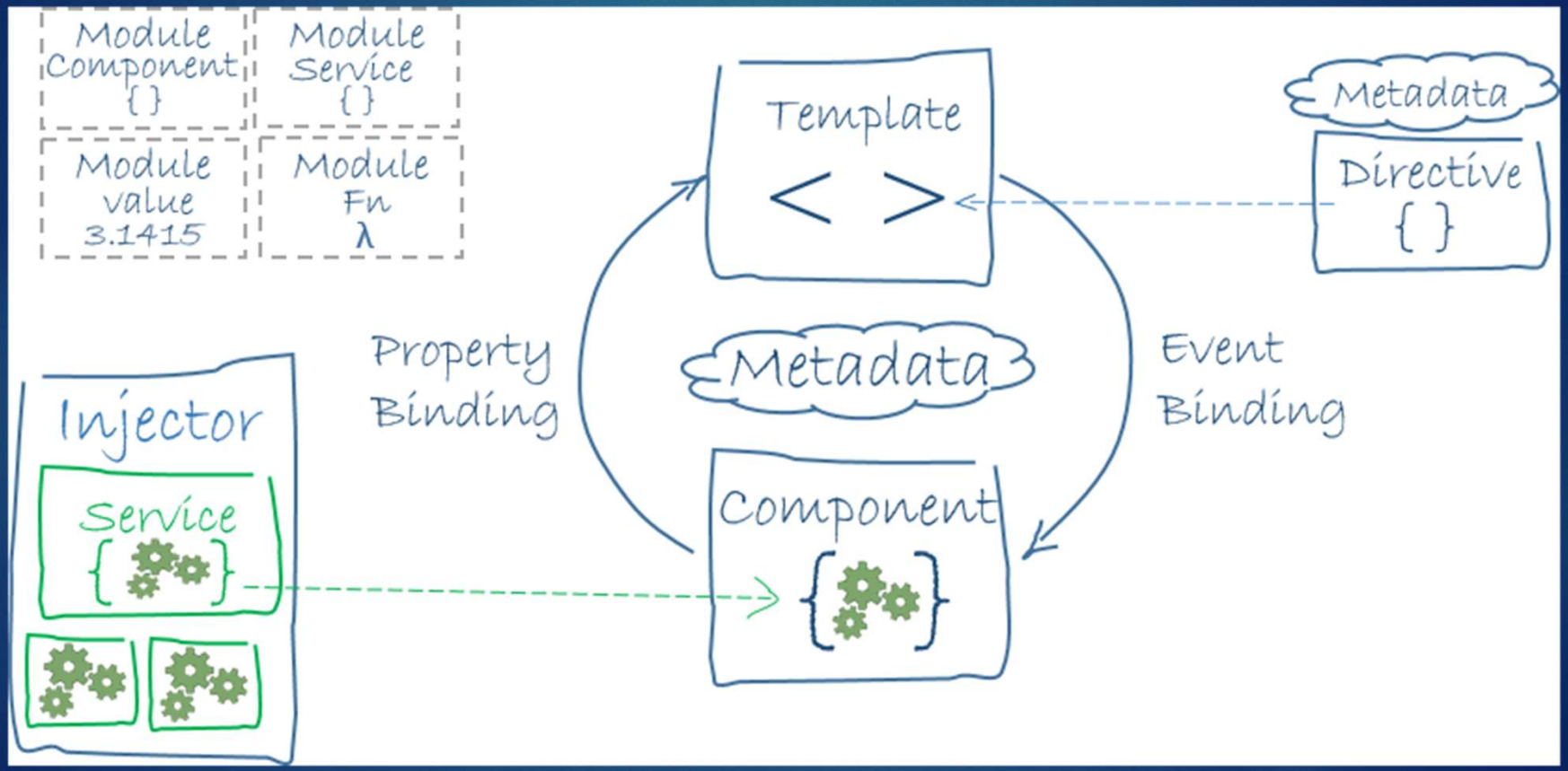
```
[HttpGet]
public IEnumerable<TimeTraveler> Get()
{
    return timeTravelerRepository.GetAll();
}
```

Parameter Binding

- JSON in Request Body converted to Class
- Example:

```
[HttpPost]
public void Post([FromBody]Answer answer)
{
   _service.Insert(answer);
}
```

Architecture of Angular



Component Example

```
import { Component } from '@angular/core';

@Component({
    selector: 'timemachine',
    templateUrl: './timeMachine.component.html',
    styleUrls: ['./timeMachine.component.css']
})
export class TimeMachineComponent {
}
```

app.module – Root Module

- declarations
  - Declare available components
- imports
  - Reference other modules
- Providers
  - Define available services
- bootstrap
  - Define root components to be loaded

Directive Examples

```
<tr *ngFor="let log of logs">
    <td>{{log.firstName}}</td>
    <td>{{log.lastName}}</td>
    <td>{{log.travelToYear}}</td>
</tr>

<p *ngIf="!assessments"><em>Loading Assessments...</em></p>

<p [ngClass]="{'hidden':assessments.length > 0}"><em>Loading
Assessments...</em></p>
```

Data Binding Examples

- Interpolation

Hello {{ name }}

- One-way binding

<input type = 'text' [value]="firstName" />

- Two-way binding

<input [(ngModel)]="firstName" />

- Event binding

<button (click)="onSaveClick()">Save</button>

Selectors

- Used to select elements for styling
- Sample below

Name	Syntax	Example
ID	#[Element ID]	#PersonTable
Class	.[Class Name]	.highlightTable
All Elements	*	*
Element	[Element]	p
Element in Element	[Element] [Element]	div p
Multiple Elements	[Element], [Element]	div, p
Element direct child	[Element] > [Element]	div > p

Box Model