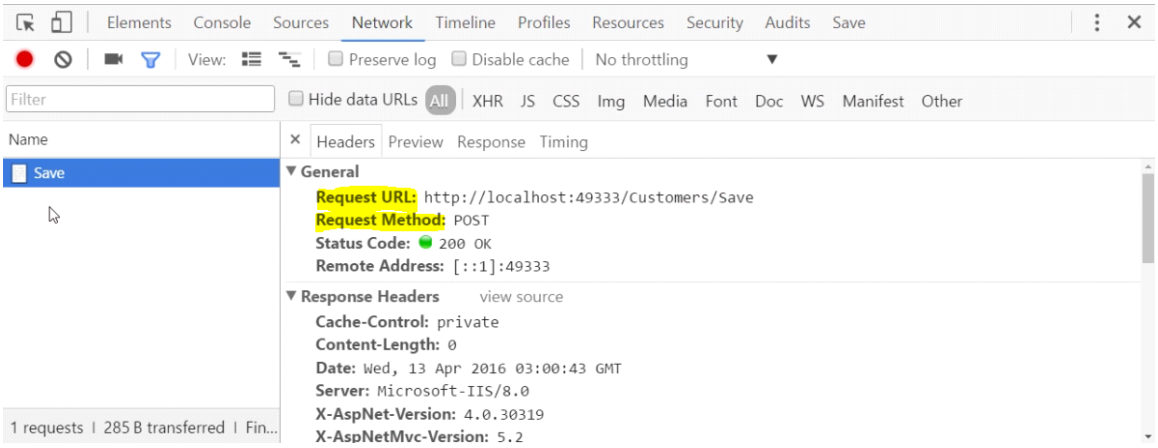
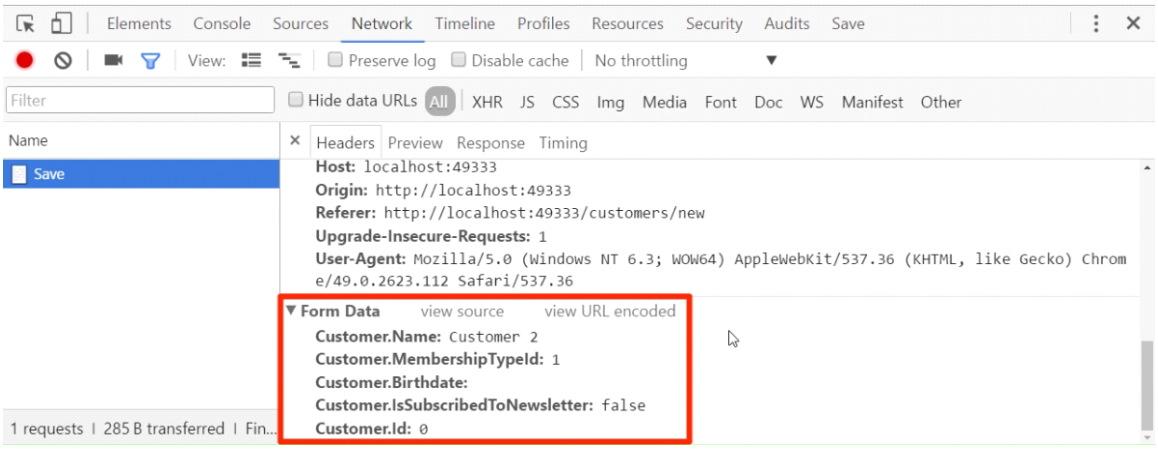
Anti-forgery Token

U Chrome-u Inspect > Network tab > Kliknemo Save na formi. VIdimo zahtev *Save*. U zaglavlju vidimo da je metoda POST i da ide u određen end-point *(localhost:49933/Customers/Save)*



Ispod, u telu, možemo videti podatke sa forme.



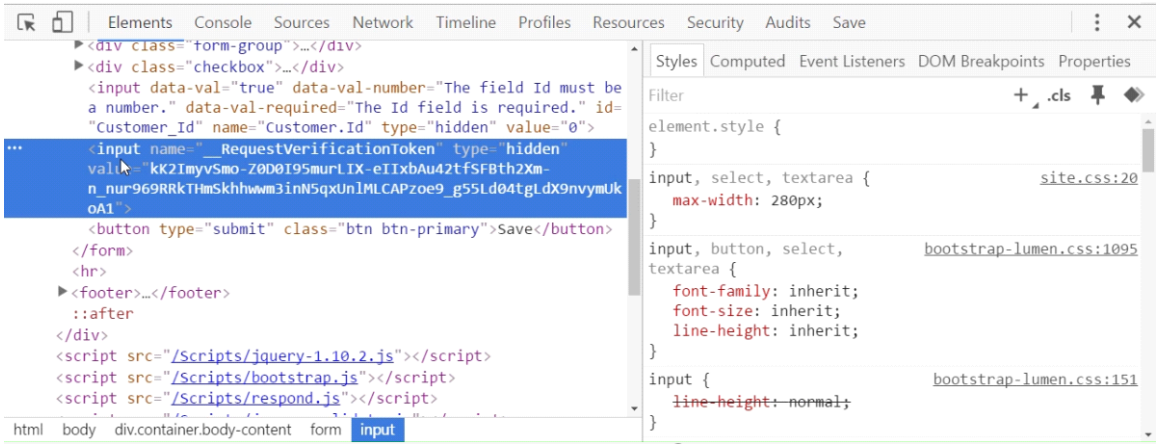
Sada zamislimo da korisnik koji je zadužen za pravljenje customer-a napusti sajt bez odjavljivanja. Dakle, ovaj korisnk ima aktivnu sesiju na serveru, tako da su i dalje autentifikovani za narednih par minuta (default podešavanje je oko 20 minuta).

