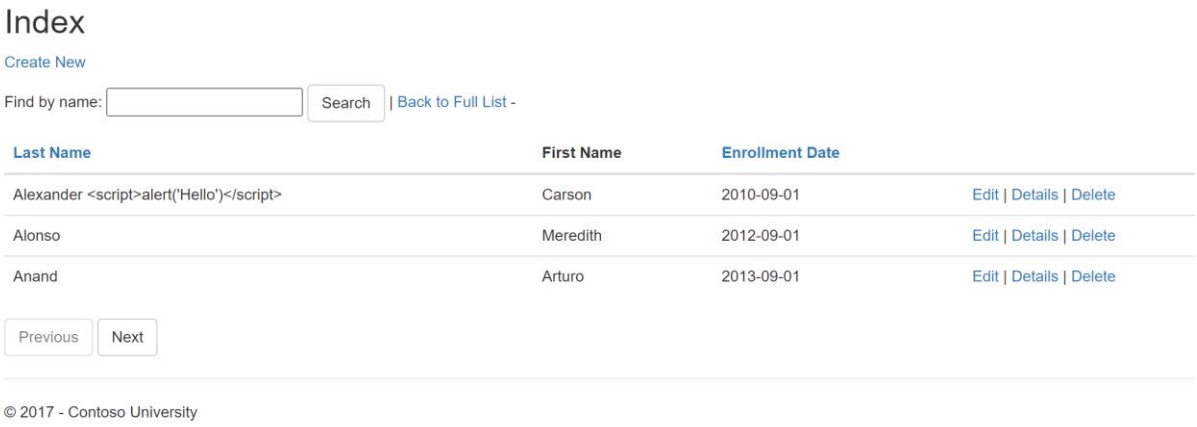
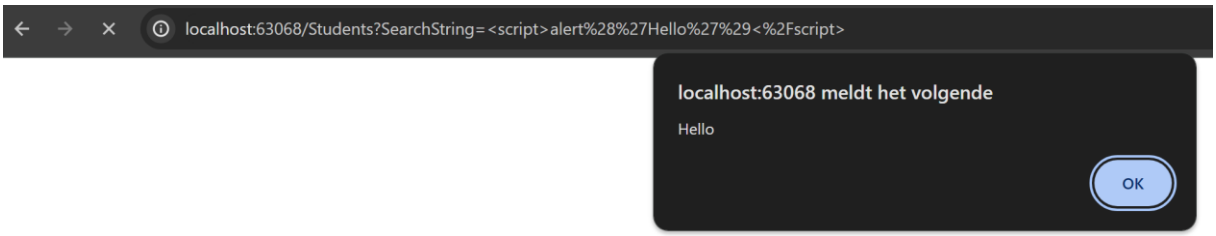
# CST1 - Practicum Les2 – XSS

## XSS door kwetsbaarheid in server side code

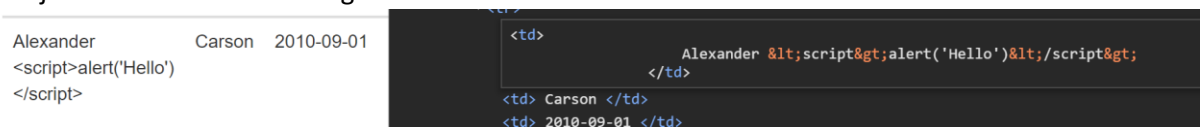
1. *Wijzig de naam van de student naar een naam waarin XSS inzit (=html script tag). Zijn de pagina's waarin de naam getoond wordt kwetsbaar voor XSS? Leg uit waarom wel of niet kwetsbaar.*
   De naam van de studenten is encoded. Hierdoor kan een gebruiker wel XSS aan een naam toevoegen, maar dit niet via deze manier laten werken. http://localhost:63068/Students zelf is niet vulnerable voor XSS.




   Als je de zoek functie gebruikt is XSS wel mogelijk. Het is alleen de vraag of je dit nog als de zelfde pagina ziet of niet. Als dit wel het geval is kan je zeggen dat deze pagina vulnerable is voor XSS.



