

Hogeschool PXL
Departement IT
Academiejaar 2019-2020

Vak	PROEFEXAMEN Web Expert
Resultaat	/20
Periode	Semester 1 Semester 2 Herexamen
Datum	
Klassen	3TIN AON / 3TIW
Lectoren	Dries Swinnen

Studentengegevens	
Achternaam student	
Voornaam student	
Klas	
Lector	
Lokaal	

Samenstelling bundel			
Onderdelen (*)	Deel 1		
Inhoud	Praktijk		
Pagina's			
Puntenverdeling / 100p (100%)		
Digitaal beginbestand	X		
Digitale indiening	X		
Toegelaten hulpmiddelen:			
* rekenmachine			
* laptop	X		
* internet	X		
* cursusmateriaal	X (digitaal)		
Opmerkingen:			

Elke student(e) is verantwoordelijk voor de correcte samenstelling van zijn/haar bundeltje. Eventuele afwijkingen moeten onmiddellijk aan de toezichthouder gesignaleerd worden.

MD5Hash:

[illegible]

Aanvangsuur examen: 08:30

Einde examen: 12:00



**DE HOGESCHOOL
MET HET NETWERK**

Checklist voor het afleggen van examen

- ☐ GSM's/smartphones liggen uitgeschakeld op tafel
- ☐ Studentenkaart ter beschikking houden
- ☐ Jassen en tassen vooraan in het lokaal
- ☐ GEEN GSM's of smartphones in jassen of tassen
- ☐ Bij laptopexamen strikt de opgelegde regels volgen
- ☐ Examenbundels blijven steeds samengeniet
- ☐ Toegestaan: 1 droog koekje en 1 drankje in hersluitbaar flesje
- ☐ Iedereen zwijgt tijdens het examen

-
- ☐ Afgeven mag niet voor 9:00/14:00 uur

- ☐ Niet vergeten op je examenkopij te vermelden:

Naam van de lector

Eigen naam

Klas

MD5Hash



EXAMEN Web Expert

Puntenverdeling:

- Praktijkexamen (open boek met laptop): 100%
- In eerste instantie wordt er gekeken naar functionaliteit, daarna (eventueel) pas naar de code. Het is belangrijker dat je werkende stukken code (use cases, bv. het tonen van items in een lijst, het filteren van de gegevens, ...) hebt. Stukken code die niet werken leveren dus ook geen punten op!
- Code in commentaar of code die niet aangeroepen wordt, wordt ook niet geëvalueerd.
- Code die duidelijk uit voorbeelden gekopieerd wordt en niet werkt, wordt ook niet geëvalueerd.

Bij het verbeteren van de praktijkvraag wordt er rekening gehouden met de onderstaande criteria.

1001 0110	10%
--------------	-----

Niet compileerbare code of niet uitvoerbare code leidt tot een malus van 10% procent op het resultaat van de opgave.

Je haalt 14/20 op een opgave, maar je project compileert niet: 10% van 14/20 is 1,4/20 => Je behaalt 12,6/20.

✓ SPEC	1% max 10%
--------	---------------

Het niet volgen van de specificatie (naamgeving van bestanden, klassen, variabelen, ...) zoals gevraagd in de onderstaande implementatiedetails, leidt tot een malus van 1% per fout met een maximum van 10%.

De percentages worden eerst opgeteld vooraleer de malus berekend wordt.

Je behaalt 14/20, maar het compileert niet (10%) en er zijn 5% specificatiefouten => Dit geeft een malus van 15%.

Elk maluspercentage wordt PER VRAAG berekend.

STAPPEN UIT TE VOEREN ‘VOORALEER’ JE MET HET EXAMEN BEGINT

Clone de repository die voor je is aangemaakt met de examenbestanden. Pas het bestand ./ACHTERNAAM_VOORNAAM aan naar jouw naam. Dit alles zonder spaties en in deze volgorde. Vervolgens voer je in de map ./vraag1 en in de map ./vraag2 het commando `npm install` uit.

STAPPEN UIT TE VOEREN ‘TIJDENS’ JE EXAMEN

Voor elke vraag maak je gebruik van de bestaande projecten en structuur in de map AchternaamVoornaam_webExpert. Je mag niet van deze structuur afwijken. Leg de focus op functionaliteiten die werken.

