

.NET Okruženje

Vježba 10: API, AJAX, Dropzone

Sadržaj

Web API	3
Dohvat podataka	3
Create i Update	4
Brisanje zapisa	5
AJAX	6
Dropzone	8

Web API

ASP.NET WebApi je mehanizam pomoću kojeg možemo osigurati pristup podacima koji se nalaze na našem serveru, ili ih naš server može na neki način generirati. Naprotiv, „obični“ ASP.NET MVC je mehanizam pomoću kojeg korisniku kao rezultat vraćamo kompletnu HTML stranicu koju može otvoriti u manje-više bilo kojem pregledniku. U verziji .NET Core radnog okvira ne postoji odvojeni API controller i MVC controller – već je controller identičan. Samim time, može se kombinirati korištenje „običnih“ MVC stranica i dohvaćanje podataka putem API-ja.

ASP.NET WebApi se najčešće koristi u 3 scenarija:

- Iz javascript-a pozivamo API servise koji nam dobavljaju podatke koji se kasnije prikazuju korisnicima
- Server komunicira s drugim serverom putem API metoda
- Mobilni uređaj ili desktop aplikacija komunicira s API servisima

Preko API poziva moguće je napraviti nešto od sljedećeg:

- Dohvatiti rezultate, eventualno po parametru (query ili id, metoda GET)
- Spremiti novi rezultat (POST)
- Urediti postojeći (PUT)
- Obrisati zapis (DELETE)

U prijašnjim verzijama ASP.NET MVC radnog okvira bilo je potrebno naslijediti klasu ApiController kako bi se mogla koristiti u kontekstu Web API mehanizma, međutim od .NET Core verzije koristi se ista klasa ali ju je potrebno označiti s **[ApiController]** anotacijom. Ovaj korak nije nužan, ali služi u kompleksnijim aplikacijama za razlikovanje Web API od „običnog“ MVC sučelja.

ClientApiController.cs

```
[Route("api/client")]
[ApiController]
public class ClientApiController : Controller
{
    private ClientManagerDbContext _dbContext;
```

Dohvat entiteta je najlakše implementirati i isprobati, stoga krećemo od njega.

Dohvat podataka

API funkcije Get u pravilu kao rezultat vraćaju objekt koji se serijalizira u traženi format – bilo JSON, bilo XML. Danas je JSON popularniji od XML-a jer se lakše konzumira u JavaScript jeziku, a i dosta je „lakši“ od ipak prilično glomaznog XML-a.

Zadatak 10.1

Implementirati API metode za dohvat podataka o klijentima.

1. Kreirati **ClientApiController**
 - Osigurati da se do akcija unutar tog controller-a dolazi

- Kreirati **ClientDTO** klasu sa podacima: ID, FullName, Address, City (CityDTO klasa), Email
- 2. Metoda **Get()** bez parametara koja vraća kolekciju svih ClientDTO objekata iz baze podataka. Ova metoda se poziva kao /api/client
 - Pretvorbu iz Client -> ClientDTO napraviti u samoj Get() metodi
- 3. Metoda **Get(id)** koja vraća jedan ClientDTO objekt s ispravnim ključem. Ova metoda se poziva kao /api/client/1
- 4. Metoda **Get(string q)** koja vraća sve ClientDTO objekte koji unutar naziva imaju traženi pojam q (jednostavna pretraga). Ova metoda se poziva kao /api/client/pretraga/q
 - Koristi atribut kako bi se definirala ruta
- 5. **Dodatno pitanje:** Kako ponovno iskoristiti kod za pretvorbu Client -> ClientDTO?

Sve gornje metode testirati u nekom od popularnih Internet preglednika kao što je primjerice [IE7](#).

Create i Update

Ranije je spomenuto da jedan od osnovnih mehanizama koji čine ASP.NET MVC tako efikasnim je princip automatskog kreiranja i povezivanja modela s obzirom na poslane vrijednosti. Isti principi vrijede i za Web API – vrijednosti poslane putem forme ili putem JSON objekata se također na isti način povezuju u C# objekte koji su zatim dostupni unutar controllera.

