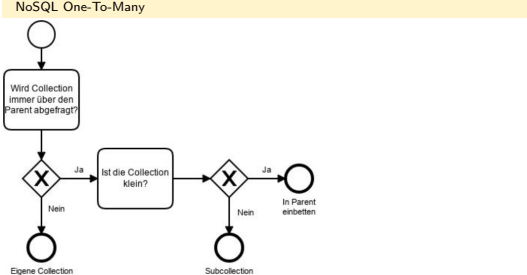


| |
|--|
| |
| Selectors |
| Getter bei den Reducern, die einen Subtree des Stores zurückgeben. Wissen über den Aufbau des State-Trees bleibt bei den Reducern. |
| Firebase |
| Läuft in der Google Cloud Platform. Hauptfokus von Firebase sind Mobile- und Web-Apps. |
| Firebase Authentication |
| Backend Services für Authentifizierung und einfache Userverwaltung. SDKs für diverse Plattformen. Vorgefertigte UI Libraries |
| Firebase Hosting |
| Einfaches Hosting für statischen Content. |
| <ul style="list-style-type: none"> Immer per HTTPS ausgeliefert Automatisches Caching in CDNs |
| Dynamischer Content nur über Cloud Function , wenn das nicht reicht: |
| <ul style="list-style-type: none"> PaaS: Google App Engine Docker: Google Container oder Kubernetes Engine |
| Serverless Computing |
| Cloud Provider verwaltet Functions: |
| <ul style="list-style-type: none"> Deployment geschieht on-demand Plattform bestimmt die Parallelisierung Entwickler hat keine Kontrolle über laufende Instanzen Funktionen sind Stateless Abgerechnet werden Aufrufe und Laufzeit der Funktion |
| Limitationen: Ausführungszeit / Memory begrenzt. Teilweise hohe Latenz. |
| Firebase Cloud Functions |

Anwendungsszenarien: Code als Reaktion auf einen Event ausführen, Administration (Crn Jobs), REST API für Mobile und SPAs zur Verfügung stellen.

| |
|--|
| Cloud Firestore |
| <ul style="list-style-type: none"> NoSQL, document-oriented database DB besteht aus mehreren Collections mit Documents Document ist ein JSON-Objekt Document kann Collections beinhalten Vergleichbar mit MongoDB Stark eingeschränkte Queries (keine Volltextsuche) |
| <pre>// Auf Collections / Documents zugreifen const colRef = db.collection("todos"); const docRef = db.collection("todos").doc("..."); // Dokumente erstellen db.collection("todos").add({text: "..."}); // Dokument bearbeiten .doc("...").update({text: "..."}); // Daten Abfragen db.collection("todos").doc("...").get().then(d => { if(!doc.exists) { /* ... */ } else { console.log(d.data()); } }).catch(err => { /* ... */ }); // Daten abfragen mit Filter db.collection("todos").where("checked", "==", true) .orderBy("createdAt").get().then(snapshot => { // ... });</pre> |
| NoSQL One-To-Many |



- | |
|--|
| NoSQL Many-To-Many |
| <ul style="list-style-type: none"> Wie in relationaler Datenbank mit Assoziationstabelle <ul style="list-style-type: none"> Kein kopieren von Daten Komplexere Abfragen, keine Joins im Firestore Oder Daten kopieren und einbetten |

Kopieren der Daten: muss kein Nachteil sein. Preisänderung eines Produktes hat keinen Einfluss auf vergangene Bestellungen. Adressänderung eines Kunden verändert keine alten Bestellungen. Kopierte Daten können mittels Trigger und Cloud Function wieder synchronisiert werden.

