Klik hier voor pdf...

FILTEREN, SORTEREN, PAGINEREN, 1-OP-VEEL SCAFFOLDEN

Les 10 Webdevelopment

WAAR STAAN WE?

1. Frontend: HTML, CSS, Javascript
2. Frontend: Bootstrap
3. Backend: C#
4. MVC
5. Backend: C# LINQ
6. ORM
7. Testen
8. Layout
9. Security
10. **Zoeken/filteren, Sorteren, Pagineren**
11. Web API, JSON, Ajax, Azure
12. Architectuur

VANDAAG OP HET PROGRAMMA

- Inleiding
- LINQ: IEnumerable vs IQueryable
- Sorteren
- Filter
- Pagineren
- Scaffolden van relaties
- Toets bespreken

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

INLEIDING

Tijdens college bouwen we verder aan bestaande Index

We voegen toe:

- Én sortering
- Én filtering
- Én paginering

En dat allemaal in één methode?

- Hou de bestaande action Index zo eenvoudig mogelijk
- Bouw Index op uit aanroepen van methodes
- En maak voor sorteren, filteren en pagineren aparte methodes.
- Code smell: long method
- Code smell: duplicate code, bij andere Index actions.

Aan het eind van het college komt de code op Blackboard.

In de opdracht breiden we de code uit.

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LINQ: IENUMERABLE VS IQUERYABLE

- Tot nu toe: LINQ 'achter elkaar' geplakt. Nu: ook tijdelijk in variabelen opslaan.
- Beide werken met deferred execution
- IQueryable implementeert IEnumerable
- Voorbeeld:

```
IEnumerable<Student> first = db.Students;
var second = first.Where(s => s.Naam == "Bob");
var third = second.Where(s => s.Id > 100);
var result = third.Where(s => s.Id < 200);
Console.Write(result.First().Naam);
```

  Er wordt `SELECT * FROM Students` uitgevoerd. Verander IEnumerable in IQueryable en er wordt `SELECT TOP 1 * FROM Students WHERE Naam = "Bob" AND Id > 100 AND AND Id < 200` uitgevoerd.
- Kijk naar de verschillende parametertypen
- Niet alles mag in een 'expressie'
- ⚠ Dit is niet OO! ⚠

LEES DE SQL QUERY UIT DE IQUERYABLE

Bron (voor EF Core 5.0 is dit niet nodig)

```
public static string ToSql<TEntity>(this IQueryable<TEntity> query) where TEntity : class
{
    var enumerator = query.Provider.Execute<IEnumerable<TEntity>>(query.Expression).GetEnumerator();
    var relationalCommandCache = enumerator.Private("_relationalCommandCache");
    var selectExpression = relationalCommandCache.Private<SelectExpression>("_selectExpression");
    var factory = relationalCommandCache.Private<IQuerySqlGeneratorFactory>("_querySqlGeneratorFactory");
    return factory.Create().GetCommand(selectExpression).CommandText;
}

private static object Private(this object obj, string privateField) =>
    obj?.GetType().GetField(privateField, BindingFlags.Instance | BindingFlags.NonPublic)?.GetValue(obj);
private static T Private<T>(this object obj, string privateField) =>
    (T)obj?.GetType().GetField(privateField, BindingFlags.Instance | BindingFlags.NonPublic)?.GetValue(obj);
```

Alternatief: log alle SQL queries

SORTEREN

HET SORTEREN VAN EEN LIJST

Hoe wordt een lijst gesorteerd?

```
return View(await _context.Student.OrderBy(s => s.Naam).ToListAsync());
```

Sorteren kan op meerdere manieren:

- verschillende properties van een `Student` (tegelijkertijd?)
- sorteren kan alfabetisch of in omgekeerde volgorde (`OrderByDescending`)

In deze demo: alleen sorteren op Naam.

HET SORTEREN VAN EEN LIJST

Hoe wordt een lijst gesorteerd?

```
return View(await _context.Student.OrderBy(s => s.Naam).ToListAsync());
```

Sorteren kan op meerdere manieren:

- verschillende properties van een `Student` (tegelijkertijd?)
- sorteren kan alfabetisch of in omgekeerde volgorde (`OrderByDescending`)

In deze demo: alleen sorteren op Naam.

HET SORTEREN VAN EEN LIJST

Hoe wordt een lijst gesorteerd?

```
return View(await _context.Student.OrderBy(s => s.Naam).ToListAsync());
```

Sorteren kan op meerdere manieren:

- verschillende properties van een `Student` (tegelijkertijd?)
- sorteren kan alfabetisch of in omgekeerde volgorde (`OrderByDescending`)

In deze demo: alleen sorteren op Naam.

HET SORTEREN VAN EEN LIJST

Hoe wordt een lijst gesorteerd?

```
return View(await _context.Student.OrderBy(s => s.Naam).ToListAsync());
```

Sorteren kan op meerdere manieren:

- verschillende properties van een `Student` (tegelijkertijd?)
- sorteren kan alfabetisch of in omgekeerde volgorde (`OrderByDescending`)

In deze demo: alleen sorteren op Naam.

HET SORTEREN VAN EEN LIJST

Hoe wordt een lijst gesorteerd?

```
return View(await _context.Student.OrderBy(s => s.Naam).ToListAsync());
```

Sorteren kan op meerdere manieren:

- verschillende properties van een `Student` (tegelijkertijd?)
- sorteren kan alfabetisch of in omgekeerde volgorde (`OrderByDescending`)

In deze demo: alleen sorteren op Naam.

HET SORTEREN VAN EEN LIJST

Hoe wordt een lijst gesorteerd?

```
return View(await _context.Student.OrderBy(s => s.Naam).ToListAsync());
```

Sorteren kan op meerdere manieren:

- verschillende properties van een `Student` (tegelijkertijd?)
- sorteren kan alfabetisch of in omgekeerde volgorde (`OrderByDescending`)

In deze demo: alleen sorteren op Naam.