Elk half uur doe je een commit met daarin alle wijzigingen met als bericht “voortgang xxuxx” en je pusht deze naar github. Indien deze commits ontbreken, kan dit gezien worden als onregelmatigheid. Een onregelmatigheid resulteert in het cijfer 0 op het examen.

Om 9u is dit dus bijvoorbeeld “voortgang 09u00”, om 9u30 is dit “voortgang 9u30” enz.

STAPPEN UIT TE VOEREN WANNEER JE KLAAR BENT MET HET EXAMEN

Verwijder bij elke vraag de folder “node_modules”. Controleer of je de file ACHTERNAAM_VOORNAAM hebt aangepast. Voorzie een commit met als bericht “einde examen” en push de wijzigingen naar github. Controleer op github zelf of de wijzigingen correct gepusht zijn. Geef in de chat op MS Teams aan dat je hebt afgegeven, de lector geeft je groen licht om het kanaal te verlaten.

Veel Succes!

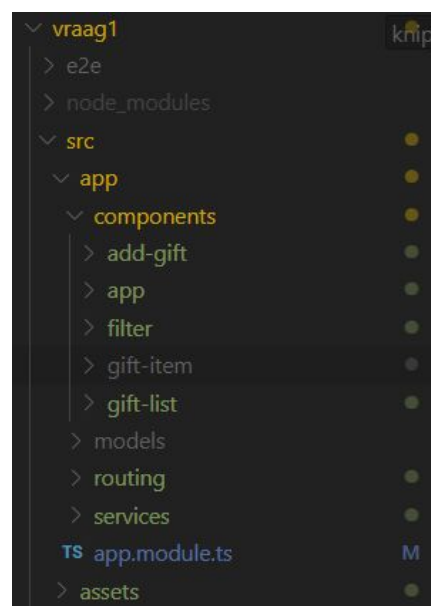
Vraag 1: Angular (70 pt)

1001 0110	10%	✓ SPEC	1% max 10%
--------------	-----	--------	---------------

Voor deze vraag bouw je een Angular applicatie waarbij je de bestanden uit de map “vraag1” gebruikt. De evaluatie zal gebeuren aan de hand van Google Chrome. Je bouwt een applicatie waarin klanten hun getrouwheid bonus kunnen innen in de vorm van cadeaus. Klanten kunnen cadeau’s kopen met punten. De implementatie van het puntensysteem & opbouwen van punten valt buiten de scope van deze opgave.

De applicatie wordt gebouwd in Angular 8. De CSS in dit project is voorzien en de opmaak is niet van belang. Indien je de implementatie details strikt volgt, zou je resultaat er hetzelfde moeten uitzien als de screenshots.

De structuur van het project ligt vast. Er moeten geen extra folders aangemaakt worden. Het is toegestaan om extra bestanden aan te maken indien nodig. De componenten zijn voor je aangemaakt en ook de routing module is aangemaakt. Er is ook een in-memory-web-API voorzien in het mapje “services” en deze is gelinkt aan de applicatie. Deze service is verantwoordelijk voor het mocken van de API. De endpoints van deze API zijn terug te vinden in de in-memory-data.service.ts file.



Bij het opstarten van de applicatie zal de app component geladen worden met daarin de navigatiebalk en een router-outlet. In de navigatiebalk voorzie je 2 linken (<a>) naar de pagina's home en add. Voorzie ook dat de huidige link de CSS klasse 'active' meekrijgt.

Zorg ervoor dat de routes gekoppeld worden in de aparte routing module. Voorzie volgende routes in je applicatie:

- / redirect naar /home
- /home toon de gift-list component
- /add toon de add-gift component
- Alle andere routes worden doorverwezen naar de /home route