| |
|---|
| Angular |
| Flexible SPA Framework für CRUD applications |
| <ul style="list-style-type: none"> Typescript 4.1 based Reduces boilerplate Code Dependency Injection Mechanism JS-optimized 2-way binding Clearly structured, information hiding Increases testability / maintainability of client-side code |

| |
|---|
| Architecture |
| ngModules: Cohesive block of code dedicated to closely related set of capabilities. (<i>module</i>) Directives: Provides instructions to transform the DOM. (<i>class</i>) Components: Directive-with-a-template; it controls a section of the view. (<i>class</i>) Templates: Form of HTML defining how to render the component. (<i>HTML / CSS</i>) Metadata: Describes a class and defines how to process it. (<i>decorator</i>) Services: Provides logic of any value, function or feature that the app needs. (<i>class</i>) |
| Angular Modules (ngModule) |
| Base for Angular modularity system. Every app has at least one Module, the root Module (a.k.a app). Root Module ist launched to bootstrap the app. Modules export features (directives, services) required by other modules. |
| TypeScript Module vs. ngModule: ngModule is a logical block of multiple TypeScript modules linked together. The ngModule declaration itself is placed into a TypeScript module. Modules can accommodate sub-modules. All public TS members are exported as an overall <i>barrel</i> |

| |
|--|
| @NgModule({ |
| imports: [|
| CommonModule |
|], |
| declarations: [] |
| }) |
| export class CoreModule { } |
| declarations: View Classes that belong to this module (Components, Directives, Pipes). |
| exports: Subset of declarations that should be visible and usable by other modules. |
| imports: Specifies the modules which exports/providers should be imported. |

- forChild-Import: returns an object with a **providers** and **ng-Module** property
 - allows you to configure services for the current Module level
 - Use if you need to configure the foreign module
- forRoot-Import: returns an object with a **providers** and **ng-Module** property
 - It provides and condigures services at the same time
 - Will instantiate the required services exactly once, globally
 - If no configuration is required, use tree shakable providers { *providedIn: 'root'* }
 - If no configuration is required, use tree shakable providers { *providedIn: 'root'* }

providers: Creators of services that this module contributes to the global collection of services (DI Container). They become accessible in all parts of the app.

bootstrap: Main application view, root component. Only the root module should set this property.

| |
|--|
| Module Types |
| Root / App Module: Provides the entry point (bootstrap component) for the app. Has no reason to export anything. Feature Modules: Specifies boundaries between app features. |
| <ul style="list-style-type: none"> Domain Modules: Deliver a UI dedicated to a app domain Routing Modules: Specifies routing configurations Service Modules: Provides utility services Widget Modules: Makes components, directives and pipes available to external modules Lazy Modules: Lazily loaded feature modules |

Shared Modules: Provides globally used components/directives/pipes. Is a global UI component module. Do not specify app-wide singleton providers in a shared module. **Core Module:** Keeps your Root Module clean. Contains components/directives/pipes used by the Root Module. Globally used services can be declared here. Only imported by the Root Module

| |
|---|
| Components |
| Manages the view and binds data from the model. Consists of: <ul style="list-style-type: none"> Controller (App logic), TS Class with <i>@Component</i> decorator HTML file, visual interface (HTML / template expression) (S)CSS file, styles behind HTML |
| Can be nested, results in Component tree. |
| Provide Information Hiding: <ul style="list-style-type: none"> Each Component declares part of the UI Should be implemented as small coherent piece to support: <ul style="list-style-type: none"> Testability, Maintainability, Reusability |

| |
|--|
| import { Component } from '@angular/core'; |
| @Component({ |
| selector: 'wed-navigation', |
| templateUrl: './view@tmn.component.html', |
| styleUrls: ['./ngcss@css.component.css'] |
| }) |
| export class NavigationComponent { |
| // ... |
| } |
| Logic (TypeScript) |
| navigation.component.html |
| CSS { |
| } |
| navigation.component.css |
| Components must be declared within the containing module so its selector is registered for all sub-components of that module. They can be exported, so other modules can import and use them. |
| Component Lifecycle |
| Most important events are ngOnInit (Creation / Hydration) and ngOnDestroy (Destruction / Dehydration). ngAfter... events are mainly for control |