SORTEER-OPTIES

Er moeten sorteeropties van de client naar de server.

- In de argumenten van de action in de Controller (via query parameters (href) of post data (form))
- In de View (argument had ook een boolean kunnen zijn, maar misschien in de toekomst wordt dit naam_oplopend):

```
<a href="/Students?sorteerVolgorde=oplopend">↑</a>
<a href="/Students?sorteerVolgorde=aflopend">↓</a>
```

- Iets beter (Anchor Tag Helpers):

```
<a asp-action="Index" asp-route-sorteerVolgorde="oplopend">
```

- In de Controller:

```
public async Task<IActionResult> Index(string sorteerVolgorde)
```

SORTEER-OPTIES

Er moeten sorteeropties van de client naar de server.

- In de argumenten van de action in de Controller (via query parameters (href) of post data (form))
- In de View (argument had ook een boolean kunnen zijn, maar misschien in de toekomst wordt dit naam_oplopend):

```
<a href="/Students?sorteerVolgorde=oplopend">↑</a>
<a href="/Students?sorteerVolgorde=aflopend">↓</a>
```

- Iets beter (Anchor Tag Helpers):

```
<a asp-action="Index" asp-route-sorteerVolgorde="oplopend">
```

- In de Controller:

```
public async Task<IActionResult> Index(string sorteerVolgorde)
```

SORTEER-OPTIES

Er moeten sorteeropties van de client naar de server.

- In de argumenten van de action in de Controller (via query parameters (href) of post data (form))
- In de View (argument had ook een boolean kunnen zijn, maar misschien in de toekomst wordt dit naam_oplopend):

```
<a href="/Students?sorteerVolgorde=oplopend">↑</a>
<a href="/Students?sorteerVolgorde=aflopend">↓</a>
```

- Iets beter (Anchor Tag Helpers):

```
<a asp-action="Index" asp-route-sorteerVolgorde="oplopend">
```

- In de Controller:

```
public async Task<IActionResult> Index(string sorteerVolgorde)
```

SORTEER-OPTIES

Er moeten sorteeropties van de client naar de server.

- In de argumenten van de action in de Controller (via query parameters (href) of post data (form))
- In de View (argument had ook een boolean kunnen zijn, maar misschien in de toekomst wordt dit naam_oplopend):

```
<a href="/Students?sorteerVolgorde=oplopend">↑</a>
<a href="/Students?sorteerVolgorde=aflopend">↓</a>
```

- Iets beter (Anchor Tag Helpers):

```
<a asp-action="Index" asp-route-sorteerVolgorde="oplopend">
```

- In de Controller:

```
public async Task<IActionResult> Index(string sorteerVolgorde)
```

SORTEER-OPTIES

Er moeten sorteeropties van de client naar de server.

- In de argumenten van de action in de Controller (via query parameters (href) of post data (form))
- In de View (argument had ook een boolean kunnen zijn, maar misschien in de toekomst wordt dit naam_oplopend):

```
<a href="/Students?sorteerVolgorde=oplopend">↑</a>
<a href="/Students?sorteerVolgorde=aflopend">↓</a>
```

- Iets beter (Anchor Tag Helpers):

```
<a asp-action="Index" asp-route-sorteerVolgorde="oplopend">
```

- In de Controller:

```
public async Task<IActionResult> Index(string sorteerVolgorde)
```

DE ACTION

```
IQueryable<Student> lijst = _context.Student;
switch (sorteerVolgorde)
{
    case "aflopend": lijst = lijst.OrderByDescending(s => s.Naam); break;
    default: lijst = lijst.OrderBy(s => s.Naam); break;
}
return View(await lijst.ToListAsync());
```

DE SORT METHODE

Maak er een aparte methode van.

```
private IQueryable<Student> Sort(string sorteerVolgorde)
{
    IQueryable<Student> lijst = _context.Student;
    switch (sorteerVolgorde)
    {
        case "aflopend": lijst = lijst.OrderByDescending(s => s.Naam); break;
        default: lijst = lijst.OrderBy(s => s.Naam); break;
    }
    return lijst;
}
public async Task<IActionResult> Index(string sorteerVolgorde)
{
    return View(await Sort(sorteerVolgorde).ToListAsync());
}
```

De methode Sort hoeft niet async te zijn!

ONTHOUDEN SORTEEROPTIES CLIENT-SIDE

- Er moeten sorteeropties van de Controller naar de View.
- In ViewData (of ViewBag) of in een ViewModel.
-
```
ViewData["Sorteer"] = sorteerVolgorde ?? "oplopend";
```
- Pas op: `ViewData["..."] == "....."` checkt alleen op reference, doe een cast. s

```
@if ((string)ViewData["Sorteer"] == "aflopend")
{ <a asp-action="Index" asp-route-sorteerVolgorde="oplopend">↑</a> }
else
{ <a asp-action="Index" asp-route-sorteerVolgorde="aflopend">↓</a> }
```

ONTHOUDEN SORTEEROPTIES CLIENT-SIDE

- Er moeten sorteeropties van de Controller naar de View.
- In ViewData (of ViewBag) of in een ViewModel.
- ```
  ViewData["Sorteer"] = sorteerVolgorde ?? "oplopend";
  ```
- Pas op: `ViewData["..."] == "....."` checkt alleen op reference, doe een cast. s

```
@if ((string)ViewData["Sorteer"] == "aflopend")
{ <a asp-action="Index" asp-route-sorteerVolgorde="oplopend">↑</a> }
else
{ <a asp-action="Index" asp-route-sorteerVolgorde="aflopend">↓</a> }
```
```

ONTHOUDEN SORTEEROPTIES CLIENT-SIDE

- Er moeten sorteeropties van de Controller naar de View.
- In ViewData (of ViewBag) of in een ViewModel.
-
```
ViewData["Sorteer"] = sorteerVolgorde ?? "oplopend";
```
- Pas op: `ViewData["..."] == "....."` checkt alleen op reference, doe een cast. s

