Dependency Injection

En andere *hippe* Design Patterns

Design patterns

- Al bekend met design patterns?
- Vanuit opleiding
- Leuk maar met mate(n)
- Classes

Classes: Lightning edition

```
// Blauwdruk voor ding
class Ding {

    public score = 1;

    logScore() {
        console.log('Mijn score is ', this.score)
    }

}
```

```
// Nope, Ding = blauwdruk
Ding.logScore()
```

```
const lekker = new Ding();
lekker.logScore(); // Mijn score is 1
```

```
lekker.score = 10;
lekker.logScore(); // Mijn score is 10
```

```
const aanvaar = new Ding();
aanvaar.logScore(); // Mijn score is 1
lekker.logScore(); // Mijn score is 10
```

Interfaces (/ inheritance)

```
class Bier implements AlcoholischeVersnapering {
    getPromillage(): number {
        return 5;
    }
}
class Wodka implements AlcoholischeVersnapering {
    getPromillage(): number {
        return 40.0;
    }
}
}
```

```
class JouwBaasOpWerk {
    vindAcceptabelOpWerk( glas: AlcoholischeVersnapering ) {
        return glas.getPromillage() < 10;
    }
}
```

```
const baas = new JouwBaasOpWerk();
const glas = new Baileys();
if ( ! baas.vindAcceptabelOpWerk(glas)) {
    console.error('Moeten wij even op gesprek?')
}
```

```
interface AlcoholischeVersnapering {
    getPromillage(): number;
}
```

```
class Baileys implements AlcoholischeVersnapering {
    getPromillage(): number {
        return 17.0;
    }
}
```

Singleton: “Er mag er maar 1 van zijn”

```
class Koelkast {
    koud: boolean = false;
}
```

```
const kast = new Koelkast();
kast.koud = true;
```

```
function pakBier() {
    const kast = new Koelkast();
    if (kast.koud) { // Altijd false
        console.log('Lekker zeg');
    }
}
```

Singleton: “Er mag er maar 1 van zijn”

```
class Koelkast {
    koud: boolean = false;
}
```

```
function pakBier() {
    const kast = Koelkast.instance;
    if (kast.koud) {
        console.log('Lekker zeg');
    }
}
```

```
class Koelkast {
    koud: boolean = false;

    static instance = new Koelkast();
    private constructor() {}
}
```

```
const kast = Koelkast.instance;
kast.koud = true;
```

Dependency Injection (context)

```
class MailService {

    private quoteApi: QuoteApi;

    constructor() {
        this.quoteApi = new KanyeQuoteApi();
    }

    sendMail() {
        const quote = this.quoteApi.getQuote();
        mailTo( who: 'leon', text: 'Check deze quote: ' + quote);
    }

}
```

```
interface QuoteApi {
    getQuote(): string;
}
```

```
const mail = new MailService();
mail.sendMail();
```

Dependency Injection (context)

```
class MailService {

    private quoteApi: QuoteApi;

//  constructor() {-
    constructor(quoteApi: QuoteApi) {
//      this.quoteApi = new KanyeQuoteApi();
        this.quoteApi = quoteApi;
    }

    sendMail() {
        const quote = this.quoteApi.getQuote();
        mailTo( who: 'leon', text: 'Check deze quote: ' + quote);
    }
}
```

```
interface QuoteApi {
    getQuote(): string;
}
```

```
const api = new KanyeQuoteApi();
const mail = new MailService(api);
mail.sendMail();
```

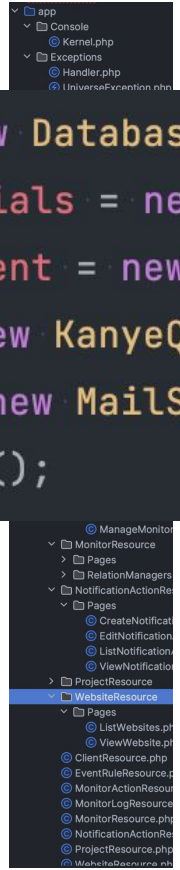
```
const api = new TypischeDeveloperUitsprakenApi();
const mail = new MailService(api);
mail.sendMail();
```