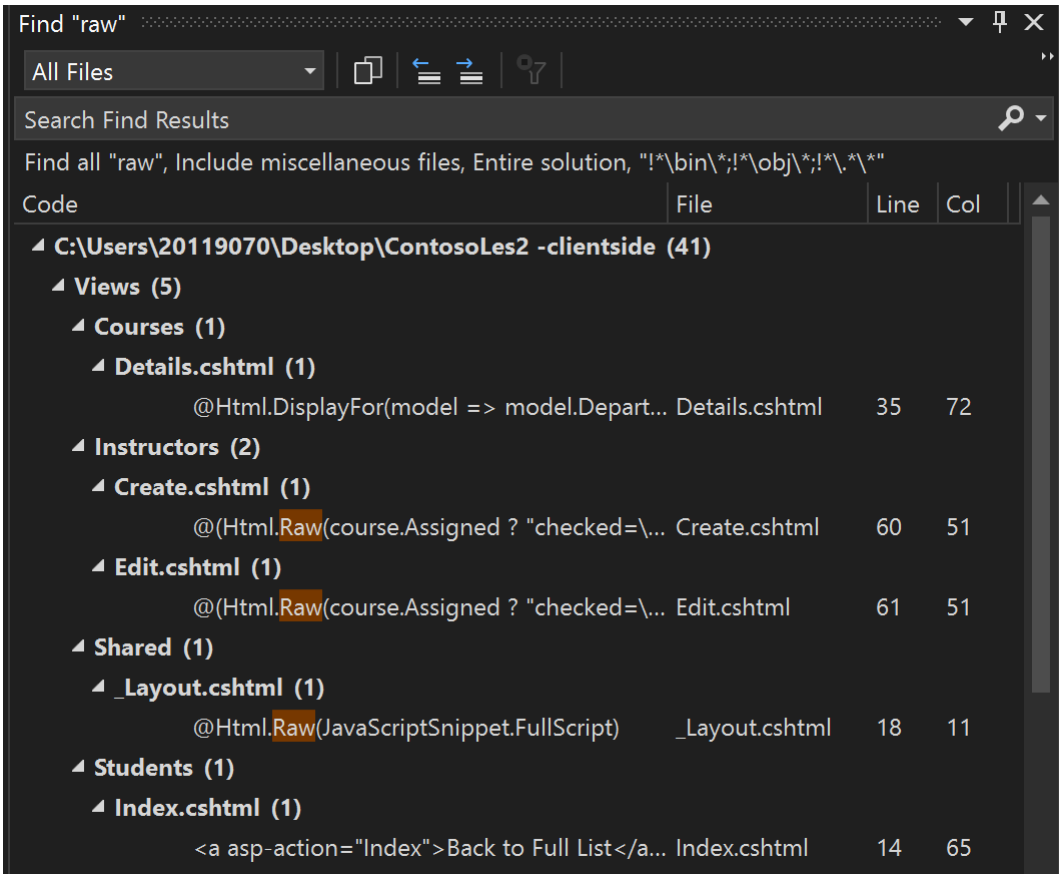

2. *Toon via F12 aan (element en/of netwerk tab) dat er encoding wordt toepast*
   Via de netwerk tab kan je op de rechter muisknop drukken om de html code van een naam te bekijken. We zien dan het volgende:




   We zien hier "<" in plaats van "<". Dit is de HTML encoding. Er wordt dus encoding toegepast.

3. *Waarom is het zoeken naar een server side XSS erg tijdrovend als je op de manier van vraag 1 op zoek gaat een XSS in een webapplicatie?*
   Je zou met de manier van vraag 1 voor elke pagina op elk veld moeten kijken of er wel of niet encoding wordt gebruikt.

4. In de Razor view kan je data ook tonen zonder encoding (zie sheets). Zoek naar deze methode via: [Edit],[Find and Replace],[Find in files] . Op welke plekken wordt deze methode gebruikt?




5. *Toon aan dat er een XSS zit in een view van de map courses*
In de view van de map courses zie ik html.raw staan. Deze code is gevaarlijk aangezien er geen HTML encoding aanwezig is. Er is dus hier XSS aanwezig.

```
<dd>
    @Html.DisplayFor(model => model.Department.Name) -   @Html.Raw(Model.Department.Name)
</dd>
```

Ik heb een department aangemaakt met de naam “ XSS ”. Hierna heb ik een course aangemaakt met deze aangepaste department. Zodra ik dan op details druk is het volgende te zien.