JSON objekt	C# Company objekt
<pre>{ "id": 1, "name": "Encian", "email": "xsa@purusDuntum.ca", "address": "577-6124 Ading Avenue", "latitude": 47.83972, "longitude": 48.45743, "dateFrom": "2014-03-18T00:00:00", "cityID": 478512556, "city": { "id": 478512556, "postalCode": "XD9 6GR", "name": "Siverek" } }</pre>	<pre>public class Company { public int ID { get; set; } public string Name { get; set; } public string Email { get; set; } public string Address { get; set; } public decimal Latitude { get; set; } public decimal Longitude { get; set; } public DateTime? DateFrom { get; set; } public int CityID { get; set; } public City City { get; set; } }</pre>

Model Binding mehanizam će iz strukture ulaznih podataka (u ovom slučaju JSON) pročitati sva polja (properties) i u objektu odgovarajuće klase ih povezati po imenu. Velika prednost takvog povezivanja je upravo u tome što će se tipovi podataka automatski pretvoriti u odgovarajuće tipove u C#, a kompleksniji property će također biti rekurzivno povezan (u gornjem primjeru to je property City).

Kako bi se povezivanje ispravno obavilo, potrebno je naznačiti iz kojeg se izvora popunjava model – u donjem slučaju to je tijelo zahtjeva, te je potrebno naznačiti **[FromBody]** anotaciju:

ClientApiController.cs

```
...
public async Task<ActionResult<ClientDTO>> Put(int id, [FromBody] Client model)
...
```

Zadatak 10.2

Potrebno je u **ClientApiController** klasi dodati metode za kreiranje novog i spemanje promjena na nekom postojećem podatku.

1. Post metoda služi za kreiranje nove vrijednosti
 - a. Poziv putem POST poziva na **/api/client**
2. Put metoda će spremiti promjene na postojećoj vrijednosti
 - a. Poziv putem PUT poziva na **/api/client/id**
3. U oba slučaja model treba biti tipa Client, i automatski se povezati iz POST podataka u JSON formatu
4. Isprobati funkcionalnost na način da se u Chrome doda plugin Advanced REST Client ili Postman.

Brisanje zapisa

Metoda kontrolera koja briše neki zapis izgleda po prilici ovako:

ClientContorller.cs

```
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    // ...

    return Ok();
}
```

Zadatak 10.3

- Dodati metodu za brisanje kojoj se proslijeđuje ID klijenta koju treba obrisati.
- U akciji controllera pozvati odgovarajuću metodu iz repozitorija.
- Provjeriti je li klijent uistinu obrisao tako da se nakon toga pozove dohvat svih klijenata.

AJAX

Upravo jQuery se često koristi u kombinaciji s PartialView-om za dohvaćanje podataka putem ajax poziva. Štoviše, često je jednostavnije koristiti upravo takve ajax pozive umjesto MVC ajax formi jer nude veću fleksibilnost i slobodu implementacije. Protok podataka izgleda po prilici ovako:

1. Poziva se `jQuery.ajax({...})`
2. Zahtjev dolazi na server, u odgovarajuću kaciju controller-a
3. Obrađujemo zahtjev, i kao povratnu vrijednost vraćamo:
 - a. JSON ako želimo u javascriptu obraditi te podatke i napraviti nešto:
`return Json(new { x = 5 }, JsonRequestBehavior.AllowGet);`
 - b. PartialView ako želimo novi element ubaciti na stranicu:
`return PartialView("_Filter", model);`
4. Na klijentu (u javascriptu koji je inicirao ajax poziv), obrađujemo rezultat

Zadatak 10.4

Filter forma koja trenutno radi na način da se radi potpuni POST na server, potrebno je modificirati da ne radi potpuni refresh stranice (full POST), već da dohvati podatke koje treba prikazati pomoću još jedne AJAX funkcije.

