SPA: Single Page Application und PWA: Progressive Web Apps

Web App: +Plattformunabhängig, +Kein Backup, +Einfaches Software Update, +SaaS möglich, -Screen Optimization, Limitierter Zugang zu Hardware/OS

SPA: Inhalte anstatt ganze HTML Seiten werden dynamisch geladen und im DOM ersetzt. Logik vom Server wandert in den Client:

Views/Routing: Plain HTML/CSS/JS; no page reloads; working back-button; bookmarkable links
Models/Services: Provides offline functionality
Data Access: Uses RESTful services for data access

Vorteile: Geschwindigkeit, Offline Friendly, No Page Reloads, Complex Navigation is easy
Nachteile: SEO (search engine optimization) Support, Initial Page Load, Application Size, (Back-btn, Book-Marking)

PWA: Webseiten die wie eine native Applikation daherkommen (Offline Support mit Service Worker)
Vorraussetzungen: TLS, Web App Manifest mit name, short_name, start_url, display, icon (144x144)

Service Workers: scriptable network proxy in the browser to manage HTTP requests programmatically
+ Besser User Experience, +Eine Codebase für Web/Mobile, -Hardware Zungang abhängig vom Standrd.

Vue (sehr flache Lernkurve)

Lifecycle Hooks: created (fetch data here), mounted, updated, destroyed
methods: add methods for eventhandlers here
computed: add complex logic here, instead of in template e.g. message.split('').reverse().join(''), computed properties are cached! // usually best option!
watch: perform asynchronous or expensive operations in response to changing data

Directives: Form: v-directive='expression', get automatically refreshed, when dependeny changes

Event Listener:	`<a v-on:click="doSomething"></a>` → Shorthand: `<a @click="doSomething"></a>`
For Loops:	`<ul><li v-for="i in list" v-once>{{i.firstName}}</li></ul>` // render only once, after that static
If and Show	`<h1 v-show="ok">Hello!</h1>` or `<h1 v-if="seen">Hello!</h1>` // prefer v-show if you need to toggle something very often, and prefer v-if if the condition is unlikely to change at runtime
V-Bind	`<button v-bind:disabled="isDisabled">Click</button>` // mustaches {{}} cannot be used within html
V-Bind Shorthand:	`<a :href=""/>`
Binding:	`<input v-model="person.name">` // 2-Way `<p>Message is: {{ person.name }}</p>` // 1-Way
Skip HTML Rendering	`<div v-html="html"></div>`

Components: JS → Vue.component('my-component', { template: '<div>A custom component!</div>'})
HTML→ `<div id="app"> <my-component></my-component> </div>`

Bundler: A JavaScript bundler is a tool that bundles your code into one JS-file (Gulp, Grunt, Webpack)

```
<!DOCTYPE html><head> <script src="https://cdn.jsdelivr.net/npm/vue"></script></head> <body>
   <div id="app"> <h2 class="hello-title">Hello {{name}}!</h2> </div>
   <script type="text/javascript">
     const vm = new Vue({ el: '#app',
        data: { name: 'Hello Vue!', selected: '', data: [] },
        created: function () { ('/spa/'+window.location.hash.substring(1)).then(response => response.json())
           .then(body => { this.name = body.value;}); }});
        methods: { doSomething: function() {…}, AnotherFunction: function() {…},   // not cached
        computed: { reversedName: function() {this.name.split('').reverse().join();}, // cached
        watch: { name: function(newVal, oldVal) { this.name = oldVal + ' changedTo ' + newVal; } }
    </script> </body> </html>
```

React (The V in MVC) → Library, Kein Framework!