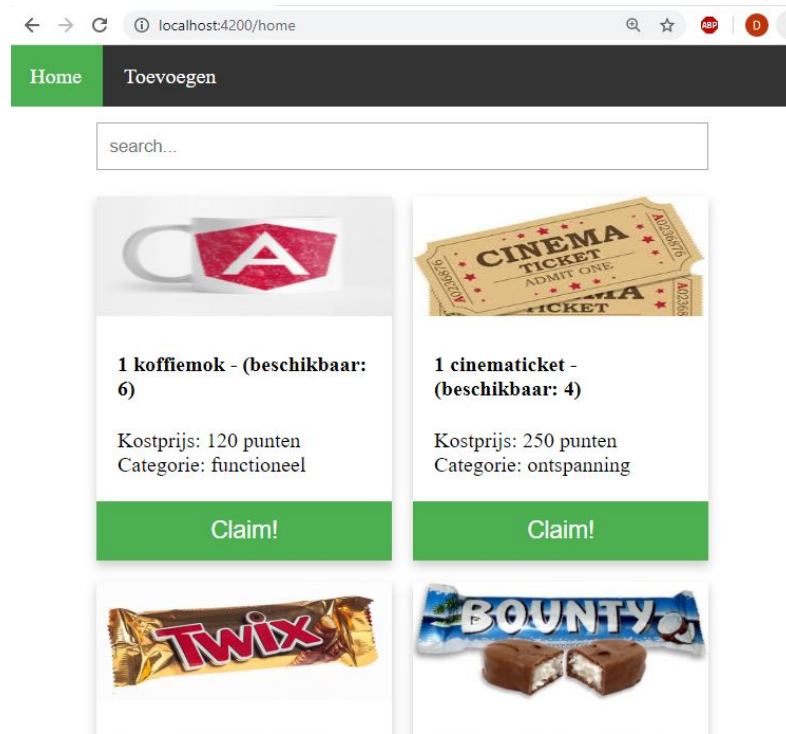
Voor het ophalen/manipuleren van de data maak je zelf een gift.service.ts aan. Deze service zal de reeds bestaande in-memory-data.service aanspreken. Het is niet toegestaan om de in-memory-data.service aan te passen! De volgende endpoints zijn beschikbaar:

```
GET /api/gifts                // alle gift objecten
GET /api/gifts?description=j // gifts waarbij description 'j' bevat
GET /api/gifts/1             // gift met als id=1
POST /api/gifts              // gift object toevoegen
PUT /api/gifts               // gift object updaten
DELETE /api/gifts/1          // delete gift met als id=1
GET /api/categories          // alle category objecten
```

Maak zelf een gift model aan in het mapje "models" en gebruik dit doorheen je applicatie. De structuur van dit model kan je terugvinden in de in-memory-data.service .

Home

Initieel kom je op de ‘home’ URL terecht. Deze pagina toont de gift-list component. Hierin zit zowel de filter-component als ook een lijst van gift-items (één gift = één gift-item component). De data in deze lijst komt uit een zelf geschreven gift-service die de mock API aanspreekt.



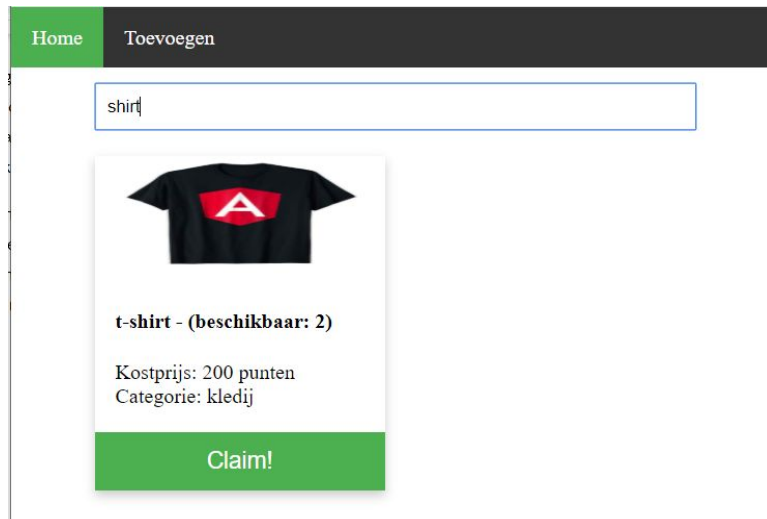
De gift-items bestaan uit een afbeelding van het object, een h4 met daarin de omschrijving en de beschikbaarheid (=stock) van een item. Daaronder staat een paragraaf met de kostprijs en de categorie van deze gifts. Tenslotte staat er nog een knop “claim” waarmee een gift geclaimed kan worden.

Het weergeven van de kostprijs doe je aan de hand van een zelf geschreven pipe die het getal omzet naar het getal + “punten”. De pipe krijgt dus als input bijvoorbeeld “60”. De output van deze pipe is dan “60 punten”.