```
@if ((string)ViewData["Sorteer"] == "aflopend")
{ <a asp-action="Index" asp-route-sorteerVolgorde="oplopend">↑</a> }
else
{ <a asp-action="Index" asp-route-sorteerVolgorde="aflopend">↓</a> }
```

ONTHOUDEN SORTEEROPTIES CLIENT-SIDE

- Er moeten sorteeropties van de Controller naar de View.
- In ViewData (of ViewBag) of in een ViewModel.
- ```
  ViewData["Sorteer"] = sorteerVolgorde ?? "oplopend";
  ```
- Pas op: `ViewData["..."] == "....."` checkt alleen op reference, doe een cast. s

```
@if ((string)ViewData["Sorteer"] == "aflopend")
{ <a asp-action="Index" asp-route-sorteerVolgorde="oplopend">↑</a> }
else
{ <a asp-action="Index" asp-route-sorteerVolgorde="aflopend">↓</a> }
```
```

FILTER

DE ACTION ONTVANGT DE ZOEKOPDRACHT

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter)
{
    IQueryable<Student> lijst = Sort(sorteerVolgorde);
    lijst = Filter(lijst, filter);
    return return View(await lijst.ToListAsync());
}
```

- Je ziet: we houden de Action nog steeds zo eenvoudig mogelijk!
- Het sorteren en zoeken hebben we uitbesteed aan twee aparte methodes.
- Hou de namen van de methodes zo algemeen mogelijk: misschien later veralgemeniseren voor andere properties?
- De methodes Sort en Filter maken gebruik van deferred execution en hoeven dus niet asynchroon uitgevoerd te worden

DE ACTION ONTVANGT DE ZOEKOPDRACHT

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter)
{
    IQueryable<Student> lijst = Sort(sorteerVolgorde);
    lijst = Filter(lijst, filter);
    return return View(await lijst.ToListAsync());
}
```

- Je ziet: we houden de Action nog steeds zo eenvoudig mogelijk!
- Het sorteren en zoeken hebben we uitbesteed aan twee aparte methodes.
- Hou de namen van de methodes zo algemeen mogelijk: misschien later veralgemeniseren voor andere properties?
- De methodes Sort en Filter maken gebruik van deferred execution en hoeven dus niet asynchroon uitgevoerd te worden

DE ACTION ONTVANGT DE ZOEKOPDRACHT

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter)
{
    IQueryable<Student> lijst = Sort(sorteerVolgorde);
    lijst = Filter(lijst, filter);
    return return View(await lijst.ToListAsync());
}
```

- Je ziet: we houden de Action nog steeds zo eenvoudig mogelijk!
- Het sorteren en zoeken hebben we uitbesteed aan twee aparte methodes.
- Hou de namen van de methodes zo algemeen mogelijk: misschien later veralgemeniseren voor andere properties?
- De methodes Sort en Filter maken gebruik van deferred execution en hoeven dus niet asynchroon uitgevoerd te worden

DE ACTION ONTVANGT DE ZOEKOPDRACHT

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter)
{
    IQueryable<Student> lijst = Sort(sorteerVolgorde);
    lijst = Filter(lijst, filter);
    return return View(await lijst.ToListAsync());
}
```

- Je ziet: we houden de Action nog steeds zo eenvoudig mogelijk!
- Het sorteren en zoeken hebben we uitbesteed aan twee aparte methodes.
- Hou de namen van de methodes zo algemeen mogelijk: misschien later veralgemeniseren voor andere properties?
- De methodes Sort en Filter maken gebruik van deferred execution en hoeven dus niet asynchroon uitgevoerd te worden

DE ACTION ONTVANGT DE ZOEKOPDRACHT

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter)
{
    IQueryable<Student> lijst = Sort(sorteerVolgorde);
    lijst = Filter(lijst, filter);
    return return View(await lijst.ToListAsync());
}
```

- Je ziet: we houden de Action nog steeds zo eenvoudig mogelijk!
- Het sorteren en zoeken hebben we uitbesteed aan twee aparte methodes.
- Hou de namen van de methodes zo algemeen mogelijk: misschien later veralgemeniseren voor andere properties?
- De methodes Sort en Filter maken gebruik van deferred execution en hoeven dus niet asynchroon uitgevoerd te worden

FILTEREN – HET FILTEREN ZELF

```
private IQueryable<Student> Filter(IQueryable<Student> lijst, string filter)
{
    if (!String.IsNullOrEmpty(filter))
        lijst = lijst.Where(s => s.Naam.Contains(filter));
    return lijst;
}
```

- Dit is heel eenvoudig en kan ingewikkelder
  - Hoofdlettergevoeligheid: `.ToLower()`
  - Wat als een woord meerdere keren voorkomt?
  - Wat als een woord verkeerd wordt gespeld?
  - Wat als de database heel groot is? Inverted index? Full-text seach?

ZOEKOPDRACHT OPSTUREN

Er is een `input` nodig, dus een `form` in de View: er wordt bovenaan de pagina een formulier met zoekbox en button toegevoegd

```
<form asp-action="Index" method="get">
    Filter: <input type="text" name="filter" />
    <input type="submit" value="Filter" />
</form>
```

ZOEKOPDRACHT OPSTUREN (INCLUSIEF ONVERANDERDE SORTEEROPTIES)

Voeg een hidden veld toe om de sorteeropties te bewaren:

```
<form asp-action="Index" method="get">
    <input type="hidden" name="sorteerVolgorde" value='@ViewData["Sorteer"]' /
    Filter: <input type="text" name="filter" />
    <input type="submit" value="Filter" />
</form>
```

Geen post maar een get: de client moet de url kunnen bookmarken/sharen...

ONTHOUDEN FILTEROPTIES CLIENT-SIDE

HTML:

```
<form asp-action="Index" method="get">
    <input type="hidden" name="sorteerVolgorde" value='@ViewData["Sorteer"]' /
    Filter: <input type="text" name="filter" value='@ViewData["Filter"]' />
    <input type="submit" value="Filter" />
</form>
```

HTML:

```
<a asp-action="Index"
       asp-route-sorteerVolgorde="oplopend"
       asp-route-filter='ViewData["Filter"]'>↑</a>
```

C#:

```
ViewData["Filter"] = filter;
```

`ViewData`: Het filter wordt bewaard om te kopiëren naar de searchbox

PAGINEREN

DE NIEUWE ACTION

```
private IQueryable<Student> Pagineer(IQueryable<Student> lijst, int pagina)
{
    return lijst.Skip(10 * pagina).Take(10);
}
public async Task&lt;IActionResult&gt; Index(string sorteerVolgorde, string filter, int pagina)
{
    return View(await Pagineer(Filter(Sort(sorteerVolgorde), filter), pagina).ToListAsync());
}
```

- **pagina** begint bij 0
- Wat gebeurt er als **pagina** negatief wordt?
-
```
ViewData["Pagina"] = pagina;
```
- HTML:

```
<a asp-action="Index" asp-route-pagina="@((int)ViewData["Pagina"] - 1)"
asp-route-sorteerVolgorde='@ViewData["Sorteer"]'
asp-route-filter='@ViewData["Filter"]'>
    ←
</a>
<a asp-action="Index" asp-route-pagina="@((int)ViewData["Pagina"] + 1)"
asp-route-sorteerVolgorde='@ViewData["Sorteer"]'
asp-route-filter='@ViewData["Filter"]'>
    →
</a>
```

OOPS

- Zoek eerst op Alice.
- Ga dan naar pagina 2.
- Zoek dan op Bob.
- Is het gewenst dat de pagina op 2 blijft staan?
- In de form met de zoekopdracht, reset de pagina.

```
<form asp-action="Index" method="get">
    <input type="hidden" name="sorteerVolgorde" value='@ViewData["Sorteer"]' />
    <input type="hidden" name="pagina" value='0' />
    Filter: <input type="text" name="filter" />
    <input type="submit" value="Filter" />
</form>
```

- In de link voor de sorteervolgorde, reset de pagina.

```
<a asp-action="Index"
    asp-route-pagina="0"
    asp-route-sorteerVolgorde="oplopend"
    asp-route-filter='ViewData["Filter"]'>↑</a>
```

- Op een gegeven moment is de lijst leeg... Doorklikken hoort niet te kunnen...

EEN NIEUW MODEL MAKEN

We kunnen extra ViewData toevoegen, maar nu maken we liever een nieuw Model aan.

Compositie vs. overerving

Herriner:

```
private IQueryable<Student> Pagineer(IQueryable<Student> lijst, int pagina)
{
    return lijst.Skip(10 * pagina).Take(10);
}
```

We maken een speciaal soort lijst:

```
public class GepagineerdeList<T> : List<T>
{
    public int Pagina { get; private set; }
    public int PaginaAantal { get; private set; }
    public GepagineerdeList(List<T> lijstDeel, int totaalAantal, int pagina, int perPagina)
    {
        Pagina = pagina;
        PaginaAantal = (int)Math.Ceiling (totaalAantal / (double)perPagina);
        this.AddRange(lijstDeel);
    }
    public bool HeeftVorige() {  return Pagina > 0; }
    public bool HeeftVolgende() { return Pagina < PaginaAantal - 1; }
}
```

De Pagineer methode wordt een statische methode...

HET NIEUWE MODEL GEBRUIKEN

(Constructors kunnen niet async zijn)

```
public static async Task<GepagineerdeList<T>> CreateAsync(
    IQueryable<T> lijst, int pagina, int perPagina)
{
    return new GepagineerdeList<T>(
        await lijst.Skip(pagina * perPagina).Take(perPagina).ToListAsync(),
        await lijst.CountAsync(),
        pagina,
        perPagina);
}
```

En in de action:

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter, int pagina)
{
    return View(await GepagineerdeList<Student>.CreateAsync(Filter(Sort(sorteerVolgorde), filter), pagina, 3));
}
```

Verwijder alle ViewData!

HET NIEUWE MODEL GEBRUIKEN

(Constructors kunnen niet async zijn)

```
public static async Task<GepagineerdeList<T>> CreateAsync(
    IQueryable<T> lijst, int pagina, int perPagina)
{
    return new GepagineerdeList<T>(
        await lijst.Skip(pagina * perPagina).Take(perPagina).ToListAsync(),
        await lijst.CountAsync(),
        pagina,
        perPagina);
}
```

En in de action:

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter, int pagina)
{
    return View(await GepagineerdeList<Student>.CreateAsync(Filter(Sort(sorteerVolgorde), filter), pagina, 3));
}
```

Verwijder alle ViewData!

HET NIEUWE MODEL GEBRUIKEN

(Constructors kunnen niet async zijn)

```
public static async Task<GepagineerdeList<T>> CreateAsync(
    IQueryable<T> lijst, int pagina, int perPagina)
{
    return new GepagineerdeList<T>(
        await lijst.Skip(pagina * perPagina).Take(perPagina).ToListAsync(),
        await lijst.CountAsync(),
        pagina,
        perPagina);
}
```

En in de action:

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter, int pagina)
{
    return View(await GepagineerdeList<Student>.CreateAsync(Filter(Sort(sorteerVolgorde), filter), pagina, 3));
}
```

Verwijder alle ViewData!

HET NIEUWE MODEL GEBRUIKEN

(Constructors kunnen niet async zijn)

```
public static async Task<GepagineerdeList<T>> CreateAsync(
    IQueryable<T> lijst, int pagina, int perPagina)
{
    return new GepagineerdeList<T>(
        await lijst.Skip(pagina * perPagina).Take(perPagina).ToListAsync(),
        await lijst.CountAsync(),
        pagina,
        perPagina);
}
```

En in de action:

```
public async Task<IActionResult> Index(string sorteerVolgorde, string filter, int pagina)
{
    return View(await GepagineerdeList<Student>.CreateAsync(Filter(Sort(sorteerVolgorde), filter), pagina, 3));
}
```

Verwijder alle ViewData!