- Props sind Parameter einer Komponenten → Props sind immer read-only.
- State wird zum Zwischenspeichern von Daten zwischen den Renderings verwendet → Der State ist immer private innerhalb einer Komponente → Kann aber via Props weitergegeben werden
- Keine von Props abgeleiteten Daten im state speichern!
- Container und Presentation Componenten trennen!
- React Componenten müssen mit einem Grossbuchstaben beginnen
- JS Keywords können nicht verwendet werden (z.B className anstatt class)
- Styles werden als Objekt gesetzt (CamelCase verwenden!)
- Bei Listen sollte immer ein Key verwendet werden, damit bei einer Positionsänderung das Element wiederverwendet wird

Tooling: +Auto Reload, +Build Optimization, +Sprechende Fehlermeldungen

- JSX Conditionals: Kein If möglich, da Statement → Expression z.B Ternary Operator nötig (a==0?a:b)
- Was zu null, true, false oder undefined evaluiert wird nicht ausgegeben
- JSX wird vom Präprozessor zu React.createElement Aufrufen gewandelt

Reconciliation:
1. Render Virtual DOM 2. setState verändert Virtual DOM 3. während Aufruf render(): Diff Old DOM / New DOM 4. Create real DOM Node
- setState nimmt das Objekt und merged dieses mit dem existierenden state. → Auto Re-render after setState
- Nur angegebene Properties werden dabei überschrieben!
- State-Updates können zusammengefasst werden und laufen asynchron ab

Lifecycle: (nur Klassenkomponenten haben einen Lifecycle):
Mounting:
- constructor(props) → State initialisieren
- render()
- **componentDidMount**() → DOM ist aufgebaut, Load Async Data, setState = re-render

Updating:
- componentWillReceiveProps(nextProps) → Falls state von prop abhängig
- shouldComponentUpdate(nextProps, nextState) → true/false render?
- componentWillUpdate(nextProps, nextState)
- render()
- componentDidUpdate(prevProps, prevState) → DOM ist aktualisiert

Unmounting:
- componentWillUnmount() → aufräumen

Funktionale Komponente: Nur verwenden wenn kein State
function App(props) { return (<div> <HelloMessage name="HSR"/>{props.name}</div>) }

Component Mounting: ReactDOM.render(<App/>, document.getElementById('root'))
Klassenkomponente: Zusätzlich Methoden, State, Lifecycle Hooks

```
class Counter extends React.Component {
  constructor (props) {
    super (props);  props.children; // Child HTML Elements
    this.state = { counter: 0, username: "" };
  }
  increment() { this.setState( state => ( {counter: state.counter + 1} ) ) }
  validate = (event) => { event.preventDefault(); } // Kein bind(this) bei Lamda Syntax nötig
  render = () => (
     <div className='container'>
       <input value={this.state.counter} onChange={this.validate} /> <button onClick={ this.increment.bind(this) }>
       <AnotherComponents {...this.props}
     </div>   ) }
```

Container Komponenten:

```
class CommentListContainer extends React.Component {
  state = { comments: [] }
  componentDidMount() {
     fetch('/comments').then(response =>
      response.json().then(comments =>
      this.setState({comments})))
  }
render = () => <CommentList
  comments={this.state.comments}/>   }
```

Presentation Komponente:
+ wiederverwendbar, +einfacher testbar, +lesbarer

```
function CommentList({comments}) {
  const renderComment = ({body, author}) =>
     <li>{body} --{author}</li>
  return <ul> {comments.map(renderComment)} </ul>
}
```

Redux (State ist ohne Redux überall verteilt → oft brauchen mehrere Komponenten die selben Daten)

State Management Library: Representation des States sowie Benachrichtigung bei Änderungen
+ Zustand an einer Stelle + Einfacheres Debugging – Lohnt sich nur bei viel State→ Overhead!

Store: wird als immutable State-Tree von Objekten dargestellt (Single Source of Truth)
Action: Verändert den State { type: 'ADD_TODO', text: 'Learn React' } → Dispatch to store
Reducer: (pure JS Funktion) erstellt einen neuen State Tree. Enthält alten State und Action. Darf keine Seiteneffekte haben, keine Severcalls! → Ist immer nur für einen Slice des State-Trees zuständig!
Connect-Methode: Das Resultat von connect ist eine React-Komponente

```
function todos(state = [], action) {
  switch (action.type)
    case 'ADD_TODO':
      return [ ...state, { text: action.text, completed: false } ]
  default: return state // default: return old state }}
```

Router

```
const App = () => (
  <Router> <div>
    <ul> li><Link to="/">Home</Link></li></ul>
    <Route exact path="/" component={Home} /> //wird gerendert, sobald path matched
    <Route path="/topics" component={Topics} />
    <Route exact path="/" render={() => ( loggedIn ? ( <Redirect to="/dashboard"/> ) : ( <PublicHomePage/> ) )}/>
 </div> <Router>)
```