Bij het klikken op de “claim” knop wordt de stock van dat item verminderd met 1 in de backend. Daarnaast zorg je ervoor dat de lijst geüpdatet wordt. Indien een item een stock van 0 heeft, wordt de claim knop niet getoond en wordt de H4 in het rood en onderlijnd weergegeven.



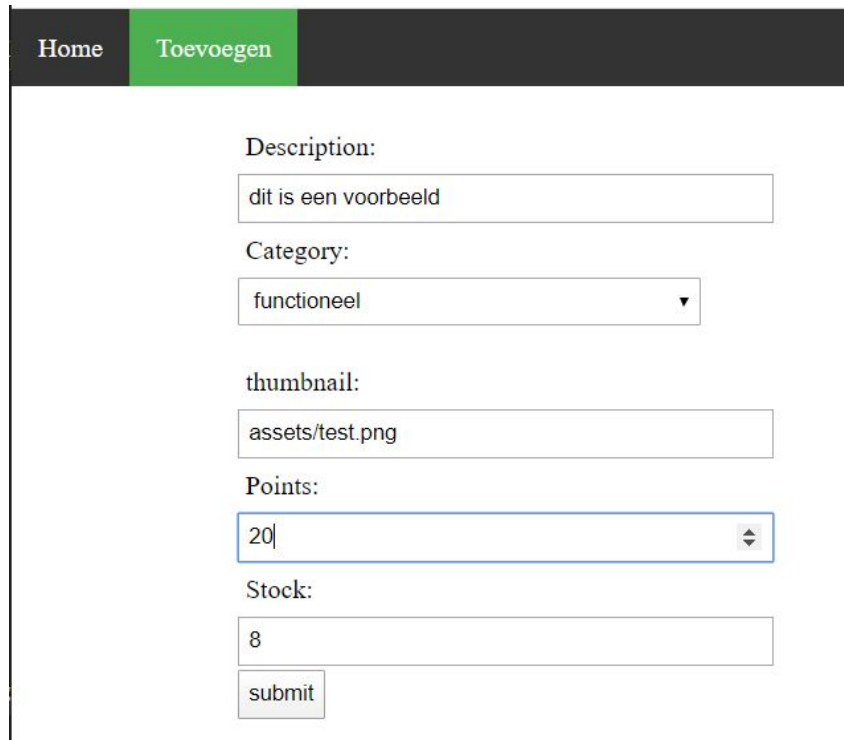
Boven deze lijst is de filter component voorzien. Bij het typen (keyup) in de filtercomponent wordt er in de property “description” gezocht naar de tekst in de filter.



Bij het dubbelklikken (dblclick) op een gift-item zijn afbeelding, wordt deze verwijderd via de API en zal de lijst refreshen.

Add gift

Bij het klikken op de “add” link in de navbar krijg je een model driven (=reactive) formulier waarbij het mogelijk is om een gift item toe te voegen. Het formulier bestaat uit een aantal inputvelden en een knop. De knop is niet aanklikbaar tenzij alle velden ingevuld zijn en alle validatieregels in orde zijn.



Home **Toevoegen**

Description:

Category:

thumbnail:

Points:

Stock:

Het dropdownmenu “Category” wordt opgevuld met data vanuit de backend. Je mag deze dus niet statisch opvullen!

Voorzie in het formulier volgende validatieregels:

- Required: description, category, thumbnail, points, stock
- Minimum 3 karakters: description

Als de validatie van de inputvelden niet in orde is, zorg je ervoor dat de rand van het inputveld rood wordt. Als er bij “description” minder dan 3 karakters ingegeven worden, voorzie je een extra foutmelding zoals te zien in onderstaande screenshot.

Description: minimum 3 karakters!

Category:

thumbnail:

Points:

Points:

Bij het verzenden van het formulier wordt het object aangemaakt en doorgestuurd naar de backend via de service. Na het aanmaken verwijst je de gebruiker door naar de homepage.

Routeguards

1. De add route mag enkel geactiveerd worden na het opgeven van een wachtwoord. Indien de gebruiker hier “admin” ingeeft, wordt de component geladen. Indien er een andere invoer is, verwijst je de gebruiker door naar de home page.

Een prompt gebruiken in javascript gaat via volgende code:

```
let msg = window.prompt("Wachtwoord:");  
console.log(msg)
```

2. Voorzie een manier om te controleren of er in het formulier van de add-gift component ergens data ingevuld is (dirty). Als de gebruiker de pagina probeert te verlaten (via de navigatie) maar wel al data ingevuld heeft, geef je een boodschap (`window.confirm(...)`) of de gebruiker wel zeker is dat hij de pagina wil verlaten.