Dependency Injection: Why

```
class MailService {

    private quoteApi: QuoteApi;

    constructor(quoteApi: QuoteApi) {
        this.quoteApi = quoteApi;
    }

    sendMail() {
        const quote = this.quoteApi.getQuote();
        mailTo( who: 'leon', text: 'Check deze quote: ' + quote);
    }
}
```

```
const db = new DatabaseConnection();
const credentials = new CredentialStore(db);
const httpClient = new HttpClient(credentials);
const api = new KanyeQuoteApi(httpClient);
const mail = new MailService(api, httpClient);
mail.sendMail();
```

```
const api = new KanyeQuoteApi();
const mail = new MailService(api);
mail.sendMail();
```



Dependency Injection: Container

```
class MailService {

    private quoteApi: QuoteApi;

    constructor(quoteApi: QuoteApi) {
        this.quoteApi = quoteApi;
    }

    sendMail() {
        const quote = this.quoteApi.getQuote();
        mailTo( who: 'leon', text: 'Check deze quote: ' + quote);
    }
}
```

```
const db = new DatabaseConnection();
const credentials = new CredentialStore(db);
const httpClient = new HttpClient(credentials);
const api = new KanyeQuoteApi(httpClient);
const mail = new MailService(api);
mail.sendMail();
```

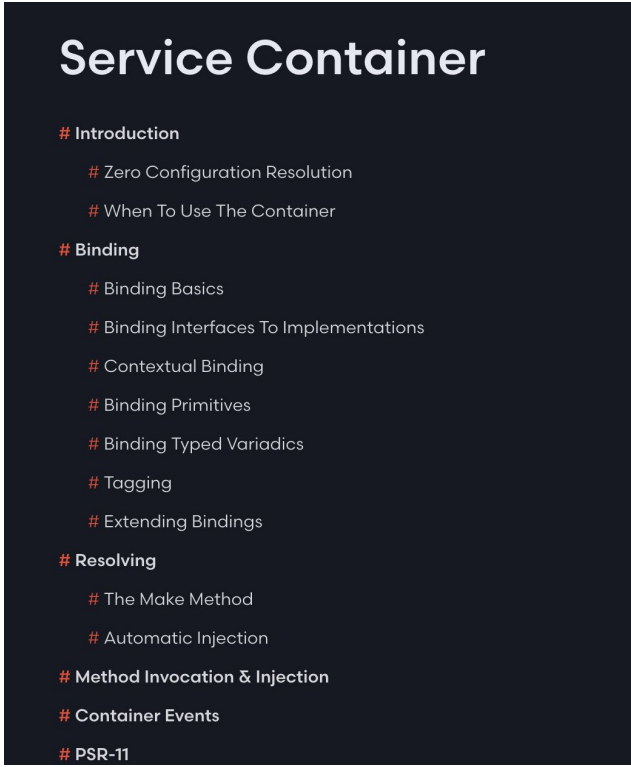
```
class DependencyInjectionContainer {

    static instance = new DependencyInjectionContainer();

    register( what: string, result: any ) {...}

    get( what: string ): MailService {...}

}
```

```
// Bootup
const di = DependencyInjectionContainer.instance;
di.register(DatabaseConnection.name, result: () => new DatabaseConnection( s: 'root', b: 'root' ) )
di.register(CredentialStore.name, result: (db: DatabaseConnection) => new CredentialStore(db));
// Etc etc
```

```
const mail = DependencyInjectionContainer.instance.get(MailService.name);
mail.sendMail();
```

Dependency Injection: Framework

- Niet zelf wiel uitvinden
  - Laravel service container
    https://laravel.com/docs/10.x/container
  - Vue Compisition DI
    https://vuejs.org/api/composition-api-dependency-injection
  - Angular DI
    https://angular.io/guide/dependency-injection
- Automagisch



That’s all folks!

Waarschijnlijk geen tijd meer maar: vragen?

Anders: Come find me @ de borrel

Dependency Injection (context)

```
class MailService {

    private quoteApi: QuoteApi;

    constructor(quoteApi: QuoteApi) {
        this.quoteApi = quoteApi;
    }

    sendMail() {
        const quote = this.quoteApi.getQuote();
        mailTo( who: 'leon', text: 'Check deze quote: ' + quote);
    }
}
```

```
class TestMailService extends Test {

    testMail() {
        const api = new FakeQuoteApi();
        const mail = new MailService(api);
        mail.sendMail();

        assertTrue(isMailSent())
    }

    testFail() {
        const api = new ErrorGooiendeFakeQuoteApi();
        const mail = new MailService(api);
        mail.sendMail();
        //...
    }

}
```

```
interface QuoteApi {
    getQuote(): string;
}
```