OOPS: DISPLAYFORNAME WERKT ALLEEN VOOR LIST

- Gebruik .FirstOrDefault() om een element of leeg element te pakken, of
- maak een IEnumerable van het Model en cast daarna waar nodig.

GEBRUIK MAKEN VAN HEEFTVORIGE EN HEEFTVOLGENDE

```
class='btn btn-default @(Model.HeeftVorige() ? "" : "disabled")'
```

ALTERNATIEVEN/EXTRAS

- `AsNoTracking`
- Geef alle data door aan de gebruiker, en blader client-side door de data heen (nadeel=performance, voordeel=performance)
- Maak een 'sessie' aan server-side zodra de gebruiker een zoekopdracht doet om te voorkomen dat **de resultaten verschuiven tijdens het bladeren**

SCAFFOLDEN VAN RELATIES

ONE-TO-MANY

```
public class Klas
{
    public int Id { get; set; }
    public string Naam { get; set; }
}
public class Student
{
    public int Id { get; set; }
    public string Naam { get; set; }
    public Klas Klas { get; set; }
}
```

2x scaffolden!

Migraties runnen en database updaten.

Helaas wordt de relatie niet helemaal gescaffold

Wat gebeurt er als Klas Required is? Probeer: voeg toe:

```
public int KlasId { get; set; }
```

Een error bij Update-Database? Haal de student weer weg.

Nu een error bij het toevoegen van de student... De Klas moet geset worden.

ONE-TO-MANY: DROPDOWN

In de Students/Create

- In de View:

```
<select asp-for="KlasId"
        asp-items="@(new SelectList((IEnumerable<Klas>)ViewData["Klassen"], "Id", "Naam"))">
    <option>Kies een klas</option>
</select>
```

- In de Action: `[Bind("Id,Naam,KlasId")]`

Voeg ook een kolom toe in de Students/Index.

DE FOUTE MANIER

In de Klas/Details:

- In de View:

```
@foreach (Student s in (IEnumerable<Student>)ViewData["Studenten"])
{
    <a asp-controller="Klas"
        asp-action="VerwijderStudent"
        asp-route-student="@s.Id">
        Verwijder @s.Naam</a>
}
```

- Maak de action VerwijderStudent aan.
- Pas op! Dit is een link, dus wordt deze opnieuw uitgevoerd als de pagina wordt ververst. Gebruik liever PRG.
- Alternatieve oplossing: i.p.v. bij elke aanpassing een request, verwerk alleen de aanpassingen aan het eind.

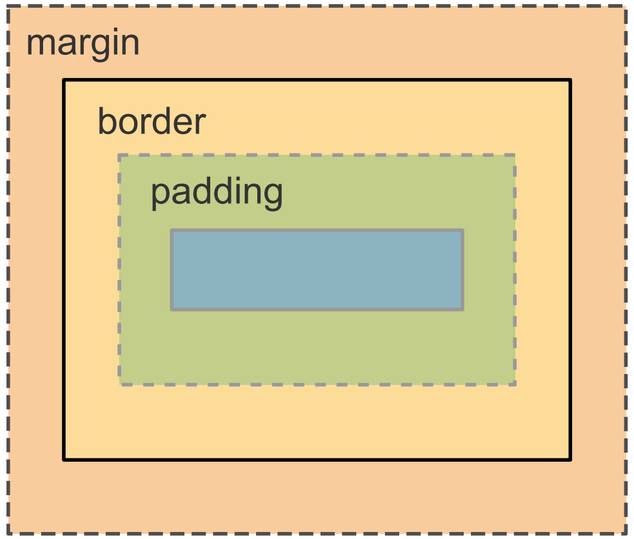
TOETS BESPREKEN

Even samen door de toets heen lopen

1A: MARGINS, PADDINGS EN BORDERS

Live

Schets:



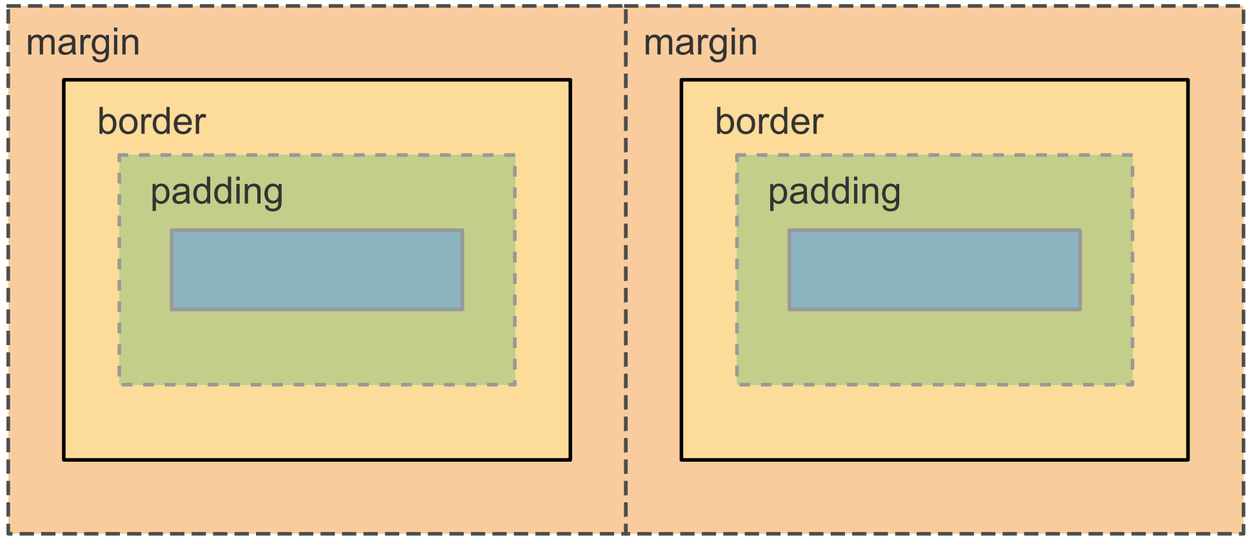
De afstand tussen de doosjes is 2px + 16px = 18px