| |
|---|
| developers to handle sub-components and their DOM. To hook into the lifecycle, interfaces of the Angular core can be implemented. Each interface has a single hook method, prefixed with ng . (OnInit contains method <i>ngOnInit</i>). |
| export class CounterComponent implements OnInit, OnDestroy { |
| ngOnInit() { |
| console.log("OnInit"); |
| } |
| ngOnDestroy() { |
| console.log("OnDestroy"); |
| } |

| |
|---------------------------------|
| Content Projection |
| <!-- component usage --> |
| <section> |
| <wed-navigation> |
| <h1 wed-title>WED3 Lecture</h1> |
| <menu><!-- ... --></menu> |
| </wed-navigation> |
| </section> |
| <!-- resulting DOM --> |
| <section> |
| <wed-navigation> |
| <header> |
| <h1 wed-title>WED3 Lecture</h1> |
| </header> |
| <nav> |
| <menu><!-- ... --></menu> |
| </nav> |
| </wed-navigation> |
| </section> |
| navigation.component.html |
| Decorator: @Component decorator |

| |
|--|
| Templates |
| View in MVC. Written in HTML annotated with Angular template syntax: |
| <ul style="list-style-type: none"> HTML5 except script-Tag Angular extends the HTML with <ul style="list-style-type: none"> Interpolation (...) Template Expression/Statements Binding Syntax Directives Template Reference Variables Template Expression Operators |

| |
|--|
| Binding |
| Two Way Binding / Banana in a box ([...]) |
| <input type="text" [(ngModel)]="counter.team"> |
| @Component({...}) |
| export class CounterComponent { |
| public counter: any = { |
| get team(): { return ...; }, |
| set team(val) {}, |
| eventHandler(): =>{ } |
| } |
| } Model Object |
| One Way (from View to Model / Event Binding) (...) |
| <button (click)="counter.eventHandler(\$event)"> |
| One Way (from Model to View / Property Binding) [...] or [...] ...] |
| <p>... {{counter.team}} </p> |
| |
| Binding targets must be declared as Inputs or Outputs : Targets stand on the left side of the binding declaration. e.g. the <i>click</i> / <i>title</i> property: <wed-navigation (click)="..." [title]="..."> |
| @Component({...}) |
| export class NavigationComponent { |
| @Output() click = |
| new EventEmitter<any>(); |
| } @Input() title: string; |
| } |

| |
|--|
| Directives |
| Similar to a component, but without a template. TypeScript class with an <i>@Directive()</i> function decorator. |
| Attribute Directives |

| |
|---|
| Changes the appearance or behaviour of an element, component or another directive. Applied to a host element as an attribute. |
| NgStyle Directive |
| Sets the inline styles dynamically, based on the state of the component. |
| <div [style.font-size]="isSpecial ? 'x-large' : 'smaller'"> |
| <!-- render element --> |
| </div> |
| NgClass Directive |
| Bind to the ngClass directive to add or remove several classes simultaneously. |
| <div [class.special]="isSpecial"> |
| <!-- render element --> |
| </div> |

| |
|---|
| Structural Directives |
| Responsible for HTML layout. Reshape the DOM's structure by adding, removing or manipulating elements. Applied to a host element as an attribute. Asterisk (*) precedes the directive attribute name. |
| NgIf Directive |
| Takes a boolean value and makes an entire chunk of the DOM appear or disappear. |
| <div *ngIf="hasTitle"><!-- shown if title available --></div> |
| NgFor Directive |
| Represents a way to present a list of items. |
| <li *ngFor="let element of elements"><!-- render element --> |
| NgTemplates: |
| <ng-template #toReference><!-- content --></ng-template> |
| Aren't rendered directly. They need a directive or component |

which takes over this part. Can be referenced by their *#id*: <div *ngIf="hasTitle; else toReference"><!-- conditional content --></div>