1. Modificirati Index.cshtml tako da se iscrtavanje same tablice izdvoji u **_IndexTable** partial view.
 - a. Za skupljanje podataka s forme koristiti funkciju **serialize**
 - b. U filter formi, staviti da je gumb za pretragu u obliku „button type=button“ kako ne bi automatski slao formu na server.
 - c. Prilikom inicijalizacije DOM elementa, zakačiti „click“ event na gumb za pretragu unutar filter forme i tek u tom event u napraviti \$.ajax poziv s podacima koji se sa forme skupe pomoću „serialize“ funkcije
2. Dodati akciju **IndexAjax[HttpPost]** koja poziva **PartialView()** i vraća **_IndexTable** PartialView kao rezultat
 - a. Nova **IndexAjax** funkcija treba primiti jedan parametar tipa **ClientFilterModel**
 - b. Postojeću akciju Indeks preraditi da ne prima nikakav parametar i ne vrši filtriranje
3. U Javascript-u koristiti **replaceWith** funkciju kako bi staru tablicu zamijenili novim podacima dobivenim ajax pozivom. Da bi to bilo moguće, najbolje je dodati id atribut na tablicu (u HTMLu) i koristiti taj selector (primjerice, `#tbl-clients`)

Schema za pomoć rješavanju se nalazi na slici ispod:

Application name Home About Contact Pregled kompanija Pregled gradova Register Log in

Pregled kompanija

on('click',...) događa se ajax poziv i dohvaća nova tablica sa servera (renderira se na serveru pomocu _IndexTable partial viewa)

[Dodaj novu kompaniju](#)

ol Pretraga po adresi Pretraga po e-mailu Pretraga po gradu [Pretraga](#)

#	Naziv	Adresa	Datum osnivanja	Email	Grad	Akcija
1	Dolor Dolor Associates	577-6124 Adipiscing Avenue	18.3.2014.	vestibulum.massa.rutrum@purusDuisselementum.ca	Siverek	Uredi
13	Dolor Fusce Mi Ltd	P.O. Box 457, 5685 Orci, St.	10.7.2014.	ornare.tortor@commodo.edu	Campili	Uredi
22	Vivamus Molestie PC	807-2541 Et Rd.	6.1.2015.	eu.ligula@magna.net	Anlier	Uredi
39	Volutpat Nunc LLC	2171 Sed Rd.	15.6.2014.	Morbi@Phasellus.com	Goulburn	Uredi
46	Fusce Mollis Duis Inc.	811-9915 Nulla. St.	3.9.2014.	semper@ametluctus.edu	Cairo Montenotte	Uredi
78	Molestie Foundation	Ap #804-8697 Aliquam Rd.	29.12.2014.	auctor	Arviat	Uredi
92	Eget Mollis Industries	805 Ut Street	26.9.2014.	Aenean	Buren	Uredi
98	Molestie Dapibus Consulting	P.O. Box 160, 4289 Donec Street	13.2.2015.	felis@nunc	Municipal District	Uredi

© 2016 - My ASP.NET Application

Index.cshtml view

_IndexTable.cshtml partial view

Konceptuano rješenje zadatka

- Unutar Index.cshtml, potrebno je dodati javascript funkciju (primjerice, performFilter)
 - Ovu funkciju je potrebno pozvati kad se klikne na gumb „pretraga“
- Unutar funkcije performFilter:
 - **[client]** Dohvatiti filter formu koristeći neki od jQuery selektora
 - **[client]** Napraviti \$.ajax poziv, tako da se popune sljedeći parametri
 - url – koristiti @Url.Action() da se definira na koji URL se šalje forma
 - method – „POST“
 - data – koristiti jQuery.serialize() da bi se polja filter forme pretvorila u podatke koje zna koristiti \$.ajax funkcija
 - **[server]** Sve ostaje manje više isto (model se automatski puni), ali je razlika u tome što se treba koristiti PartialView prilikom vraćanja rezultata kako se ne bi iscrtavao cijeli _Layout (meni, header, footer, itd...)
 - **[client]** U callback od ajax poziva (success), dobivamo HTML sa servera i potrebno je zamijeniti trenutni <table> s podacima koji su došli sa servera

Dropzone

Nije rijedak slučaj da je potrebno omogućiti upload datoteka na server. Kako bi upload datoteka bio jednostavan i intuitivan, korisno je koristiti neku od poznatijih biblioteka za upload datoteka. Jedna od jednostavnijih takvih biblioteka je Dropzone (<https://www.dropzonejs.com>).