Course

| | |
|---|---|
| **Number** | 1 |
| **Title** | Testat |
| **Credits** | 1 |
| **Department** | XSS - **XSS** |

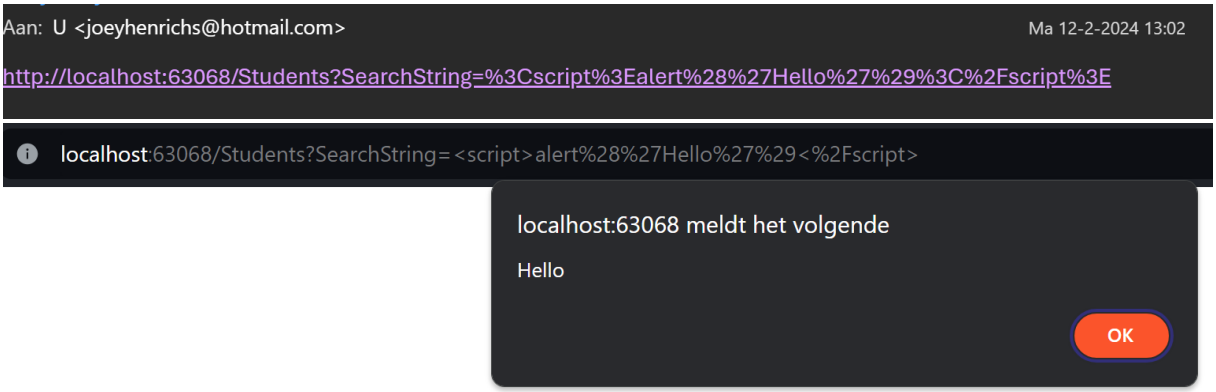
Edit | Back to List

XSS is dus aanwezig op de course map.

6. *Is dit een stored of een reflected XSS? Leg uit waarom*
   Dit is een stored XSS. De aanpassing van de naam van de department wordt opgeslagen in de database. Vervolgens kan elke andere gebruiker dit inladen.

7. *Wat voor een XSS ziet in het zoeken van een student? Leg uit waarom?*
   Bij het zoeken van een student is er spraken van reflected XSS. Dit komt omdat de zoekfunctie gebruikt maak van een link. Deze link kan vervolgens gedeeld worden met anderen.

8. Kan je deze XSS kwetsbaarheid ook aantonen als je een mail naar je zelf stuurt en dan op een link in deze mail klikt?
   Zodra ik deze link naar mijzelf mail, kan ik de XSS gebruiken. Zo toon ik aan dat het reflected XSS is.




## Client side code en REST

9. *Bekijk de coursesapicontroller. Deze biedt een REST interface aan conform de REST conventies. Welke acties kan je allemaal uitvoeren middels de code in deze controller?*
   De acties die je kan uitvoeren zijn: GET, PUT, POST en DELETE. Deze acties zijn te vinden in de code.