| |
|--|
| Template Reference Variables |
| References a DOM element within a template. Can also be a reference to an component or directive. A hash (#) declares a reference variable. |
| <input placeholder="phone number" #phone> |
| <!-- lots of other elements --> |
| <!-- phone refers to the input element; pass its 'value' to an event handler --> |
| <button (click)="callPhone(phone.value)">Call</button> |
| Services |
| Provides any value, function or feature. Typical Services: logging service, data service, message bus, tax calculator, etc. Strongly related to DI: Angular uses DI to provide components with needed services. Therefore, services must be registered within the DI container. |
| @Injectable ({ providedIn: 'root' }) |
| export class CounterService { /* ... */ } |

| |
|---|
| providedIn: 'root': The service is registered for the whole application. |
| Decorator function to mark class as an injectable instance |
| @Injectable({ providedIn: 'root' }) |
| export class CounterService { |
| private model: CounterModel; |
| = new CounterModel(); |
| constructor(private counterService: CounterService) { |
| this.counter = counterService.load(); |
| } ... |
| } Required services (dependencies) are automatically injected by Angulars injector. |
| @Component({...}) |
| export class CounterComponent { |
| counter?: CounterModel; |
| } |

| |
|--|
| Forms |
| Angular Forms is an external, optional ngModule called FormsModule. It's a combination of multiple provided services and multiple directives (ngModule, ngForm, ngSubmit). |
| Template-driven forms: Angular Template syntax with the form-specific directives and techniques. Less code but places validation logic into HTML. (Useful for small forms) |
| Reactive / model driven forms: Import ReactiveFormsModule. Form is built within the Controller (FormBuilder). Validation logic is also part of the controller (easier to test). |

| |
|--|
| Template-driven |
| <input type="text" class="form-control" id="name" required (ngModel)="model.name" names="name" #nameField="ngModel"> |
| <div [hidden]="nameField.valid nameField.pristine" class="alert alert-danger"> Name is required |
| </div> |

Two-Way-Binding: [(ngModel)] directive to bind values. Reads out the value of the model for the first time. Updates are automatically written back into the bound model.

Validation: Reference the [(ngModel)] directive and check its valid property.

| |
|---|
| Submitting the form: |
| ngForm can also be referenced |
| <ul style="list-style-type: none"> This is useful to bind validation state on the submit button or pass the form to the submit method |
| @Component({ ... }) |
| export class SampleComponent { |
| public doLogin((): NgForm): boolean { |
| if (!f.frm.valid) { // store data |
| return false; // avoid postback |
| } |
| } |
| <form (ngSubmit)="doLogin(sampleForm)" #sampleForm="ngForm"> |
| <button type="submit" class="btn btn-success" [disabled]="!sampleForm.frm.valid">Submit</button> |
| </form> |
| Asynchronous Services |

| |
|---|
| Event Emitter example: |
| @Injectable(providedIn: 'root') |
| export class SampleService { |
| private samples: SampleModel[] = []; // simple cache |
| public samplesChanged: EventEmitter<SampleModel[]> = new EventEmitter<SampleModel[]>(); |
| constructor() { /* inject data resource service */ } |
| load(): void { |
| /* in real word app, invoke data resource service here */ |
| this.samples = [new SampleModel()]; |
| this.samplesChanged.emit(this.samples); |
| } |
| } Emit changed event to notify the registrars (e.g. UI components). |
| export class SampleModel { } |

```
@Component({ ... })
export class SampleComponent implements OnInit, OnDestroy {

  private samples: SampleModel[];
  private samplesSubscription: Subscription;
  constructor(private sampleService: SampleService) {
    // Subscription is used to unsubscribe
    // the update event when the
    // component is de-hydrated.
  }

  ngOnInit() {
    // Register samplesChanged event on underlying business service when component
    // is hydrated. Subscribe() returns a Subscription which is used for deregistration.
    this.samplesSubscription = this.sampleService.samplesChanged.subscribe(
      (data: SampleModel[]) => { this.samples = data; });
  }

  ngOnDestroy() {
    // Update procedure; refresh data on
    // the UI level.
    this.samplesSubscription.unsubscribe();
    // Unsubscribe the update event when
    // the component is de-hydrated.
  }
}
```

Data Access

HTTP Client API

Implements asynchronisms by using the RxJS library. RxJS is a third-party library that implements the Observable pattern. An Observable can be turned into a promise.