Jest und Enzyme

Jest: +Kommt bereits mit create-react-app, +Interaktiver Watch Modus, +Snapshot Testing, Code Coverage, +Mocks für Callbacks, +Expect Methdos
Enzyme: Einfachere Asserts, Manipulation und Traviersierung von Komponenten (Shallow, Mount)

React Selbststudium: React Performance Testing

React's performance tools: react-addons-perf → methods to measure rendering time of a component and how many unnecessary renderings (when nothing changed) were made. To minimize wasted renderings, use lifecyclehook *shouldComponentUpdate()*

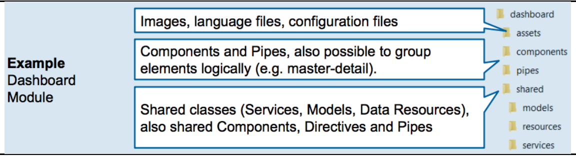
Angular (für langlebige, wartbare SPA, gut geeignet für distributed development)

+ TypeScript 2.0, + Integrated Depency Injection Container, +Sehr strukturiert

View Encapsulation: defines whether the template and styles defined within the component can affect the whole application → **ShadowDOM** ermöglicht Style Encapsulation! Angular can either use ShadowDOM or for older browsers can emulate a ShadowDOM
Change Detection (on Event, XHR, Timers): works with ngZone, each component has its own change detector, performace improvement: mark Component with e.g. ChangeDetectionStrategy.OnPush
Zone: Execution context that allows us to hook into asynchronous tasks

Modules //A cohesive block of code dedicated to closely related set of capabilities.

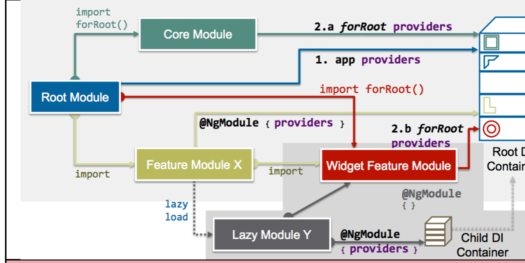
App/Root: Bootstrapping (keine Exports!)
Core: Hält das App Module aufgeräumt (wird vom App Module Importiert) + Global Services
Shared: Common components, services for Feature Modules (Keine App-wide Singleton! →wegen Lazy)
Feature-Module: Domain, Routing, Service, Widget (z.B Material), Lazy Modules (Own DI-Container!)


Example Dashboard Module
Images, language files, configuration files
Components and Pipes, also possible to group elements logically (e.g. master-detail).
Shared classes (Services, Models, Data Resources), also shared Components, Directives and Pipes
dashboard
assets
components
pipes
shared
models
resources
services

```
const EXPORTED_DECLARATIONS = [  // External View Classes (Components / Directives / Pipes) ];
const INTERNAL_DECLARATIONS = [ ...EXPORTED_DECLARATIONS,  // Internal Classes (Components / Directives / Pipes)];
const EXPORTS = [  ...EXPORTED_DECLARATIONS  // External Modules to export ];

@NgModule({
 declarations: INTERNAL_DECLARATIONS, // components, directives, pipes
 imports: [ // Other Modules to import (imports the exported Components/Directives from the other module) ],
      CoreModule.forRoot();  // Only call in App Module!!!
      AppRoutingModule
 exports: EXPORTS,
 providers: [   // Services for the global store of services ]
 bootstrap: [ AppComponent // only in the App Module!!!! ]
})
export class AppModule {
   static forRoot(config?: {}): ModuleWithProviders { // Only call in App Module!!!
      return {
         ngModule: MyModule,      // Declare in Feature or Core Module Module!
         providers: [ GlobalService // Global providers, instantiated exacly once ]
   };}
   constructor (@Optional() @SkipSelf() parentModule: CoreModule) { // Only in Root Module: Guard against dupl. Import
      if (parentModule) { throw new Error( 'CoreModule is already loaded. Import it in the AppModule only'); }
   } }
```

@NgModule({ imports: [ForeignModule.forChild({ })] }) → Configure Services for the current Module (z.B RouterModule)
@NgModule({ imports: [ForeignModule.forRoot({ })] }) → Provider werden von Lazy Modules nur einmalig geladen. Nur im App Module aufrufen. Services entweder in @NgModule oder forRoot Methode deklarieren. **NIE in beiden**!!!



