

Odissee  
DE CO-HOGESCHOOL

# Application Development

Modelbinding



**Sam Van Buggenhout**

Academiejaar 2023-2024

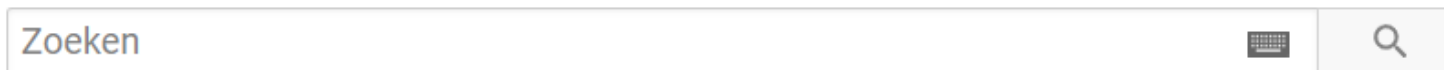
- **Herhaling formulieren**
- **Models**
- **Model Binding**
- **Data Annotations**



# Herhaling formuleren

# Hoe werkt een formulier?

- Formulieren zijn alomtegenwoordig op het web:
  - Registratie voor een website (bv.: nieuw account)
  - Contactformulieren
  - Bestellingen/reservaties
  - Maar ook: login-formulieren, zoekbalken, ...

A horizontal search bar with a light gray border. Inside the bar, the word 'Zoeken' is written in a dark gray font. To the right of the text, there is a small icon of a keyboard and a magnifying glass icon.

→ *Dit is ook een formulier!*

## Hoe werkt een formulier?

- Formulieren worden algemeen gebruikt om **informatie** van de gebruiker (client) naar de website (server) te versturen voor **verwerking**:
  - De gegevens van een bestelling
  - De gegevens (gebruikersnaam, wachtwoord, ...) voor jouw nieuwe account op *Bol.com*
  - Je *r-nummer* en wachtwoord om in te loggen op Toledo
  - Een zoekterm, wanneer je op zoek bent naar een nieuw boek over HTML en CSS

## Hoe werkt een formulier?

- De server die de gegevens ontvangen heeft, stuurt vervolgens een **antwoord** terug:
  - Een bevestigingspagina dat je bestelling voltooid is
  - De homepagina van Toledo, of opnieuw de inlogpagina, om aan te geven dat je wachtwoord niet correct was
  - Het resultaat van je zoekopdracht
  - ...

# Hoe werkt een formulier?

1. De gebruiker vult het formulier in en klikt op *“submit”*



Web Browser

2. De browser “verpakt” de gegevens uit het formulier en stuurt deze naar de server voor verwerking



3. De server ontvangt de gegevens en verwerkt deze via een server-script



Web Server



# Hoe werkt een formulier?

3. De browser ontvangt het resultaat, en geeft het weer op het scherm



Web Browser

2. Het resultaat is een HTML-pagina



Web Server

1. De server stuurt het resultaat van het server-script terug naar de browser

# HTML - Formulier

formulier

```
<form name="nieuwsbrief" action="registreer.php" method="POST">
  <p>Schrijf je nu in op onze nieuwsbrief!</p>
  <p>E-mail: <input type="text" name="email"></p>
  <p><input type="submit" name="registreer" value="Registreer nu!"></p>
</form>
```

Schrijf je nu in op onze nieuwsbrief!

E-mail:

Registreer nu!

# HTML - Formulier

```
<form name="nieuwsbrief" action="registreer.php" method="POST">
```

De start-tag van een **<form>**-element kan de volgende attributen bevatten:

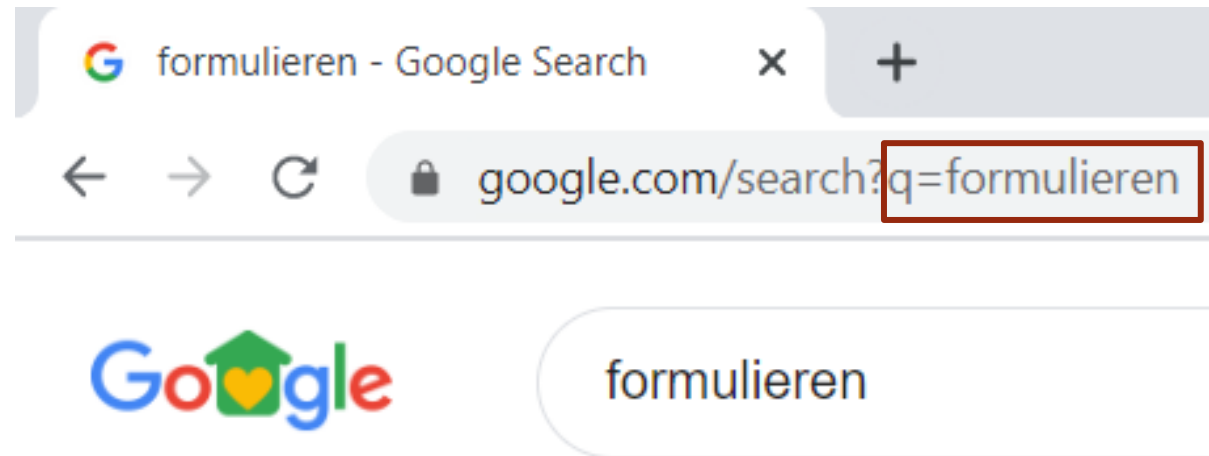
Attribuut	Betekenis
<b>name</b>	Naam voor een formulier, zodat de server-code weet welk formulier verstuurd werd
<b>action</b>	URL van het server-script waar de gegevens naartoe gestuurd moeten worden
<b>method</b>	HTTP-method die aangeeft op welke <b>manier</b> de gegeven naar de server gestuurd worden ("GET" of "POST")

## HTML – Formulier (GET vs POST)

- Via het **method-attribuut** van het <form>-element, kunnen we aangeven op welke **manier** de gegevens van het formulier naar de server moeten gestuurd worden (*dit komt overeen met de HTTP-method die hiervoor gebruikt wordt*)
- Er zijn twee veelgebruikte waarden voor het method-attribuut:
  - GET
  - POST

## HTML – Formulier (GET vs POST)

Bij GET worden de gegevens in het formulier aan de URL van de pagina toegevoegd (dit noemt men de query-string):



## HTML – Formulier (GET vs POST)

- Een gevolg hiervan is dat je “GET” **NOOIT** mag gebruiken om gevoelige data naar een server te sturen!  
(bv.: wachtwoorden, betalingsgegevens, ...)



<http://www.mywebpage.be/login.html?username=admin&password=admin123>

## HTML – Formulier (GET vs POST)

- Je mag **GET** enkel gebruiken voor operaties die **idempotent** zijn
  - Een operatie is **idempotent** als meerdere identieke aanvragen hetzelfde beoogde effect op de server hebben als één zo'n request
- Bijvoorbeeld: als je 10x een zoekopdracht uitvoert met een zoekterm “puppies”, heeft dit voor de server hetzelfde effect als je deze opdracht één keer zou uitvoeren → **idempotent**
- Maar: als je 10x eenzelfde bestelformulier doorstuurt, heeft dit wél een ander effect op de server dan wanneer je dit één keer zou doorsturen (10 bestellingen in plaats van één) → **NIET idempotent**

→ Kort samengevat: gebruik **GET** enkel voor “**lees-operaties**”, niet voor “**schrijf-operaties**”

## HTML – Formulier (GET vs POST)

- Nog een aantal eigenschappen van **GET**-requests:
  - kunnen gecached worden
  - komen in browsergeschiedenis
  - kunnen gebookmarked worden
  - zijn beperkt in lengte



## HTML – Formulier (GET vs POST)

- Eigenschappen van POST-method:
  - gegevens komen niet in de URL van de request terecht (is dus wél geschikt voor gevoelige data)
  - POST-operaties zijn niet idempotent
  - requests worden niet gecached
  - requests komen niet in browsergeschiedenis terecht
  - requests kunnen niet gebookmarked worden