Hot Observables: Sequences of events (mouse moves / stock tickers). Shared amongst all subscribers. Postfix hot-observables with a \$

Cold Observables: Start running on subscriptions (such as async web requests). Not shared amongst subscribers. Are automatically closed after Task is finished.

```
var subscription = this.http.get('api/samples').subscribe(
  function (x) { /* onNext -> data received (in x) */ },
  function (e) { /* onError -> the error (e) has been thrown */ },
  function () { /* onComplete -> the stream is closing down */ }
);
```

Routing

External, optional NgModule called RouterModule. Combination of multiple provided services and directives: RouterOutlet, RouterLink, RouterLinkActive.

Defining Routes: The router must be configured with a list of route definitions. Each definition maps a route to a component.

- `.forRoot()`: use exactly once to declare routes on root level
 - contains all the directives, the given routes and the router service itself
 - Every app has one singleton instance of the router
- `.forChild()`: When declaring sub-routings
 - contains all directives and the given routes

Each NgModule defines its own routes. Load modules on-demand (lazy load) with the `import`-Syntax.

```
@NgModule({
  imports: [ RouterModule.forRoot(appRoutes) ],
  exports: [ RouterModule ]
})
export class AppRoutingModule {}

@NgModule({
  imports: [ RouterModule.forChild(welcomeRoutes) ],
  exports: [ RouterModule ]
})
export class WelcomeRoutingModule {}
```

Router Outlet: Directive from the Router module. Defines where the Router should display the views.

```
<router-outlet></router-outlet>
```

Route Configuration:

```
const appRoutes: Routes = [
  // matches /hero/42, 42 saved in param
  {path: 'hero/:id', component: 'Hero'},
  // redirect
  {path: '', redirectTo: '/heroes', pathMatch: 'full'},
  // Wildcard route
  {path: '**', component: PageNotFound}
];
```

The router uses a first-match-wins strategy.

```
Lazy Loading Configuration
{ path: 'config',
  loadChildren: () => import('./cfg/cfg.module').then(m => m.CfgModule),
  canActivate [ AuthGuard ] }
```

Angular Architectures

MVC+S

Observable Business Data Service: Provides data to multiple parts of the app in a stream-like manner. An *Observable* is provided. Stores/Caches business objects.

RxJS Subject: Heart of an observable data service. *EventEmitterTs* derives from Subject. Hot Observable and does not provide the latest value.

Behaviour Subject: Emits the initial state. Can be called some kind of warm. Stores the data and emits *next()* events on change. Do not expose to the Service API.

Data Resources: Return cold Observables. Must be converted into a hot Observable (*share()*).

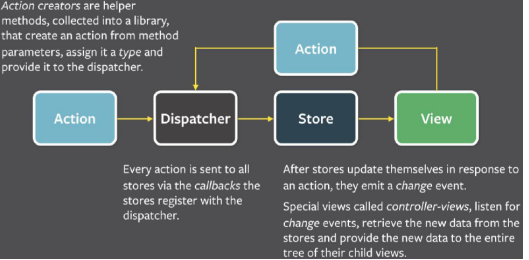
| Observable | Business | Data | Service | Example: |
|---|----------|------|---------|----------|
| <pre>@Injectable({providedIn: 'root'}) export class SampleService { private samples: BehaviorSubject<SampleModel[]> = new BehaviorSubject([]); public samples\$: Observable<SampleModel[]> = this.samples.asObservable(); constructor(private resourceService: SampleResourceService) { }</pre> | | | | |
| Event bus which is used to store the last state and to notify subscribers about updates. | | | | |
| Postfix hot-observables (streams) with a \$. | | | | |
| Convert event bus into an observable, which can be provided to the UI or other services. | | | | |

```
public addSample(newSample: SampleModel): Observable<any> {
  return this.resourceService
    .post(newSample)
    .pipe(
      tap(res => {
        this.samples.next([...this.samples.getValue(), newSample]);
      }),
      catchError((err) => this.handleError(err));
    );
}

private handleError(err: HttpErrorResponse): Observable<any> {
  // Store the retrieved data into the BehaviorSubject and emit the
  // data changed event. It is important to create a new array with
  // the new data; otherwise, the async pipe won't track the change.
}
```

