

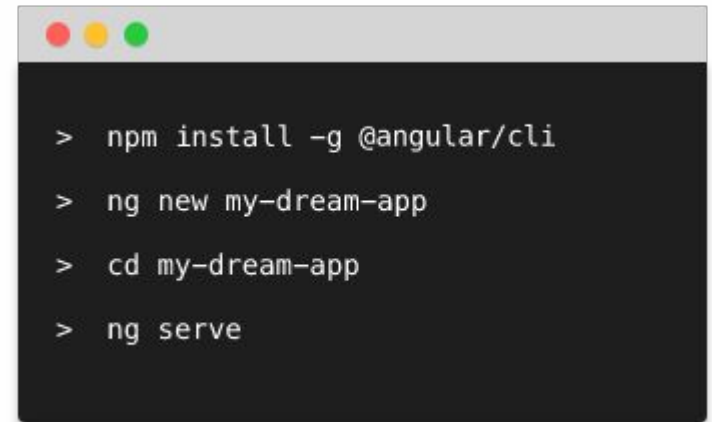
# Intro til Angular



# ANGULAR

# Tooling

- Node  
<https://nodejs.org/en/>
- NPM  
<https://www.npmjs.com/>
- Angular  
<https://cli.angular.io/>
- ???
- Profit

A terminal window with a dark background and light gray text. It shows a series of commands being entered at a prompt. The commands are: 'npm install -g @angular/cli', 'ng new my-dream-app', 'cd my-dream-app', and 'ng serve'. The window has a standard macOS-style title bar with red, yellow, and green buttons.

```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

CLI

Læs <https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f> for kicks



# Style guide

---

Angulars style guide er et “must read”

<https://angular.io/docs/ts/latest/guide/style-guide.html>

Den giver ikke så meget værdi at læse fra ende til anden uden et godt kendskab til Angular. Så i stedet brug den som “Just-In-Time learning”. Dvs. skal du lave en service eller component så slå op i style guiden hvad “best practices” er, og derefter implementer det.

# Component

Et component består af et template (view) en klasse (controller) og noget metadata (prefixed med @, tænk på dem som C# Attributter) som binder det sammen og andet konfig.

## Component class:

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'app works!';
}
```

## Template view:

```
<h1>
  {{title}}
</h1>
```

# OnInit

Et component har et lifecycle hook kaldt OnInit og denne skal bruges til at lave “complex initialization” af dit component. Dette kunne evt. være at hente data fra en backend.

OnInit bliver kaldt når component’et bliver vist, ikke når det bliver oprettet. Vigtig forskel.

Og det bruges således:

```
@Component({selector: 'my-cmp', template: `...`})
class MyComponent implements OnInit {
  ngOnInit() {
    // ...
  }
}
```

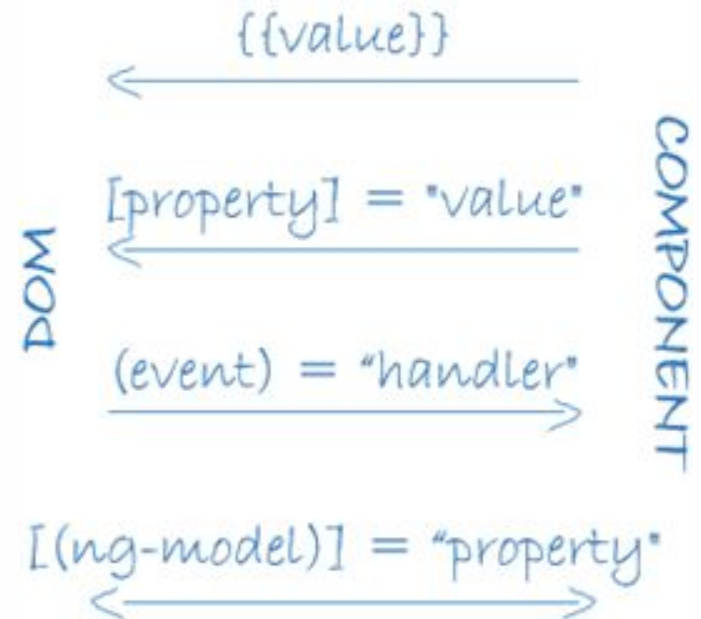
Der findes flere lifecycle hooks:

<https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html>

# Angular Templates & Data binding

```
<h1>{{title}}</h1>
<h2>My favorite hero is: {{myHero.name}}</h2>
<p>Heroes:</p>
<ul>
  <li *ngFor="let hero of heroes">
    {{ hero.name }}
  </li>
</ul>
<p *ngIf="heroes.length > 3">There are many heroes!</p>

<input [(ngModel)]="title">
<button (click)="setTitle()">Set default title</button>
```



# Module

Et module beskriver hvordan applikationens dele passer sammen. Alle applikationer skal have et root module.

```
@NgModule({
  declarations: [ // hvilke components, pipes, directives... deklarerer dette module
    AppComponent
  ],
  imports: [ // hvilke andre moduler afhænger dette module af
    BrowserModule,
    FormsModule,
    HttpClientModule,
    AppRoutingModule
  ],
  providers: [], // hvilke services udstiller dette module, bruges i forbindelse med DI
  bootstrap: [AppComponent] // start component
})
export class AppModule { }
```

# Services

Services bruges til at uddelegere ansvaret fra fx components. Man definere en services således:

```
@Injectable()
export class FooService {}
```

En data service kunne se således ud:

```
@Injectable()
export class HeroService {
  private heroesUrl = "app/heroes.json";

  constructor(private http: Http) { }

  public getHeroes(): Observable<Hero[]> {
    return this.http.get(this.heroesUrl).map(response => response.json());
  }
}
```

Husk at tilføje din service til providers i det module der skal bruge det. Hvis du bruger CLI'en og tilføjer --module <modulename> så gør den det for dig.



# Routing

Routes defineres således. (forRoot er kun på root module, ellers forChild)

```
RouterModule.forRoot([  
  { path: '', redirectTo: '/heroes', pathMatch: 'full' },  
  { path: 'heroes', component: HeroesComponent }  
])
```

Bruger du CLI'en med --routing så får du generet næsten alt. Du skal blot udfylde routes arrayet.

Man definere hvor i viewet indholdet fra en route skal sættes ind via

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

Man definere links via routerLink directive'et

```
<a routerLink="/heroes" routerLinkActive="active">Heroes</a>
```

Routes kan indeholde parameter ved at prefix “placeholderen” med :

```
{ path: 'hero/:id', ...
```

# Architecture

Del applikationen op i mindre feature modules som fungerer som selvstændige angular apps.

Forsøg at overholde LIFT (Locate code quickly, Identify the code at a glance, keep the **Flattest** structure you can, and **Try** to be DRY).

Basalt set bare følg styleguiden.

```
app
  core
    core.module.ts
    exception.service.ts|spec.ts
    user-profile.service.ts|spec.ts
  heroes
    hero
      hero.component.ts|html|css|spec.ts
    hero-list
      hero-list.component.ts|html|css|spec.ts
    shared
      hero-button.component.ts|html|css|spec.ts
      hero.model.ts
      hero.service.ts|spec.ts
    heroes.component.ts|html|css|spec.ts
    heroes.module.ts
    heroes-routing.module.ts
  shared
    shared.module.ts
    init-caps.pipe.ts|spec.ts
    text-filter.component.ts|spec.ts
    text-filter.service.ts|spec.ts
  app.component.ts|html|css|spec.ts
  app.module.ts
  app-routing.module.ts
```



# Tour of Heroes



Jeg vil kraftigt opfordre jer til at følge Tour of Heroes tutorialen. Den vil tage jer igennem de vigtigste aspekter af Angular.

<https://angular.io/docs/ts/latest/tutorial/>

# Debugging

<https://augury.angular.io/>

The screenshot displays the Angular Augury debugging tool interface. The top navigation bar includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Audits, and Augury. The Augury tab is active, showing a sub-menu with Component Tree, Router Tree, and NgModules. The Component Tree on the left shows a hierarchy starting with AppComponent, which contains a (text="Heroes", hash=""), router-outlet (name=""), and HeroesComponent. The HeroesComponent is selected and highlighted in purple. The right panel shows the Properties and Injector Graph for the selected HeroesComponent. It includes a link to View Source, a note that it is the first instance in the console, and a Change Detection status of Default. The State section is expanded, showing heroService as HeroService and heroes as an array of two objects. The first object has a name property set to 'foo'. The Dependencies section is also visible but collapsed.

```
▼ AppComponent
  a (text="Heroes", hash="")
  router-outlet (name="")
  HeroesComponent
```

Properties | Injector Graph

HeroesComponent (View Source) (1st in Console)

Change Detection: Default

▼ State

- ▶ heroService: HeroService
- ▼ heroes: Array[2]
  - ▼ 0: Object
    - name: foo
  - ▶ 1: Object

▶ Dependencies