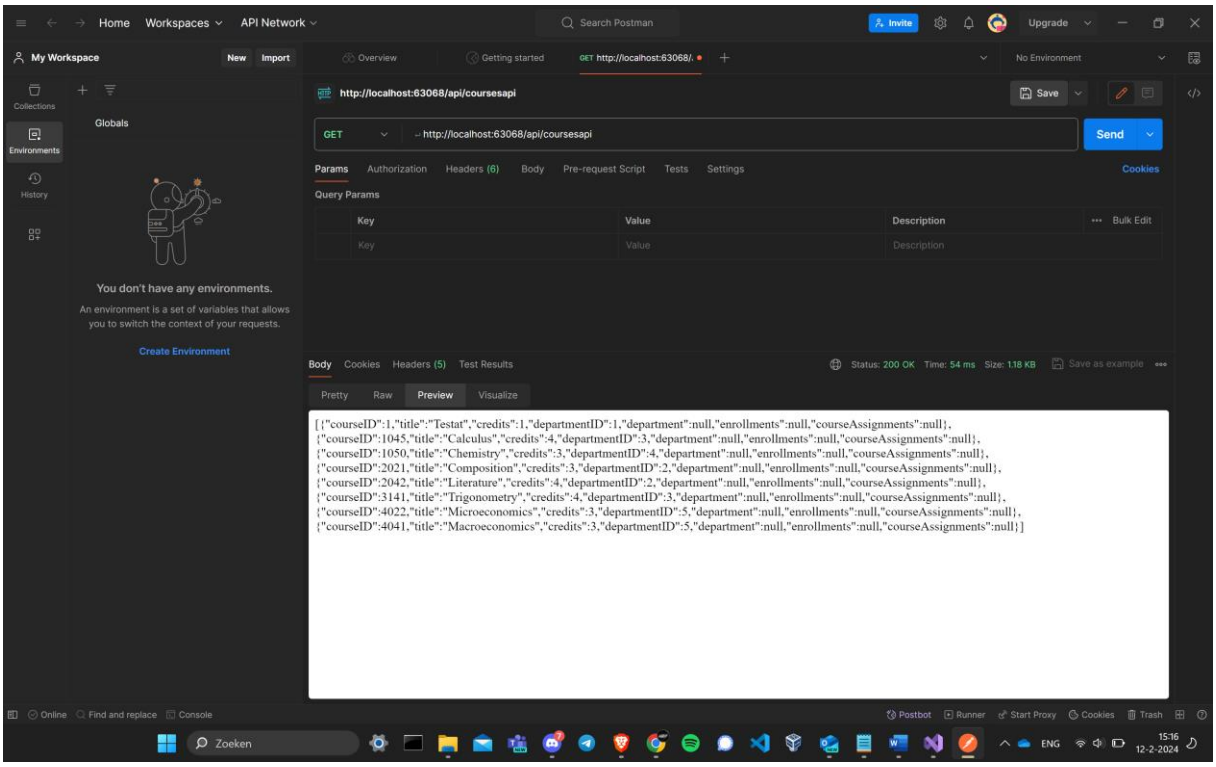
```
// GET: api/CoursesApi
[HttpGet]
0 references
public IEnumerable<Course> GetCourses()
{
    return _context.Courses;
}

// GET: api/CoursesApi/getCoursesWithDepartment
[HttpGet("getCoursesWithDepartment")]
0 references
public IActionResult GetCourses2()
{
    var result = _context.Courses.Include(c => c.Department);
    return Json(result);
}

// GET: api/CoursesApi
[HttpGet("getCoursesWithEnrollment")]
0 references
public IEnumerable<Course> GetCourses3()
{
    return _context.Courses.Include(e => e.Enrollments);
}

// GET: api/CoursesApi/5
[HttpGet("{id}")]
0 references
public async Task<IActionResult> GetCourse([FromRoute] int id)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var course = await _context.Courses.SingleOrDefaultAsync(m => m.CourseID == id);

    if (course == null)
    {
        return NotFound();
    }

    return Ok(course);
}
```

```
// PUT: api/CoursesApi/5
[HttpPut("{id}")]
0 references
public async Task<IActionResult> PutCourse([FromRoute] int id, [FromBody] Course course)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    if (id != course.CourseID)
    {
        return BadRequest();
    }

    _context.Entry(course).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!CourseExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}
```