Flux Architecture

Invented by Facebook. Enforces a unidirectional data flow. More of a pattern than a formal framework.



Redux Architecture

ngrx: implements the Redux pattern using RxJS. **Benefits:**

- Enhanced debugging, testability and maintainability
- Undo/redo can be implemented easily
- Reduced code in Angular Components

Liabilities:

- Additional 3rd party library required
- More complex architecture
- Lower cohesion, global state may contain UI / business data
- Data logic may be fragmented into multiple effects/reducers

UI Advanced

Pipes

Can be applied within a template expression to make small transformations.

```
<p>{{counter.team | uppercase}}</p>
<p>{{counter.team | uppercase | lowercase}}</p>
<p>{{counter.date | date: 'longDate'}}</p>
```

Pure-Pipes: Executed when it detects a pure change to the input expression. Implemented as pure functions. Restricted but fast.

Impure-Pipes: Executed on every component change detection cycle (every keystroke etc.). To reduce processing time, caching is often used.

Predefined-Pipes: *date*, *number*, *currency*, *async* etc. Angular does not provide Filter- / OrderBy-Pipes because of poor performance.

Custom-Pipes: A class decorated with *@Pipe()*. It implements the *PipeTransform* interface's *transform()* method. Needs to be added to the declarations of the current Module.

```

```

Specifies arguments to be passed to the transform() method.

```
@Pipe({name: 'logo', pure: true})
export class LogoPipe implements PipeTransform {
  private logos = { /*...*/ };
  transform(value?: string, transformSettings?: string): string {
    if (value && transformSettings && this.logos[value]) {
      return (this.logos[value][transformSettings] || this.logos[value].unspec);
    }
    return value;
  }
}
```

Async Pipes: Binds Observables directly to the UI. Changes are automatically tracked. Automatically subscribes and unsubscribes from the bound Observable.

```
<div>
  <section>
    <li *ngFor="let s of sampleService.samples$ | async">
      <ul>{{s.name}}</ul>
    </li>
  </section>
</div>
```

@Component({...})

```
export class CounterComponent {
  constructor() {
    // public sampleService: SampleService()
  }
}
```

View Encapsulation

Component Styles: Apps are styled with standart CSS. The CLI transpiles SCSS to CSS. The selectors of a component's styles apply only within this own template.

Special Selectors:

- `:host` - Target styles in the element that hosts the component
- `:host-context` - Looks for a CSS class in any ancestor of the host element

Link Styles to Component - Options:

- Add a styles array properly to the *@Component* decorator
 - Add *styleUrls* attribute into a components *@Component* decorator
 - Template inline tags/styles
- ```
@Component({ /*...*/
 styleUrls: ['app/nav.component.css']
})
export class WiedNavComponent { }
```
- ```
@Component({ /*...*/
  template: `<style>...</style> ... <link rel="stylesheet" href="app/nav.component.css">`
})
export class WiedNavComponent { }
```

Controlling View Encapsulation:

- Native:* Uses the browsers native shadow DOM
- Emulated:* Emulates the behaviour of shadow DOM by preprocessing (and renaming) the CSS
- None:* No view encapsulation (scope rules) applied. All CSS added to the global styles.

PWA & Angular & Firebase