Component // directive-with-a-template; controls section of view. must be declared in exactly one NgModule.

Lifecycle: constructor > ngOnChanges > **ngOnInit** (*Hydration: fetch data*) > ngDoCheck > (ngAfterContentInit > ngAfterContentChecked > ngAfterViewInit > ngAfterViewChecked) > **ngOnDestroy** (*Dehydration: detach event handler*)

```
@Component({
  selector: 'wed-navigation', // <wed-navigation></wed-navigation>
  templateUrl: './navigation.component.html',
  styleUrls: ['./navigation.component.css'], providers: [UserService]
})
export class NavigationComponent implements OnInit, OnDestroy {
    @Output() click = new EventEmitter<any>();  // <wed-navigation (click)=""  → Fire from inside the component
    @Input() title: string;                     // <wed-navigation [title]=""  → Consume bindable values (Attr. directive)
    private counters:CounterModel[];
    private counterSubscription:Subscription; ,   // Subscription for a EventEmitter in Counter Service (Server <-> View)
    constructor(private counterService: CounterService) {  // DI Injection
       this.counter = counterService.load();
    }
    ngOnInit() {
       this.counterSubscription = this.counterServices.countersChanged.subscribe(
                                     (data:CounterModel[]) => { this.counters = data; });
    }
    ngOnDestroy() {
       this.sampleSubscription.unsubscribe();
    } }
export class CounterModel {   constructor(public count:number = 0, public team:string = "unspecified") { }   }
```

Template //A template is a form of HTML that tells Angular how to render the component.

forbidden: <script>-Tag, Operators with side effects and chaining (++,--,new), Operator with different meanings (|, %,?)
One Way: `<p>... {{counter?.team}} ..</p>` oder `<img [attr.alt]="counter.team | uppercase" src="team.jpg">` // safe op. / pipes
Two Way: `<input type="text" [(ngModel)]="counter.team">` //needs FormsModule to work
One Way BacK: `<button (click)="counter.eventHandler($event)">`

Reference Variables: `<input placeholder="phone number" #phone> <button (click)="callPhone(phone.value)">`

Component Transclusion: `<wed-navigation> <h1 wed-title>WED3 Lecture</h1> <menu>... </menu> </wed-navigation>`
`<header><ng-content select='[wed-title]'> </ng-content> </header>`
`<nav> <ng-content select='menu'> </ng-content> </nav>`

Forms:
Template Driven: Simpler, Less JS Code, Useful for small forms (#myForm = "ngForm")
Reactive/Model Driven: Form build within Controller → Validation Logic Testable, Async Validation,

Directives //Attribute: alter appearance/behavior of elements //Structural: alter layout by DOM manip.

Attribute: `<div [class.special]="isSpecial">`   `[(ngModel)]="hero.name")`
Structural: `<div *ngIf="hasTitle"><div>`   `<li *ngFor="let element of elements"><!-- render element --></li>`

```
@Directive({ selector: '[wed-highlight]'}) // similar to a componente but without template
export class HighlightDirective {             // <span [wed-highlight]="organge"></span>
   constructor(private el: ElementRef) { }
   @Input("wed-highlight")
    public set color(color:string) { this.el.nativeElement.style.backgroundColor = this.color; }
    public get color() { return this.el.nativeEleent.style.backgroundColor; }  }
```

Pipes (pure = fires on change of bound member, impure = fires on every component change detection cycle (mouse move))

```
@Pipe({name: 'logo', pure: true})    // ctr.team | myPipe → executed on changes to ctr or ctr.team, not to ctr.abc
export class LogoPipe implements PipeTransform {
     private logos = { /*...*/ };
     transform(value: string, transformSettings: string): string {
        if (this.logos[value]) { return (this.logos[value][transformSettings] || this.logos[value].unspec); }
        return value;
     } }   // Impure pipes are executed on every component change detection cycle
```