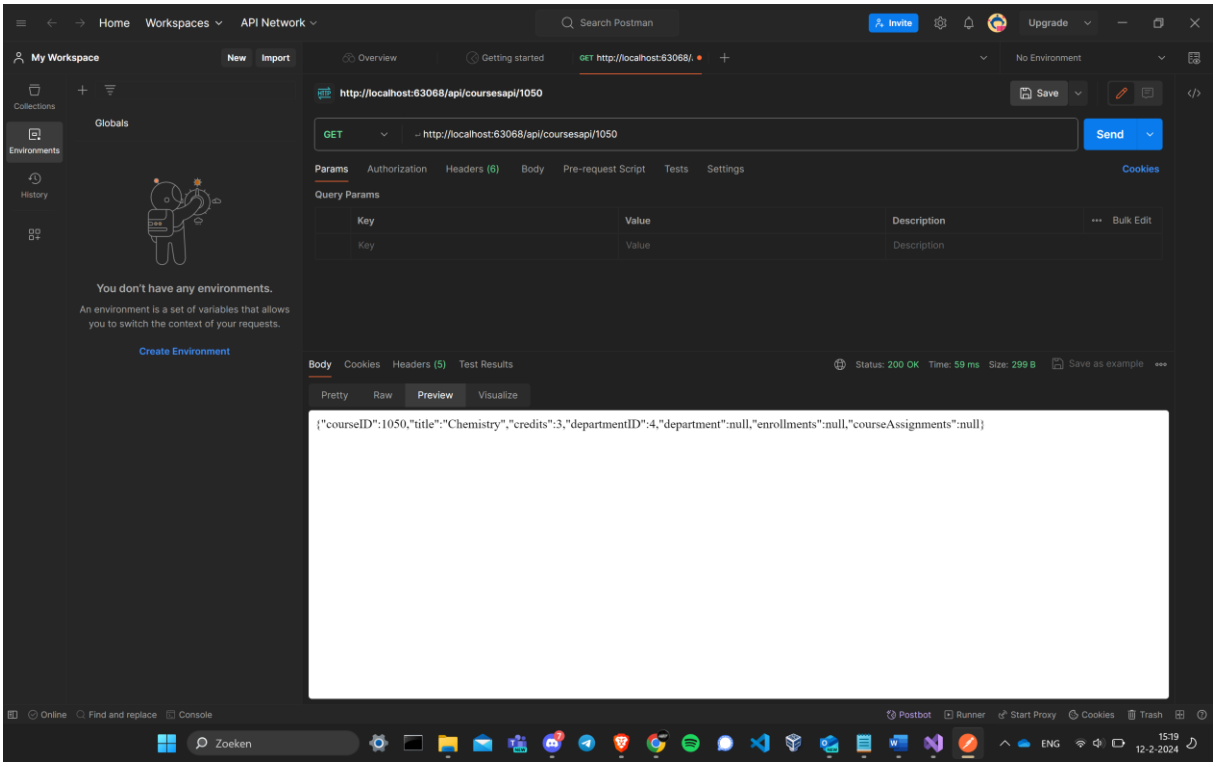
```
        // POST: api/CoursesApi
        [HttpPost]
        0 references
        public async Task<IActionResult> PostCourse([FromBody] Course course)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            _context.Courses.Add(course);
            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateException e)
            {
                if (CourseExists(course.CourseID))
                {
                    return new StatusCodeResult(StatusCodes.Status409Conflict);
                }
                else
                {
                    throw;
                }
            }

            return CreatedAtAction("GetCourse", new { id = course.CourseID }, course);
        }

        // DELETE: api/CoursesApi/5
        [HttpDelete("{id}")]
        0 references
        public async Task<IActionResult> DeleteCourse([FromRoute] int id)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var course = await _context.Courses.SingleOrDefaultAsync(m => m.CourseID == id);
            if (course == null)
            {
                return NotFound();
            }

            _context.Courses.Remove(course);
            await _context.SaveChangesAsync();

            return Ok(course);
        }

        2 references
        private bool CourseExists(int id)
        {
            return _context.Courses.Any(e => e.CourseID == id);
        }
    }
}
```

10. *Maak via Postman een http request om alle courses te laten en een request om 1 course te laten zien*