# HTML – Formulier (GET vs POST)

## GET of POST?

Use-case	HTTP-method (GET of POST)
Een login-formulier	
De detailpagina bekijken van het product met product-id 5478895	
Een registratieformulier voor een nieuwsbrief	
De productbeheerder die wijzigingen doorgevoerd heeft aan de beschrijving van een product en zijn/haar wijzigingen wilt bewaren	
Een reactie plaatsen op Facebook	
Een bericht 'liken' op Facebook	
Een zoekopdracht naar producten tussen €25 en €50	



# Models

De 'M' in MVC

## Models: inleiding

- Bij routing: **eenvoudige parameter(s)** doorgeven via **URL**
  - Voorbeeld: ID van product, student, valuta, ...
  - Voorbeeld: *https://localhost:44374/Product/Details/5*
- Mogelijk om **beperkt aantal parameters** naar Action-method te sturen, maar wat bij **groter aantal** parameters?
  - Voorbeeld: gegevens in zoekformulier, login-gegevens, inhoud van registratieformulier, ...
- **Oplossing:** parameters verzamelen in **binding model**

## Models: inleiding

- Binding model: waarden van formulier/URL worden “gebonden” aan C#-object
- Dit object wordt doorgegeven als parameter aan Action-method in controller
- Action-method kan property's in object gebruiken om deze te verwerken (bv.: parameters van zoekopdracht, registratiegegevens van een klant, ...)

# Soorten Models

- Binding model:
  - Data opgegeven door gebruiker
  - Route-parameters, query-string, inhoud van formulier, ...
  - Eén of meerdere C#-objecten
  - Wordt als parameter doorgegeven aan Action-methode van controller
- Application model:
  - Services en classes die business logica van de applicatie implementeren
  - Domain-models, Database-models, ...

# Soorten Models

- View model:
  - C#-class die alle gegevens bevat om response te genereren
  - “Encapsuleert” alle zichtbare gegevens in één object
  - Vaak ook bijkomende property's (bv.: totaal aantal zoekresultaten, ...)
- API model:
  - Variatie op View model
  - In plaats van gegevens voor view: bevat gegevens die door API als resultaat (in XML/JSON-formaat) teruggestuurd worden

## Voorbeeld

- Voorbeeld TODO-app:

username en category  
via route-values

Example TODO Home Privacy

### TODO Items

- Auto Wassen
- Boodschappen doen

Username: Sam  
Category: open

Gefilterd op basis van  
username en category

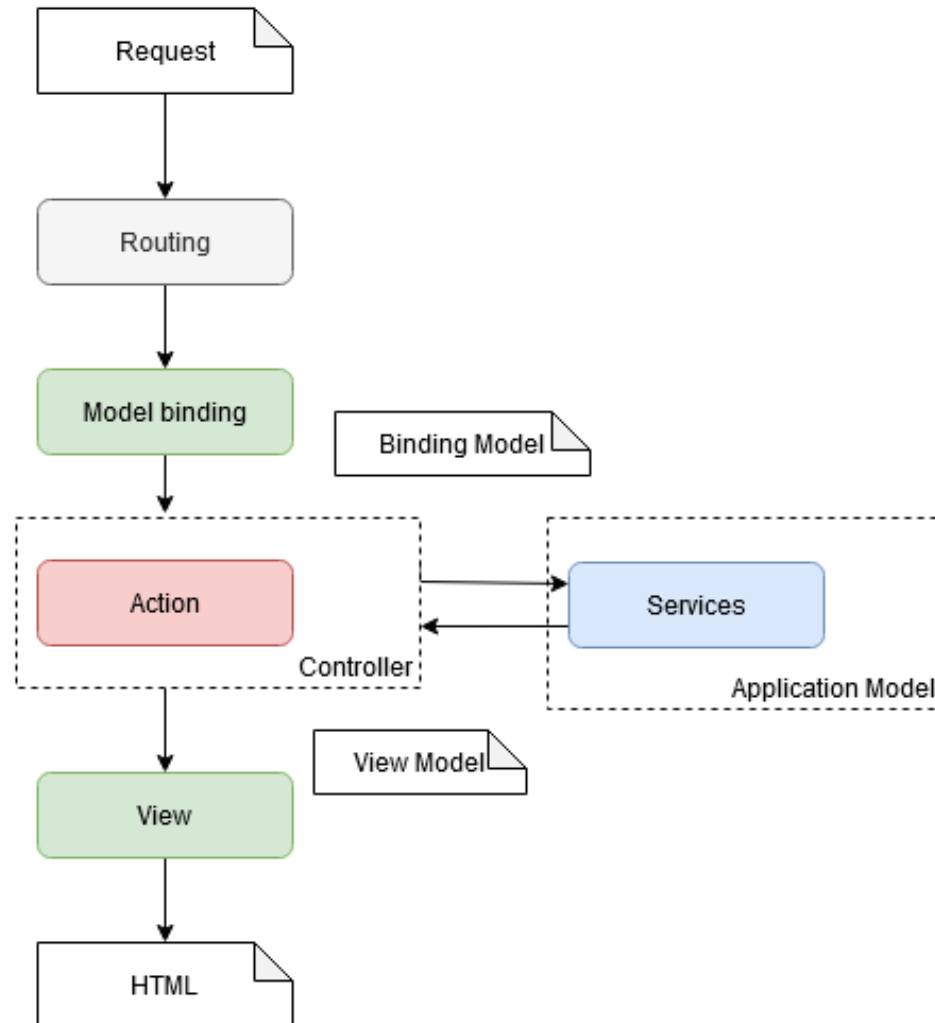
Ook weergegeven  
in View model



## Voorbeeld

- **Binding model:** username en category (via route-values)
- **Application model:** bevat todo-items, classes en services die TODO-items kunnen ophalen (bv. Repository)
- **View model:** bevat alle gegevens die weergegeven worden in de HTML-reponse (TODO-items, username, category)

# Models: overzicht





# Model binding

## Wat is Model binding?

- Bij HTTP-request wordt vaak data meegestuurd naar de server
  - Bv.: login-gegevens, gegevens van een bestelling, parameters van een zoekopdracht, ...
- Kan op verschillende manieren:
  - **query-string:** bv.: `https://localhost:1234/Products/Search?q=smartphone&os=android`
  - **route-value:** bv.: `https://localhost:1234/Student/Edit/1`
  - **form-value:** *inhoud velden formulier bij HTTP POST*
- Deze waarden worden doorgegeven als parameters aan Action-method in Controller

# Wat is Model binding?

**MAAR:** `public IActionResult CreateProduct(int id, string name, string description, double price)`

- Veel parameters!
  - Moeilijk onderhoudbaar!
  - Manuele mapping naar C#-object nodig
- Oplossing: **model binding!**

## Wat is Model binding?

**Model binding:** haalt waarden uit request en gebruikt deze waarden om C#-object te maken en dit door te geven als parameter aan de Action-method

```
public class Product {  
    public int ID { get; set; }  
    public string Name { get; set; }  
    public string Description { get; set; }  
    public double Price { get; set; }  
}
```

```
public IActionResult CreateProduct(Product product) {  
    //logica om product op te slaan  
}
```

## Model binder

- Taak van model binder: waarden uit request aan juiste property's toekennen
- Naam wordt gematcht met naam van property
- Gebeurt in volgorde:
  1. Form-values
  2. Route-values
  3. Query string-values

# Model binding: voorbeeld

## View (formulier)

### Create Product

#### Product

ID

Name

Description

Price

## Body POST-request

Id=1  
Name=Nespresso  
Description=Koffiemachine op basis  
van capsules  
Price=125.95

## C#-object (model)

```
public class Product {  
    public int ID { get; set; }  
    public string Name { get; set; }  
    public string Description { get; set; }  
    public double Price { get; set; }  
}
```

*Opmerking:* naam van waarde in POST-request wordt bepaald door name-attribuut van <input>-element

```
<input id="Price" name="Price" ...>
```



# Modelbinding: combinatie

- Sources kunnen ook gecombineerd worden
  - Vaak bij edit: id van te editeren object via route-value, gewijzigde informatie via POST
  - **OPGELET:** volgorde is van belang!