Services // must be registered in Module or Component at least once as a provider
//use: For data/logic not associated with specific view, and shares across components

typical services: logging, data, tax calculator, stepper state. Register services in 'providers' attribute og ngModule

```
@Injectable()
export class CounterService {
    private counters: CounterModel[] = [];  // use EventEmitter to notify view about changes instead of RxJS
    public countersChanged:EventEmitter<CounterModel[]> = new EventEmitter<CounterModel[]>();
    constructor(private dataResource: CounterDataResourceService) {}
    load(): void {
        this.dataResource.get().subscribe( // subscribe for changes in the data source / web service
            (counters:CounterModel[]) => { this.counters = counters; this.countersChanged.emit(this.counters); });     } }
```

```
@Injectable()
export class AuthGuard implements CanLoad, CanActivate {
    constructor(private authService: AuthService, private navigationService: NavigationService) {  }
    canLoad(route: Route): boolean { if (this.authService.hasCredentials) { return true; } return false; }
    canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean { // prefer canLoad!
        if (this.authService.hasCredentials) { this.navigationService.goToDashboard(); return false;} return true;   }}
```

RxJS (Communication between Service and Data Access)

Hot Observables: Sequence of events (Mouse Move)
Cold Observables: Start running on subscription (Web Request)

```
@Injectable()
export class CounterDataResourceService {
    constructor(private http: HttpClient) { }
    get(): Observable<SampleModel[]> {
        return this.http.get('api/counters').pipe(map((data) => this.extractData(data)),
            catchError((err) =>this.handleError(err)));
        var subscription = this.http.get('api/counters').subscribe(
            function (x) { /* onNext -> data received (in x) */ },
            function (e) { /* onError -> the error (e) has been thrown */ },
            function () { /* onCompleted -> the stream is closing down */ } );
    }
    private extractData(data: any): CounterModel[] {
        return CounterModel.fromDto(data);
    }
    private handleError(err: HttpErrorResponse) {
      if (err.error instanceof ErrorEvent) { // a client-side or network error } else { // the backend returned an unsuccessful
response code
        }     }   }
```

Interceptor: Um die Headers der HTTP Request verändern zu können, kann im Modul Http-Interceptor registriert werden. (z.B Authorization Header, Content Type) → Request immer Klonen und dann verändern!

Router

- AppModule imports AppRoutingModule which imports *RouterModule* itself with the forRoot().
- Router uses a first-match-wins strategy when matching routes
- Clientseitiges Routing: Angular uses the browser's history.pushState for navigation
- It’s important to add a <head><base href=”/”></head> element to the app's index.html

```
<h1>WED3 - App Component</h1>
<nav><a routerLink="/welcome">Welcome Page</a></nav>
<router-outlet></router-outlet>
```

```
const appRoutes: Routes = [
 {path: 'register', component: RegisterComponent},  // feature component
 {path: sample, component: SampleComponent, children: [ {path: ‘a’, component: SamplesListComponent} ]}, // /sample/a
 {path: 'dashboard', loadChildren: 'app/dashboard/dashboard.module#DashboardModule', canLoad: [AuthGuard]}, // lazy
 {path: '', redirectTo: '/welcome', pathMatch: 'full'},
 {path: '**', redirectTo: '/welcome', pathMatch: 'full'} // add last to handle invalid URLs
];                                    // Optional: forRoot(appRoutes, { useHash: true })) → Hashtag Navigation
@NgModule({ imports: [ RouterModule.forRoot(appRoutes) ], exports: [ RouterModule ] }) class AppRoutingModule {  }
```

ASP.NET

ASP.NET verwendet einen Front Controller (Authentifizierung) welcher die Anfragen an die Page Controller dispatched.
Ein Request kann von mehreren Middleware bearbeitet werden. (Hin und Zurück)

Middleware:

```
app.Use(async (context, next) => { // New middleware
    System.Diagnostics.Debug.WriteLine("Handling request");
    await next.Invoke();
    System.Diagnostics.Debug.WriteLine("Finished handling request.");
});
App.Map(“/logging”, builder => {builder.Run(async (context) => {await content.Response.WriteAsync(“”} } // For path
App.Run(async(context) => { await context.Response.WriteAsync(“”);}  // Terminates Request
```