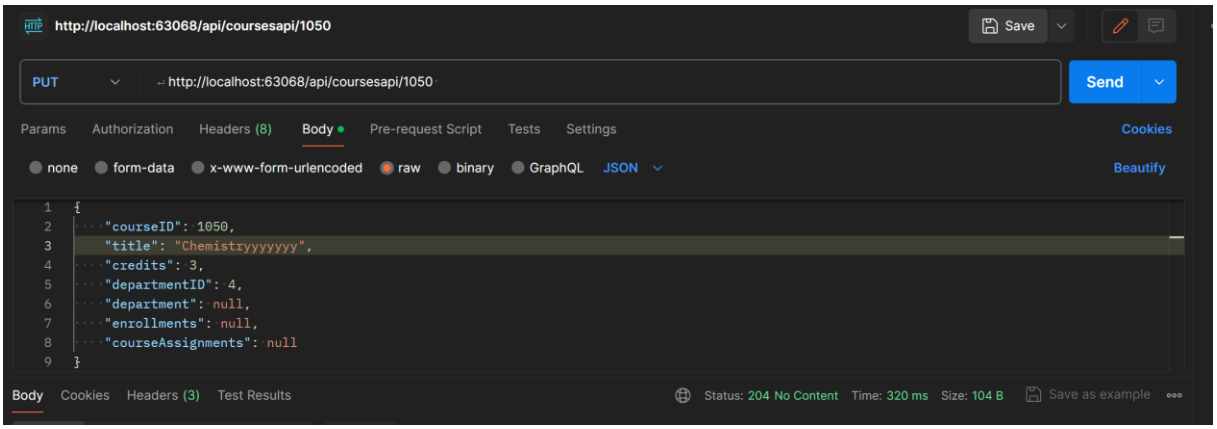
Om alle courses te weergeven heb ik “http://localhost:63068/api/coursesapi” gebruikt.

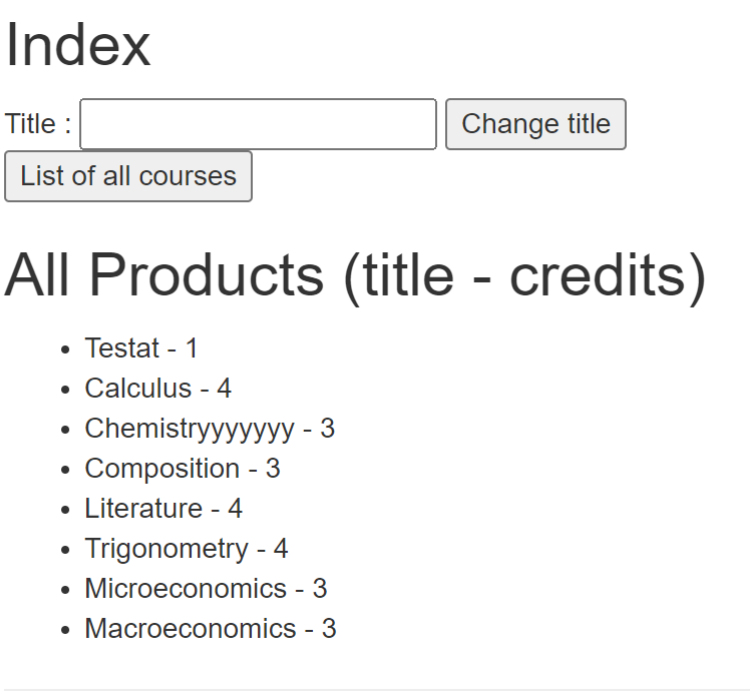



Nu om 1 specifieke course te zien gebruik je dezelfde link alleen nu met het course id erachter. Om bijvoorbeeld Chemistry te laten zien gebruik je “http://localhost:63068/api/coursesapi/1050”.