Bij het klikken op de knop “ok” navigeer je verder. Bij het klikken op de knop “annuleren” blijf je op de huidige pagina. Deze melding krijg je enkel als er al iets in het formulier is ingevuld!

Vraag 2: NodeJS (30 pt)

1001 0110	10%	✓ SPEC	1% max 10%
--------------	-----	--------	---------------

Voor deze vraag bouw je een NodeJS applicatie waarbij de bestanden uit de map “vraag2” gebruikt. Je mag deze applicatie testen met Postman of alternatieve tool. De evaluatie zal gebeuren aan de hand van Postman.

Je hebt voor deze toepassing een mongoDB database in eigen beheer waarin data terecht komt van je persoonlijke Counter Strike: Global Offensive server. Het doel is om een API te voorzien in Express waarmee data uit de databank gehaald kan worden.

Om de data uit de databank te halen maak je gebruik van Mongoose. Voorzie het model “player” in de file `./models/player.js` met volgende properties:

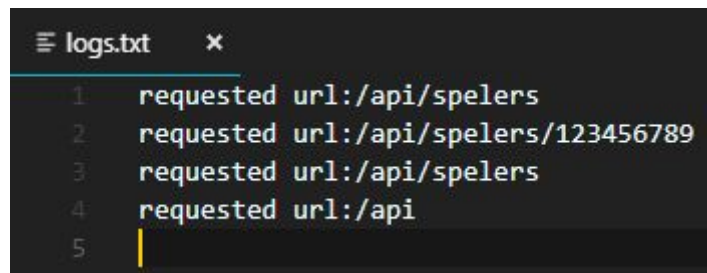
- Name: String, required
- Kills: Number, default: 0
- Deaths: Number, default: 0
- Ranking: String, required

In de file `./db.js` voorzie je functionaliteit waarmee je verbinding maakt met MongoDB. Gebruik hiervoor de database “WebExp”.

Voorzie in `./routes.js` volgende routes aan de hand van een Express router:

- **GET /api/spelers** Deze route geeft een JSON lijst terug van alle spelers in de collection uit mongoDB.
- **GET /api/spelers/<id>** Deze route geeft één speler Object terug als JSON op basis van de ID. Id is een parameter die meegegeven wordt aan de route.
- **POST /api/spelers** Via deze route kan je een nieuwe speler toevoegen aan de collection.
- **PUT /api/spelers** Via deze route kan je, op basis van de meegegeven ID uit het object, een bestaand object updaten.

In de file `./server.js` koppel je bovenstaande allemaal aan elkaar en zorg je voor de opstart van de Express applicatie. Voorzie daarnaast ook een middlewarefunctie die elke binnenkomende request(= `request.url`) wegschrijft in de file `./logs.txt`. Het bestand `logs.txt` kan er dan bijvoorbeeld (na gebruik) als volgt uitzien:



```
logs.txt x
1 requested url:/api/spelers
2 requested url:/api/spelers/123456789
3 requested url:/api/spelers
4 requested url:/api
5
```

Om sequentieel iets naar een bestand weg te schrijven, kan je gebruik maken van onderstaande documentatie van de module `filesystem`.

`fs.appendFile(path, data[, options], callback)`

► History

- `path` `<string>` | `<Buffer>` | `<URL>` | `<number>` filename or file descriptor
- `data` `<string>` | `<Buffer>`
- `options` `<Object>` | `<string>`
 - `encoding` `<string>` | `<null>` Default: `'utf8'`
 - `mode` `<integer>` Default: `0o666`
 - `flag` `<string>` See [support of file system flags](#). Default: `'a'`.
- `callback` `<Function>`
 - `err` `<Error>`

Asynchronously append data to a file, creating the file if it does not yet exist. `data` can be a string or a `Buffer`.

```
fs.appendFile('message.txt', 'data to append', (err) => {
  if (err) throw err;
  console.log('The "data to append" was appended to file!');
});
```

Hou er tenslotte rekening mee dat de applicatie in een later stadium een Angular frontend krijgt. Dit wil zeggen dat data in de body meegegeven zal worden in JSON formaat. Doe hiervoor de nodige aanpassingen a.d.h.v. `BodyParser`.