Dependency Injection and Middlewares:

```
public class Startup {
   public void ConfigureServices(IServiceCollection services) {
        services.AddTransient<IUserService, UserService>();        // DI Injection: Always a new instance every time you ask for it
        services.AddSingleton<IPersonService, PersonService>(); // DI Injection: Always same instance
        services.AddScoped<IUserService, UserService>();          // DI Injection: Instance is shared within a single request
        services.AddSession(options => { options.IdleTimeout = TimeSpan.FromMinutes(15); });
        services.AddDbContext<ApplicationDbContext>(options =>      // Entity Framework
                options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
        services.AddIdentity<ApplicationUser, IdentityRole>(options => {
                options.Password.RequireDigit = false;
                options.Password.RequireLowercase = false;
                options.Lockout.MaxFailedAccessAttempts = 3;
                options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(20);})
                .AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultTokenProviders();
         services.AddSwaggerGen();
}
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory) {
   app.UseMiddleware<UserMiddleware>();
    app.UseSession();
    app.UseMvc(routes => {
      routes.MapRoute(name: "default", template: "{controller=Home}/{action=Index}/{id:int?}");
      routes.MapRoute(name: "default2", template: "{controller}/{action}/{id?}", defaults: new {controller = "Home", action =
              "Index"}, constraints: new {id = new IntRouteConstraint()});
      });
     app.UseIdentity();
     app.UseSwagger(); app.UseSwaggerUI(options => {options.SwaggerEndpoint(“swagger.json”, “My API” });
}}
public class UserMiddleware {
   private readonly RequestDelegate _next;
   public RequestLoggerMiddleware(RequestDelegate next, ILoggerFactory loggerFactory) {
      _next = next;
   }
   public async Task Invoke(HttpContext context) {
      await _next.Invoke(context);
   } }
```

C#

Anonyme Typen: var v = new { Amount = 108, Message = "Hello" };

Extension Method:
public static int WordCount(this string str) { return str.Split(new char[] { ' ', '.', '?' }).Length; }

Dynamic Object : public dynamic CreateUser(string name) {

```
   dynamic person = new ExpandoObject();
   person.SayHi = new Action(() => Console.WriteLine(person.Name));
   person.Name = name;
   return person; }
```

Async/Await

```
static async Task Main(string[] args)
{
    Console.WriteLine("-----------");
    await RunAsync();
    Console.WriteLine("-----------");
    Console.ReadLine();
}

public static Task<string> Send()
{
    return Task.Run(() =>
    {
        System.Threading.Thread.Sleep(5000);
        Console.WriteLine("Send!");
        return "Nachricht gesendet";
    });
}

public static async Task<bool> RunAsync()
{
    Console.WriteLine("Start Send");
    Console.WriteLine(await Send());
    Console.WriteLine("End Send");
    return true;
}
```

Output: Start Send / Send! / Nachricht gesendet / End Send

Controller

- handles incoming URL requests. MVC routing sends requests to appropriate controller e.g. /student will be sent to StudentController
- all public methods = Action methods: Return ActionResult (baseclass of any possible return value (html, string, json,.))
- selector attributes determines, which action method is invoked.

```
public class HomeController : Controller {
   private readonly IPersonService _personService;
   public HomeController(IPersonService personService) { _personService = personService; } // Dependency Injection
  [AllowAnonymous]
  public ActionResult Index() {  // GET /home/index
     return View() / PartialView() / Content() / Empty() / File() / StatusCode() / Json() / Redirect() / RedirectToRoute,Action();
  }
 [HttpPost] [Authorize (Roles = "Admin,PowerUser") ]
 public ActionResult Create(Person person) {
   var user = await _userMng.GetUserAsync(User); _userMng.GetUserId(User); // Inject UserManager<ApplicationUser>
   if (ModelState.IsValid) {  // ViewBag,ViewData: dies after rendering view, TempData survives one redirect (needs session)
      ViewBag.Name = “Test” or ViewData[“Name”] = “Test2” or TempData[“Name”] = “Banana”;
      _personService.Add(person);                  //_db.Persons.Add(person); _db.SaveChanges();
      return PartialView("Person", person);       // RedirectToAction("");
   }
   return BadRequest(); / Content("Invalid Data");     } }
```

```
[Route("api/[controller]")]
public class ValuesController : Controller {
   [HttpGet("foo")]     // without / !!! Otherwise absolute path
   public IEnumerable<Value> Get() { return _valueService.All(); }
   [HttpGet("{id}")]    // ViewData: Controller Wide Dictionary → to transfer data from controller to view
   public Value Get(int id) { return _valueService.Get(id); ViewData["Key"] = "Value"; }
   [HttpPost]
   public void Post([FromBody | FromUri]Value value) { _valueService.Add(value); } } // default: primitives=Uri, class=Body
```