Sada zamislimo da smo mi hakeri. Možemo prevariti ovog korisnika da poseti malicioznu stranu koju smo napravili *(Download X for free!)*. Na ovoj stranici možemo da postavimo image ili iframe i napišemo malo JavaScript koda tako da kada se stranica učita poslaće HTTP POST zahtev na end-point aplikacije.

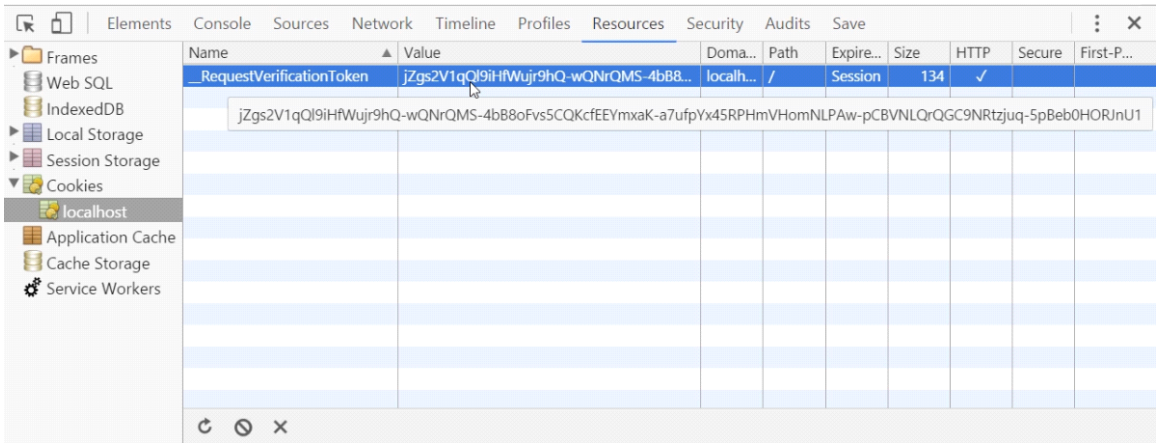
Pošto ovaj korisnik ima aktivnu sesiju na našoj aplikaciji, ovaj zahtev će proći u njegovo ime. Ovakav

napad se naziva **CSRF** (**Cross-site Request Forgery**). Haker "forge-uje" request na drugoj veb stranici, i sa ovom tehnikom može da izvrši bilo koje akcije u ime žrtve. Pored toga, čak i kada pogledamo log-ove svi zahtevi će izgledati legitimno, kao da su potekli za korisnikovog pretraživača.

Zato želimo da se osiguramo da request-ovi isključivo dolaze sa naše forme, a ne sa drugih stranica. Zato na našoj formi koristimo pomoćnu metodu **@Html.AntiForgeryToken()**. Ova metoda će napraviti token (koji je nalik tajnom kodu), i onda će ga postaviti kao hidden polje unutar forme i takođe kao cookie na korisnikovom računaru.



Ovde vidimo kako izgleda AntiForgeryToken na formi.



Ovde vidimo kolačić.

Uzećemo vrednosti iz skrivenog polja i kolačića i uporediti ih. Ako su iste, onda znamo da je to legitiman request. U suprotnom, u pitanju je napad, pošto ako haker redirektuje korisnika na njegovu stranu, on tamo nema pristupa skrivenom polju unutar naše forme, zato što to polje postoji samo kada korisnik poseti našu formu. Zato, čak iako haker ukrade cookie, on i dalje nema pristup skrivenom polju. Dakle, na serveru poredimo ove dve vrednosti, i ako nisu iste odmah prekidamo request.
Kod naše *Save* akcije dodamo DataAnnotation [ValidateAntiForgeryToken].

API

Kada request stigne našoj aplikaciji, MVC frejmvork prosleđuje taj request akciji/metodi unutar kontrolera. Ova akcija uglavnom vraća view, koji je potom parsiran od strane razor view engine-a i onda eventualno HTML zapis je vraćen klijentu. U ovom pristupu HTML markup je generisan na serveru, i onda vraćen klijentu.

Postoji alternativni način za generisanje HTML zapisa - možemo ga generisati na klijentu, tako da umesto da naše akcije vraćaju markup, one vraćaju sirove podatke. Prednosti ovog pristupa su manji trošak resursa servera (veća skalabilnost aplikacije, jer svaki klijent je odgovoran za generisanje svojih view-ova), pored toga sirovi podaci zauzimaju manje bandwith-a od HTML zapisa (poboljšanje performansi), ali ključna stvar je što imamo potencijal za mnogo veći opseg klijenata (npr. osim Web klijenta imamo i mobile i tablet aplikaciju). Ove aplikacije jednostavno pozivaju end-points naše aplikacije, uzimaju podatke i generišu view-ove lokalno. Ove krajnje tačke nazivamo Data Services ili **Web APIs** (**A**pplication **P**rogramming **I**nterface). Druge stranice i aplikacije mogu da konzumiraju naš API i dodaju nove funkcionalnosti i ukombinuju sa njihovim aplikacijama.

RESTful konvencija

Representational **S**tate **T**ransfer
HTTP GET zahtev se koristi za dobijanje resursa.
HTTP POST se koristi za pravljenje resursa.
HTTP PUT se koristi za ažuriranje resursa.
HTTP DELETE se koristi za uklanjanje resursa.
pr. : GET /api/customers