Navedena komponenta funkcionira na način da asinkrono šalje datoteke na server na URL koji se zadaje kao parametar – spremanje datoteka i pridruživanje ispravnim entitetima je odgovornost poslužitelja.

Donji odsječak koda omogućuje upload slike klijenta, uz ograničenje najviše jedne datoteke:

ClientContorller.cs

```
<div class="row">
  <div class="col-md-4">
    <form asp-action="Edit">
      <input type="hidden" asp-for="ID" />
      <partial name="_CreateOrUpdate" />
    </form>
  </div>
  <div class="col-md-6">
    <label class="control-label">Photo</label>
    <form id="profilePhotoDz" asp-controller="Client" asp-
action="UploadProfilePhoto"
      asp-route-clientId="@Model.ID"
      enctype="multipart/form-data" class="dropzone"></form>
  </div>
</div>

@section Scripts{
  <link rel="stylesheet" href="~/lib/dropzone.css" />
  <script src="~/lib/dropzone.js"></script>

  <script type="text/javascript">
    Dropzone.options.profilePhotoDz = {
      success: function (file, response) {
        console.log({ file, response });
      }
    };
  </script>
}
```

Datoteke dropzone.js i dropzone.css su preuzete iz ovog primjera:

<https://gitlab.com/meno/dropzone/raw/master/website/examples/simple.html>.

Sučelje jednostavne Dropzone komponente je prikazano na slici dolje:

Popis klijenata / Uređivanje klijenta

Ime

Prezime

Email

Radno iskustvo (god)

Photo

Drop files here to upload

Nakon što se slika odabere, automatski se šalje na server u navedenu akciju Controllera koju smo specificirali prilikom definiranja forme:

```

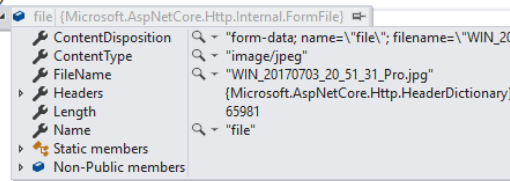
    }
    return View(model);
}

public IActionResult UploadProfilePhoto(int clientId, IFormFile file)
{
    // Spremanje datoteke

    return Json(new { success = true });
}

[ActionName("Edit")]
public IActionResult EditGet(int id)
{
    this.FillDropdownValues();
    return View(this._dbContext.Clients.FirstOrDefault(p => p.ID == id));
}

```



Zadatak 10.5

Omogućiti upload datoteke koja se veže uz nekog klijenta.

- Dodati klasu Attachment koja je vezana na klijenta (jedan klijent može imati više dokumenata)
 - Izvršiti migracijske skripte kako bi baza bila u skladu s promjenama modela
- Na formi za uređivanje (samo na Edit, ne na Create) dodati opciju uploada datoteka koristeći dropzone komponentu
 - Datoteke se prenose na server asinkrono. Akciju za upload nazvati **Client/UploadAttachment**. Treba biti moguće prenijeti više datoteka odjednom.
 - Spremiti uploadane datoteke na disk, a u bazu spremiti putanju do datoteke
- Nakon osvježavanja forme, potrebno je moći vidjeti popis uploadanih datoteka i po mogućnosti obrisati stare datoteke (AJAX poziv na akciju **Client/DeleteAttachment**)
 - Popis datoteka treba također dohvatiti AJAX pozivom nakon što se stranica učita (jQuery document.ready()). Akciju nazvati **Client/GetAttachments**, a kontrolu za prikaz nazvati **_AttachmentList**.