The screenshot shows a web browser window with the address `https://localhost:44304/Products/EditProduct/7`. The page title is "Edit Product". Below the title, there is a form for editing a product. The form has three input fields: "Name" (containing "HP Elitebook"), "Description" (containing "Laptop voor high-end gebruik"), and "Price" (containing "1499.95"). Below the form is a blue "Edit" button.

To the right of the form is a table with two columns: "Name" and "Value". The table contains the following data:

Name	Value
product	{Example_Student_modelbinding_Models.Product}
Description	"Laptop voor high-end gebruik"
ID	7
Name	"HP Elitebook"
Price	1499.95

Arrows indicate the mapping between the form fields and the table data:

- A green arrow points from the "ID" field in the table to the route value "7" in the browser address bar.
- A red arrow points from the "Name" input field to the "Name" field in the table.
- A blue arrow points from the "Description" input field to the "Description" field in the table.
- A yellow arrow points from the "Price" input field to the "Price" field in the table.

## Model binding: primitieve types

- **Primitieve types** worden gebonden door model binder
- Eén of **meerdere** parameters mogelijk
- Indien **geen waarde** voor parameter: **default-waarde** van type
  - Gebruik *nullable-types* indien je null-waarde wenst voor ontbrekende waarden (bv.: `int? id`)

### Routing:

{controller}/{action}/{from}/{to}

### URL:

Currency/Convert/USD/EUR



### Action-method (in controller):

```
public IActionResult Convert(string from, string to) {  
    //...  
}
```

- from = USD
- to = EUR

## Model binding: complexe types

- Indien veel parameters: beter klasse maken om parameters te groeperen
  - Binding: naam parameter = naam property
  - Vereisten: default constructor + public setters

```
public IActionResult Register(string firstName, string lastName, string phoneNumber, string email)
```

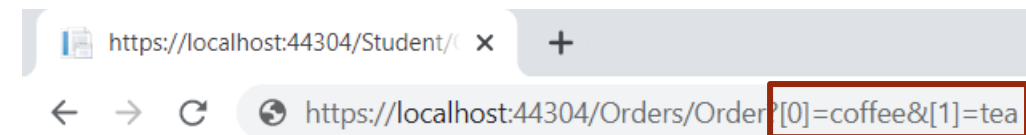
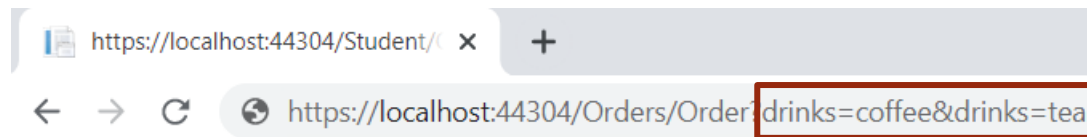
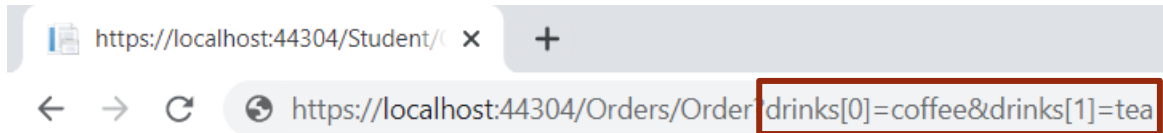


```
public class UserBindingModel {  
    public string FirstName { get; set; }  
    public string LastName { get; set; }  
    public string PhoneNumber { get; set; }  
    public string Email { get; set; }  
}
```

```
public IActionResult Register(UserBindingModel userModel)
```