View (Razor Engine)

```
@{ var myMessage = "Hello World"; }
<p>The value of myMessage is: @myMessage</p>
@foreach(var member in members) { <li> @member </li> }
```

_Layout: <!DOCTYPE html><html><head><body>@RenderBody und @RenderSection(“Scripts”, required: false)</body>

```
@model Person //asp-action equals the activated action method in the controller
<form asp-controller="Demo" asp-action="Register" method="post"></form> // 1. default post form 2. ajax form
<form asp-action="Create" data-ajax="true" data-ajax-method="POST" data-ajax-mode="replace" data-ajax-update=
  "#result">
   <div asp-validation-summary="[ModelOnly | All | None]" class="text-danger"> </div>
   <div class="form-group">
       <label asp-for="Name" class="col-md-2 control-label"></label>
       <div class="col-md-10">
            <input asp-for="Name" class="form-control"/>
            <span asp-validation-for="Name" class="text-danger" />
        </div>
    </div>
    <a asp-controller=”Home” asp-action=”Index>Back to Home</a>
   <p> @ViewData["Key"] or @ViewBag.Key or @TempData[“Key”] or @Model.Key</p>
    <input type="submit" value="Do it!"/>
</form>
<div id="result"></div>
@section Scripts{    <script src="lib/jquery-ajax-unobtrusive/jquery.unobtrusive-ajax.min.js"></script> }
```

Router http://localhost:5000/Home/About Home = Controller, About = Action

```
app.UseMvc(routes => {                               Router-Engine unterstützten bei der Auswahl der Routes. → Attributes!!
  routes.MapRoute(
    name: "default",   //Name: Name der Route.
    template: "{controller}/{action}/{id?}",    // Template: Url-Pattern, ?: optional parameter
    defaults: new { controller = "Home", action = "Index" },
    constraints: new { id = new IntRouteConstraint() });    });
```

TagHelper // ermöglichen C# Code an HTML Tags zu binden.

```
public class EmailTagHelper : TagHelper {          usage: <email mail-for="support@example.com"></email>
  public string MailFor { get; set; }               after: <a href="mailto:support@example.com">support@example.com</a>
  public override void Process(TagHelperContext context, TagHelperOutput output) {
     output.TagName = "a"; // Replaces <email> with <a> tag
     output.Attributes.SetAttribute("href", "mailto:" + MailFor);
     output.Content.SetContent(MailFor);    } }
```

Entity Framework DbContext

```
public class ApplicationDbContext : DbContext {  DbContext is the entry point for CodeFirst approach via Type Discovery
  public virtual DbSet<Order> Orders { get; set; }
  public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { appsettings→connstr. }
  protected override void OnModelCreating(ModelBuilder builder) { base.OnModelCreating(builder); } }
```

Model (Entity)

```
public class Order {
  public [long/string] Id {get;set;} // implicit primary key, otherwise [Key]
  public [long/string] CustomerId { get; set; } // implicit foreign key
  [Range(30, 250)] [Display(Name = "Höhe in cm")]
  public double Height { get; set; }
  [RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]
  public string Email { get; set; }
  [Required] [StringLength(100, MinimumLength = 10)]
  public string Name { get; set; }
  public DateTime Date { get; set; }
  [NotMapped] public OrderState State { get; set; }  }
```