De afstand tussen de tekst is alles bij elkaar = 63px

1A: MARGINS, PADDINGS EN BORDERS

Live

Schets:



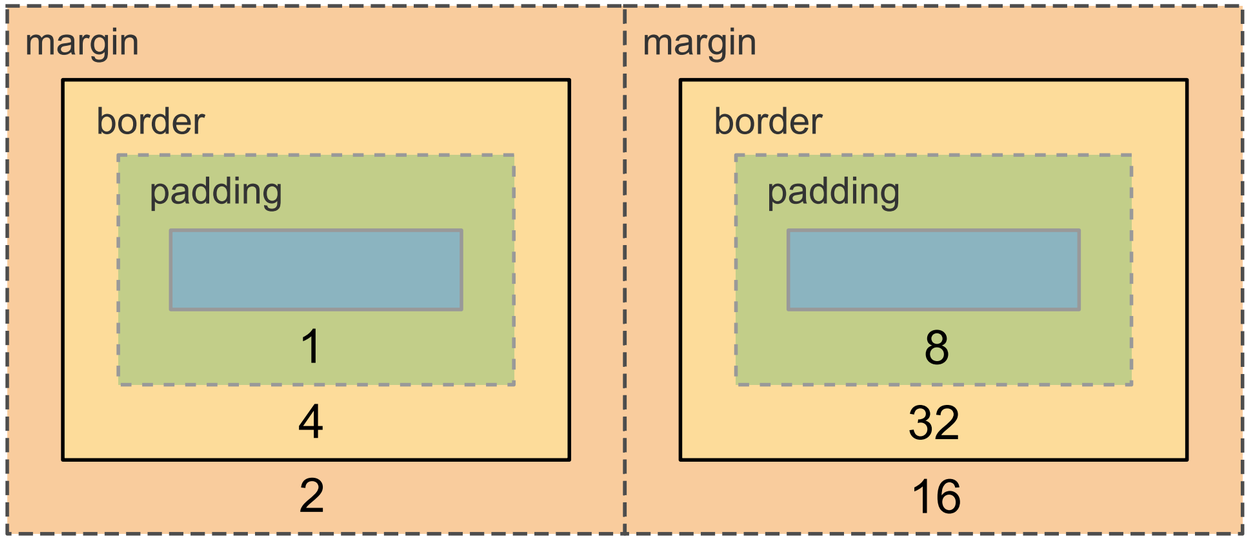
De afstand tussen de doosjes is 2px + 16px = 18px

De afstand tussen de tekst is alles bij elkaar = 63px

1A: MARGINS, PADDINGS EN BORDERS

Live

Schets:



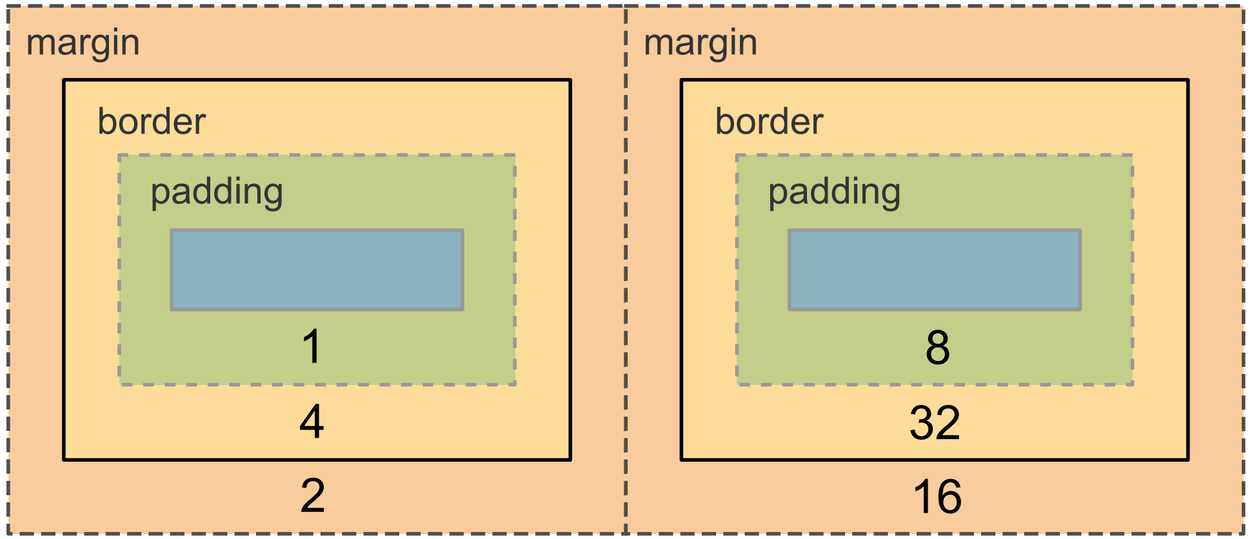
De afstand tussen de doosjes is 2px + 16px = 18px

De afstand tussen de tekst is alles bij elkaar = 63px

1A: MARGINS, PADDINGS EN BORDERS

Live

Schets:



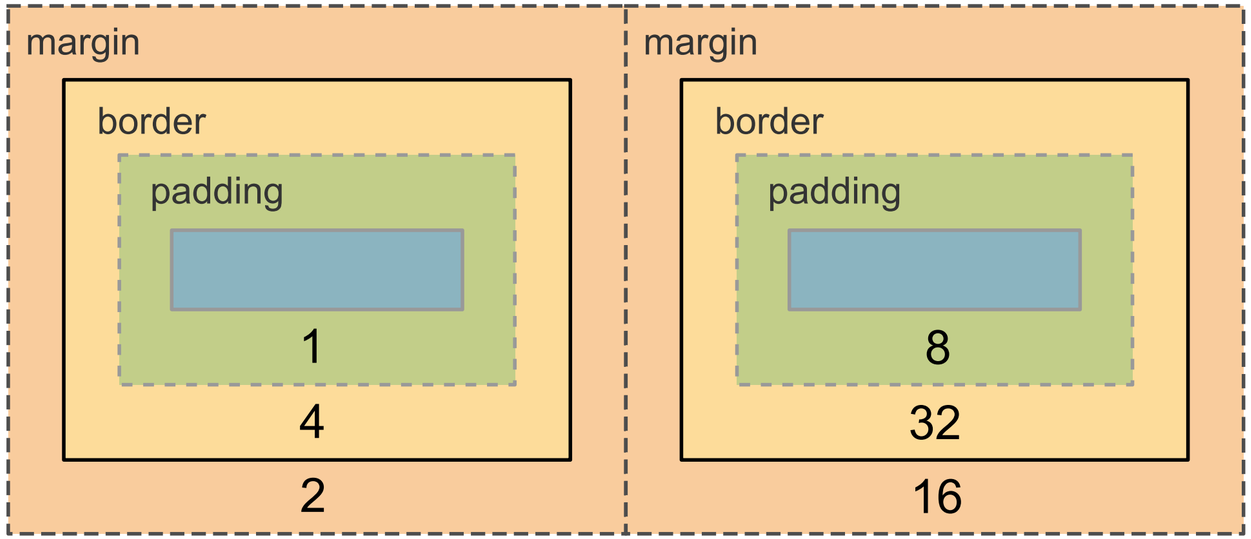
De afstand tussen de doosjes is 2px + 16px = 18px

De afstand tussen de tekst is alles bij elkaar = 63px

1A: MARGINS, PADDINGS EN BORDERS

Live

Schets:



De afstand tussen de doosjes is 2px + 16px = 18px

De afstand tussen de tekst is alles bij elkaar = 63px

1B: BORDER-BOX

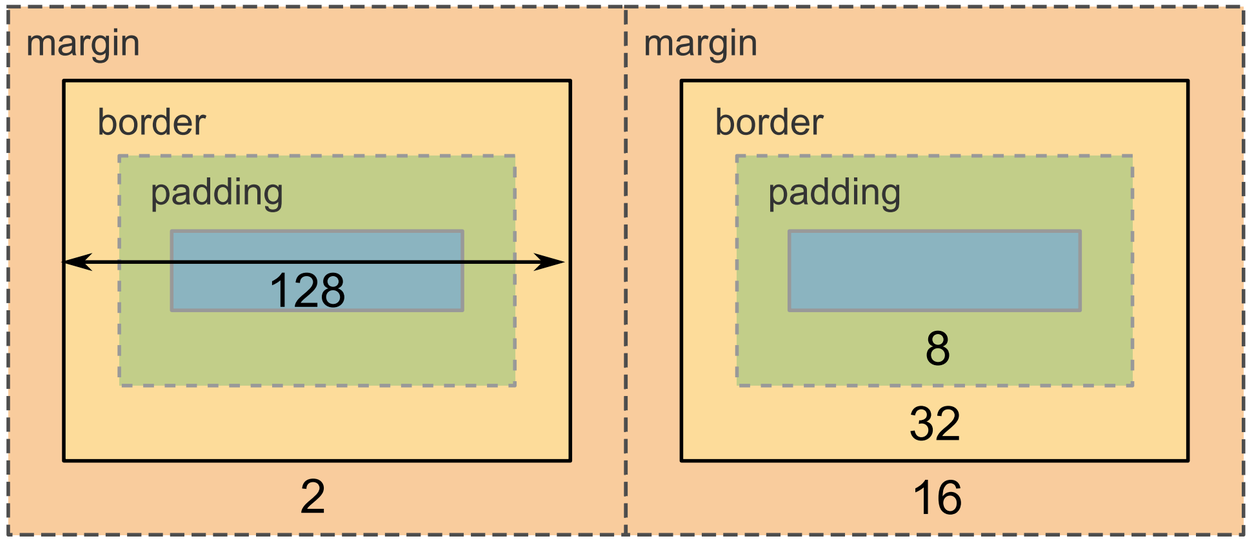
Live

Het box-sizing attribuut heeft alleen effect als de width is opgegeven.

1C: BORDER-BOX

Live

Schets:

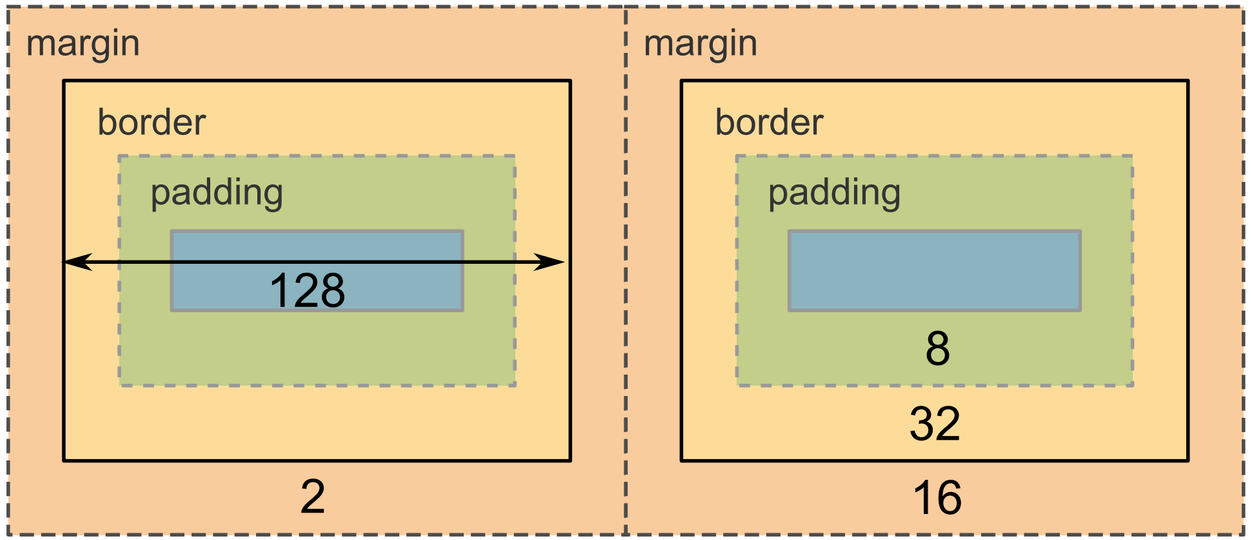


De totale breedte, min de inhoud, is 2 + 128 + 2 + 2 * (8 + 32 + 16)=244

1C: BORDER-BOX

Live

Schets:



De totale breedte, min de inhoud, is 2 + 128 + 2 + 2 * (8 + 32 + 16)=244

1D: HEADER

Lesdoel: *je kunt begijpt wat semantische elementen zijn en kunt HTML elementen in content categorieën indelen.*

De header tag is er niet om stijl aan te geven, maar is er voor semantiek. Screenreaders en bots 'begrijpen' dat dit de header is. Het ziet er niet anders uit.

1E: @MEDIA VOORWAARDEN

Lesdoel: *je kent de CSS at-regel @media en begrijp je hoe je daarvan gebruik kunt maken om de website op verschillende devices aantrekkelijk te blijven tonen.*

Lesdoel: *je kunt de breedte van een element beïnvloeden door in CSS gebruik te maken van relatieve width of max-width.*

De volgende code

```
@media screen and (min-width: 500px) { body { width: 500px; } }
@media screen and (max-width: 500px) { body { width: 100%; } }
```

Is hetzelfde als

```
body { width: 100%; max-width: 500px; }
```

1F: BOOTSTRAP COLS

Live

Lesdoel: *Je weet wat responsiveness van een website is en je kunt responsiveness met een grid-framework implementeren.*

```
<div class="container">
    <div class="row">
      <div class="col-md">
        <img style="width:100%" src="..."/>
      </div>
      <div class="col-md">
        <img style="width:100%" src="..."/>
      </div>
      <div class="col-md">
        <img style="width:100%" src="..."/>
      </div>
    </div>
</div>
```

1G: SCSS

Lesdoel: *Je begrijpt waarom SCSS bestaat en wat de relatie is met CSS.*

Performance / compatibilitiet

2A: PROPERTY OVERRIDEN

Property overriden kan. Voorbeeld van toepassing: de setter van achternaam (uit Persoon) van een GetrouwdPersoon verandert ook de achternaam van de partner.

2B: OO

1. Fout: een interface kan je niet instantieren
2. Fout: een `List<Base>` is niet een speciaal soort `List<Derived>`
3. Fout: een `List<Derived>` is niet een speciaal soort `List<Base>`
4. Goed

2C: LINQ

Runtimefout.

2D: LINQ

```
var p = personen.LastOrDefault (p => p.Naam == "Jan" && p.Leeftijd < 18)
```

2E: SINGLE

Error als er niet precies 1 is.

2F: ANONIEM TYPE

```
persoon => var { NaamLengte = persoon.Naam.Length }
```

2G: DEFERRED EXECUTION

Er zijn geen compiletime fouten. ToList() uitvoeren voordat Take wordt uitgevoerd is traag, omdat Select en take lazy zijn, en tolist breekt deze lazyness. Alleen 1 is een lijst, de rest is een IEnumerable.

2H: LINQ

Alternatieven zijn mogelijk!

```
Select(godsdienst => godsdienst.AantalGelovigen).Sum()
```

2I: TE MOEILIJK

```
Select(godsdienst => godsdienst.Gelovigen.Select(
    gelovige => godsdienst.Equals(godsdiensten.Where(
        godsdienst2 => godsdienst2.Gelovigen.Contains(gelovige)).First()))).Sum
```

3A: UNIT TESTEN

Met Theory kun je één testmethode meerdere keren uitvoeren met verschillende inputwaardes.

3A: UNIT TESTEN

```
[Fact]
public void Test1()
{
    Calculator c = new Calculator();
    Xunit.Assert.Equal(3, c.Add(1, 2));
}
```

4A: FORMULIER

de URL https://nu.nl/login?password=..., get moet post worden

4B: ROUTING

http://www.mijnwinkel.nl/WinkelMandje/Verwijder/123

of

http://www.mijnwinkel.nl/WinkelMandje/Verwijder?id=123

4C: PRG

Tekening:

4D: COMMUNICATIE CONTROLLER-VIEW

```
@model IEnumarable<Student>
```

4E: RAZOR

Zet `@if (Model.Count > 0) { … }` om de ul heen.

4F: VIEWMODELS

- maak een nieuw (view)model met de lijst en het aantal prive meldingen, of
- stuur de hele lijst met meldingen naar de view en bereken daar het aantal prive meldingen.

OPDRACHT

- Maak een Student model aan met de volgende properties:
  - Id
  - Naam
  - Lengte

  Maak een Cursus model aan met de volgende properties:
  - Id
  - Naam
- Maak nu een **veel op veel** (*in de les een op veel*) relatie aan tussen Student en Cursus
- Scaffold de controller en views
- Voor de student Index:
  - Maak een zoekfunctie
  - Implementeer paginering
  - Maak het mogelijk om **te sorteren op Id, of Naam of Lengte** (*in de les alleen op naam*). (uitdaging: bij gelijk gesorteerd, sorteer op een andere kolom)
  - Laat zien per student welke cursussen hij/zij volgt.
- Zorg dat er ook front-end is om studenten aan cursussen toe te voegen en te verwijderen.