## Model binding: collecties

- Meerdere waarden doorgeven in vorm van collectie
- Kan op drie manieren:
  - naam[index]=waarde (voorbeeld: drinks[0]=coffee)
  - naam=waarde (voorbeeld: drinks=coffee&drinks=tea)
  - [index]=waarde (**enkel indien één lijst!** Voorbeeld: [0]=coffee)



```
public IActionResult Order(List<string> drinks) {  
    //...  
}
```

# Model binding: bestanden uploaden

- Gebruik type **IFormFile** om bestanden te uploaden
  - Voorbeeld: profielfoto, PDF-bestand, ...

```
public IActionResult SavePicture(IFormFile file) {  
    if (file.Length > 0) {  
        var filePath = Path.GetTempFileName();  
  
        using (var stream = System.IO.File.Create(filePath)) {  
            file.CopyTo(stream);  
        }  
    }  
  
    return View();  
}
```

## Files uploaden: waarschuwingen



- Gebruik IFormFile enkel voor kleine bestanden
- Opgelet voor grote bestanden!
- Let op voor uitvoerbare bestanden
- Genereer altijd nieuwe filenaam voor bestanden die geüpload worden door gebruiker
- Waar data bewaren? Database? File-systeem?



# Data Annotations

## Wat zijn Data Annotations?

- **Data Annotations** zijn **attributen** die aan property's van model kunnen toegevoegd worden
- Voegen **meta-data** toe aan deze property's
  - Bv.: verplicht veld, maximale lengte, type (email, password, ...)
- Doel:
  - Bij formulier geschikte componenten genereren (*zie later*)
  - Validatie en foutboodschappen



# Voorbeeld Data Annotations

```
public class PersonRegistrationModel {  
    [Required]  
    [MaxLength(100)]  
    [Display(Name = "Voornaam")]  
    public string FirstName { get; set; }  
  
    [Required]  
    [MaxLength(100)]  
    [Display(Name = "Familiennaam")]  
    public string LastName { get; set; }  
  
    [Required]  
    [EmailAddress]  
    public string Email { get; set; }  
  
    [Required]  
    [Compare("Email")]  
    [Display(Name = "Bevestig email")]  
    public string ConfirmEmail { get; set; }  
}
```

- **[Required]**: geeft aan dat velden verplicht moeten meegegeven worden bij request
- **[MaxLength]**: geeft de maximale lengte van de string aan
- **[Display]**: gebruiksvriendelijke naam voor het veld (wordt gebruikt bij genereren van formulier-labels)
- **[EmailAddress]**: geeft aan dat string moet voldoen aan formaat van email-adres. Wordt tevens gebruikt bij genereren van formulier-element
- **[Compare]**: gaat na dat de waarde van deze property overeenkomt met de waarde van een andere property

## Veelgebruikte Data Annotations

Data Annotation	Doel
[EmailAddress]	Property moet geldig email-adres zijn
[StringLength(max)]	Maximale lengte voor string-property
[MinLength(min)]	Bepaalt minimum aantal elementen in collectie
[Range(min,max)]	Gaat na of getal tussen min- en max-waarde ligt
[RegularExpression(regex)]	Valideert of string matcht met reguliere expressie
[Url]	Gaat na of string een geldige URL is
[Required]	Geeft aan dat property verplicht een waarde moet krijgen
[Compare(name)]	Controleert of waarde van property overeenkomt met waarde van andere property
[DataType(DataType.<type>)]	Geeft aan dat property specifiek data-type bevat (bv.: <i>DataType.Password</i> )

# Data Annotations en foutboodschappen

- Data Annotations worden gebruikt voor validatie van gegevens (bv.: bij invullen van formulier)
- Bij validatie-fout: standaard foutmelding getoond (afhankelijk van type Data Annotation)
- Standaard foutmelding kan overschreven worden via ErrorMessage-property in attribuut (wordt ondersteund door alle Data Annotations)

```
public class PersonRegistrationModel {  
    [Required(ErrorMessage = "Dit is een verplicht veld!")]  
    [EmailAddress(ErrorMessage = "Dit is geen geldig email-adres!")]  
    public string Email { get; set; }  
  
    //...  
}
```