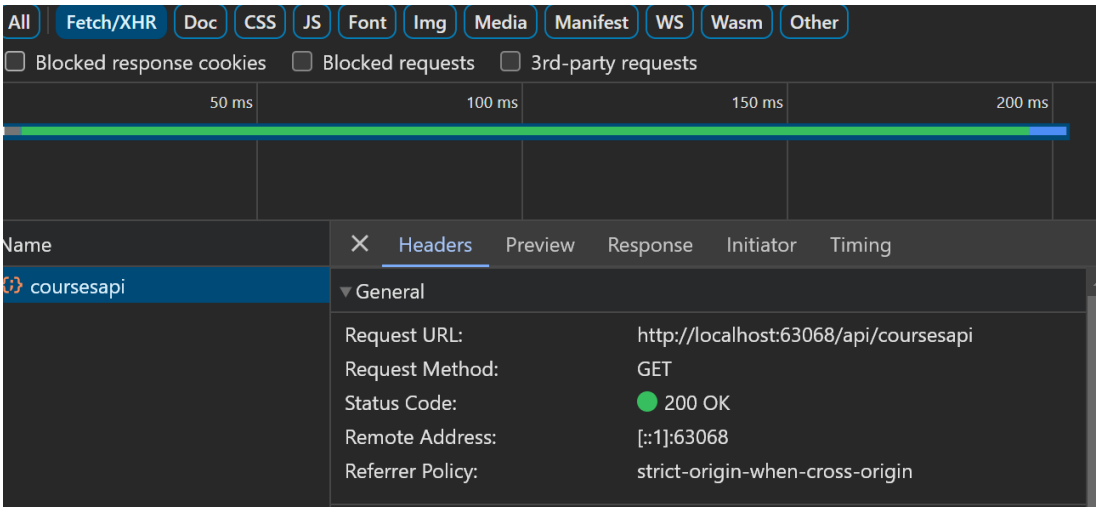


11. Wijzig de naam van een bestaande course via Postman
    Allereerst heb ik postman op PUT methode gezet. PUT aangezien we een aanpassing aan de website willen brengen. Ik heb de code van de vorige losse course overgenomen en in de body van mijn request gezet. Hierdoor geef je mee wat er met de PUT methode meegaat. Dan kan je hier de aanpassingen maken en om ze door te voeren druk je op send.

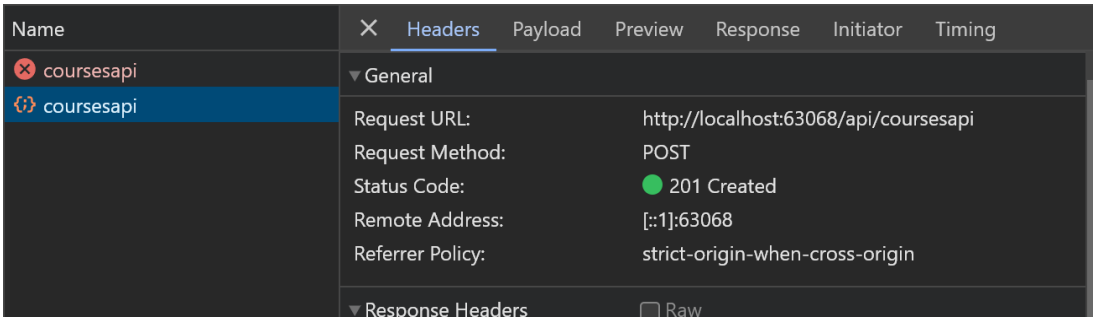



12. Klik in de webapplicatie op de menu optie Rest en klik dan op “List of all courses”. Bekijk via de netwerk tab van F12 wat er gebeurt?
    De data vanuit “http://localhost:63068/api/coursesapi” wordt opgevraagd via een GET method. Deze wordt dan getoond op de pagina.




13. Ga naar: http://localhost:63068/GuiCoursesApi/create en voeg een course toe. Bekijk weer wat er gebeurt in de netwerk tab. Waar zit de code in het visual studio project die de course toevoegt?
    Er vind een POST method plaats die de course opslaat.




Deze code is te vinden onder “C:\Users\20119070\Desktop\ContosoLes2-clientside\Controllers\CoursesApiController.cs”. Ik weet dat dit de code is aangezien dit het enigste stuk is van api/CourseApi dat gebruik maakt van een POST request.

```
// POST: api/CoursesApi
[HttpPost]
0 references
public async Task<IActionResult> PostCourse([FromBody] Course course)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    _context.Courses.Add(course);
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateException e)
    {
        if (CourseExists(course.CourseID))
        {
            return new StatusCodeResult(StatusCodes.Status409Conflict);
        }
        else
        {
            throw;
        }
    }

    return CreatedAtAction("GetCourse", new { id = course.CourseID }, course);
}
```

## XSS door kwetsbaarheid in de client side code

14. *Ga naar de pagina: http://localhost:63068/GuiCoursesApi/ zit er een XSS kwetsbaarheid in het veranderen van de titel. Welke code past de titel aan?*
    Er zit een XSS kwetsbaarheid in het aanpassen van de titel. Dit is te zien door een XSS uit te voeren. Dit is bij de volgende afbeelding te zien.