Unit Testing

Weshalb nicht direkt auf der echten Datenbank testen? → Multi-Threading Problem, Testdaten, Performance
Lösung: In Memory Datenbank oder DbContext Mocken

JWT Token Übertragung: HTTP-Header: Authorization: Bearer <token>

Struktur: Header, Payload (beinhaltet user daten), Signatur
Ablauf: POST password to server → server creates JWT → client sends requests with JWT → server checks JWT

Swagger (Alternatives: RAML, GraphQL)

+ Interactive Documentation, +Auto API Generation, + Debugging/Testing + Multiple Programming Languages, + API Dokumentation Nahe beim Code, + Tools (UI und Codegen)

SVG Default Grösse:300px*150px, Ursprung oben link

+ Flexible, + CSS Styles + JS Event Handling, + for Animations, Graphics, Charts, simple Games - Performance

Einbinden im Browser: <svg>, <object>, <img> → verlieren Interaktionsmöglichkeiten
SVG hat eigenes Koordinatensystem → Grösse muss angeben werden → ViewBox für Grössenverhältnisse
<svg preserveAspectRatio="…"> → dfiniert Verhalten bei einem Verhältnis-missmatch
<svg><style></style><svg>: Media-Queries, Animations, etc

```
<svg viewBox="0 0 200 200">  // x y width height            <g>….</g> // group svg elements
   <style> .alert{ fill: red; } </style> // Polygon schliesst das geometrische Objekt immer ab (Polyline nicht)
    <rect x="0" y="0" width="200" height="200"></rect>    <line x1="0" y1="0" x2="200" y2="200"/>
    <circle r="50" cx="50" cy="50" class=”alert”></circle>  <ellipse rx="20" ry="6" cx="43" cy="56" />
    <polygon points="200,10 250,190 160,210"/> <polyline points="20,20 40,25 60,40 80,120" /> </svg>
```

Path <path d="M 100 100 L 300 100 L 200 300 z" fill="orange" stroke="black" stroke-width="3" />

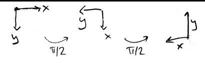
M x,y	→ Move to the absolute coordinate x,y
m x,y	→ Move to the right x and down y (or left and up if negative values)
L x,y	→ Draw a straight line to the absolute coorinates x,y
l x,y	→ Draw a straight line to a point that is realtively right x and down y (or left and up if negative)
H x	→ Draw a line horizontally to the excact coordinate x
h x	→ Draw a line horizontally relatively to the right x (or to the left if a negative value)
V y	→ Draw a line vertically to the exact coordinate y
v y	→ Draw a line vertically relatively down y (or up if a negative value)
Z (z)	→ Draw a straight line back to the start of the path

Canvas

+Performance,+JS,+Browser Support,+Pixel Support,-Accessiblity,-Event Handling,- No Layers, -Manual Animations

```
<canvas id="painting" width="600" height="600"> Hello World Demo // HTML Fallback if no canvas support </canvas>
<script>
    var painting = document.getElementById("painting");
    if(painting.getContext) {
        var ctx = painting.getContext("2d");
        painting.height = window.innerHeight; painting.width = window.innerWidth;
        ctx.fillRect(0, 0, 300, 150);
        ctx.beginPath(); ctx.arc(150, 100, 50, 0, Math.PI); ctx.moveTo(150,200); ctx.lineTo(200,250); ctx.stroke(), ctx.fill();
         // ctx.arc(centerX,centerY,radius,startangle,endangle, counterclockwise); //angle 0 → x axi
        ctx.beginPath();ctx.ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle, anticlockwise);ctx.stroke();
        // for whole ellipse, choose startAngle=0, endangle=2*Math.PI
    }
</script>  //ctx.restore() setzt zustand auf Zeitpunkt von ctx.save() zurück
```

ctx.translate(50,50); // move object   ctx.scale(2,4);
ctx.rotate(Math.PI); // rotate whole coordinate system for future drawings, earlier translate not affected!
Rotate and Translate != Translate and Rotate



Double Buffering: Kontinuirliche Bildfrequenz ohne Flackern: Paint Canvas 2 in Background and Swap!

Pre-Rendering: (DRY) Paint Objects in offscreen Canvas. Wieverwenden des vorgezeichneten Canvas