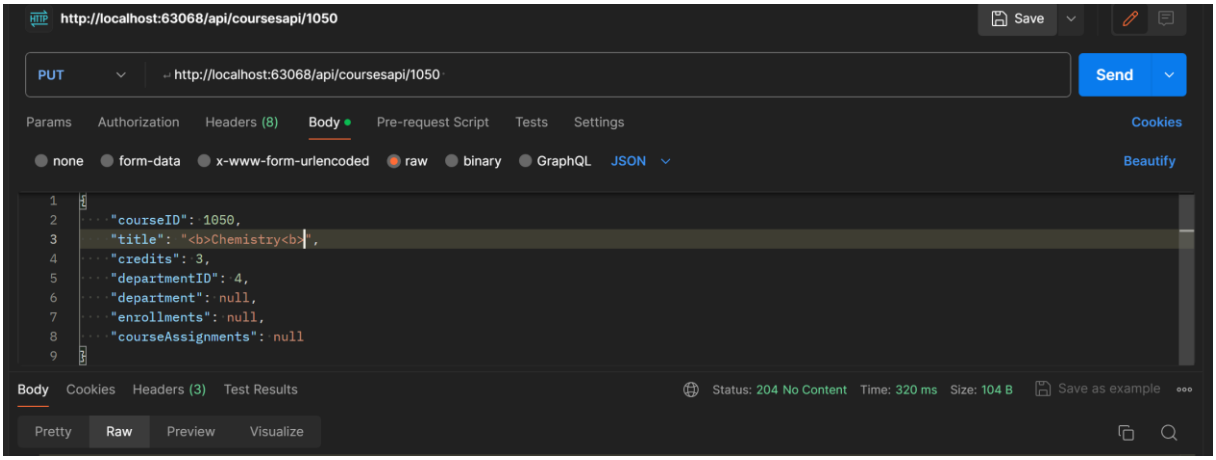

    De code die de titel aan past is te vinden in “C:\Users\20119070\Desktop\ContosoLes2 - clientside\Views\GuiCoursesApi\Index.cshtml”.

```
<h2 id="titel">Index</h2>
<div> Title : <input type="text" id="inputtitle" />
<input type="button" value="Change title" onclick="test();" />
</div>
    <div>
        <input type="button" value="List of all courses" onclick="getList();" />
    </div>
<div>
    <h2>All Products (title - credits)</h2>
    <ul id="products" ></ul>
</div>

<script>
    function test() {
        var input = $("#inputtitle").val();
        $('#titel').html(input);
    }

    function getList() {
        $.ajax({
            url: "/api/coursesapi",
            method:'GET',
            dataType:'json',
            success: function (result) {
                //$("h2").text(result[0].title);

                $.each(result, function (key, item) {
                    // Add a list item for the product.
                    $('<li>', { html: formatItem(item) }).appendTo($('#products'));
                })
            },
            error: function (xhr) {
                $('h2').text("An error occured: " + xhr.status + " " + xhr.statusText);
            }
        })

    }

        function formatItem(item) {
            return item.title + " - " + item.credits;

        }


</script>
```

15. *Hoe kan je deze XSS kwetsbaarheid oplossen? Zie https://www.w3schools.com/js/js_jquery_elements.asp*
    Om deze kwetsbaarheid op te lossen kan encoding worden toegepast. De input wordt dan alleen als tekst gezien. Hierdoor heb je niet meer een XSS kwetsbaarheid

16. Als je alle courses laten zien zoals in vraag 12: Is deze code kwetsbaar voor XSS? Is dit een stored XSS of een ander soort XSS? Waar zit het probleem in de code?
    Het weergeven van deze code is kwetsbaar voor XSS. Dit is een stored XSS. Dit weet ik aangezien ik een aanpassing maak aan de course die wordt opgeslagen in de database.

Om dit te doen heb ik de coursenaam van chemistry aangepast met XSS in de naam. Ik kreeg dan chemistry dikgedrukt te zien.




# Index

Title : Change title

List of all courses

# All Products (title - credits)

- Testat - 1
- Test Course - 1
- Calculus - 4
- **Chemistry - 3**
- Composition - 3
- Literature - 4
- Trigonometry - 4
- Microeconomics - 3
- Macroeconomics - 3

Ik zie dat de namen van de courses niet encoded zijn. Hierdoor zijn deze vatbaar voor XSS.