CARNET

Postman

CERT.hr-PUBDOC-2020-4-399

CERT.hr

Sadržaj

Ovaj dokument izradio je Laboratorij za sustave i signale Zavoda za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

1 Uvod

Postman je komercijalna aplikacija dostupna za operacijske sustave *Windows*, *macOS* i *Linux*. Prvenstveno je namijenjen testiranju i interakciji s REST API-jima, pri čemu njegovi autori tvrde da je riječ o jedinom alatu koji ima potpunu podršku za razvoj API-ja.

Postman se u početku koristio kao dodatak za *web* preglednik *Chrome*, ali s vremenom je prerastao u samostalnu aplikaciju. Korištenje *Postmana* kao dodatka *web* pregledniku više nije podržano.

Iako se prvenstveno koristi za razvoj i komunikaciju s RESTful API-jima (pridjev RESTful označava da API prati smjernice arhitekturnog stila REST) koji podatke razmjenjuju u formatu JSON, *Postman* podržava i API-je temeljene na SOAP i *GraphQL* protokolima, kao i ostale formate razmjene podataka poput XML-a.

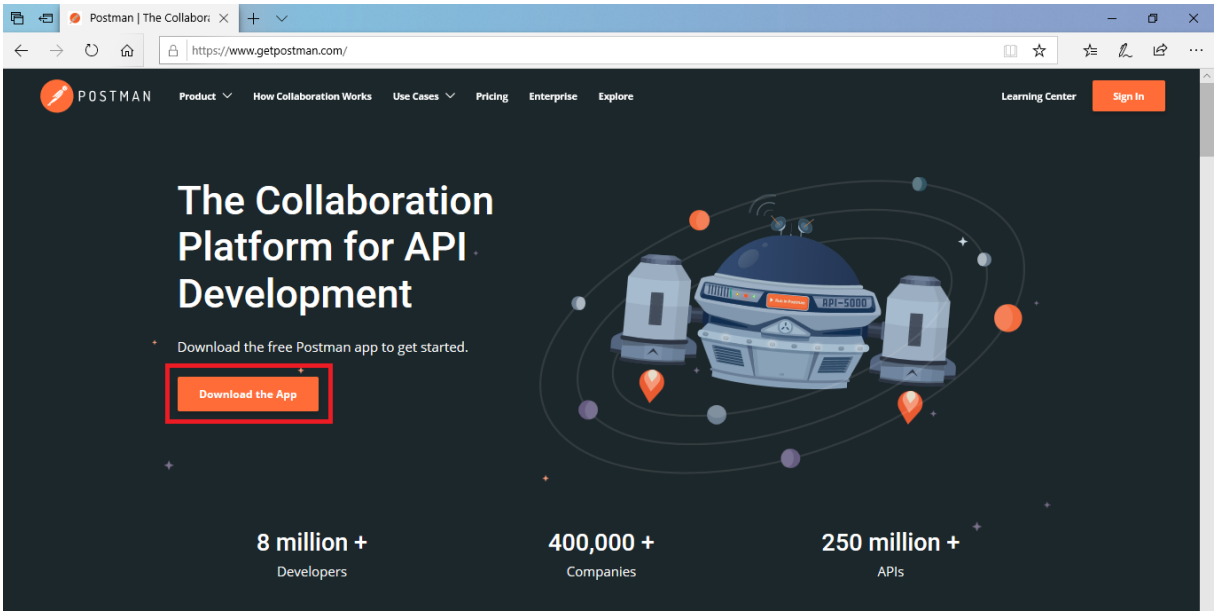
O REST API-jima se pisalo više u dokumentu Nacionalnog CERT-a Sigurnost HTTP REST API-ja i korisno je pročitati taj dokument prije čitanja ovog.

Postman je vrlo prilagodljiv, što ga čini pogodnim za (sigurnosno) testiranje jer se može automatizirati većina posla čak i kad se naiđe na odstupanje od standarda (npr. modificirana zaglavlja, HTTP metode koje nisu dio standarda, već proširenje itd.).

Također, *Postmanom* je moguće sastaviti automatizirane testove i uključiti ih u proces razvoja aplikacije kako bi programeri koji razvijaju API imali mehanizam koji prilikom testiranja može javiti da trenutna inačica aplikacije ima grešku, uspoređujući izlaz s očekivanim (ili prethodnim) izlazom.

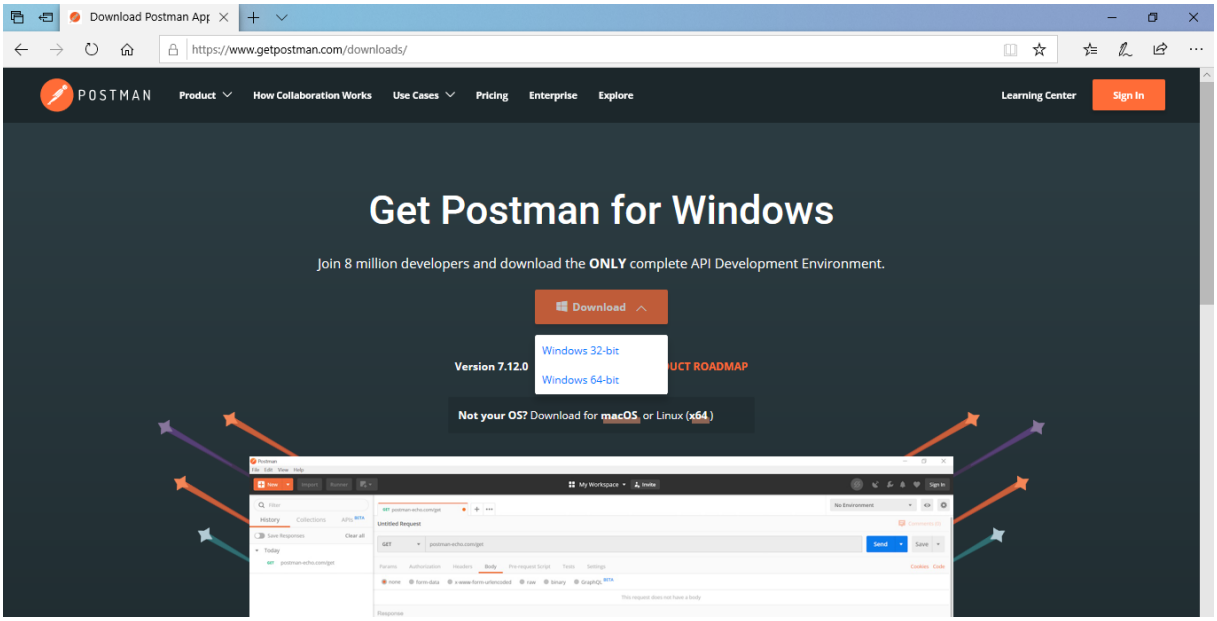
2 Instalacija alata *Postman*

Postman je inicijalno razvijen kao dodatak za *web* preglednik *Chrome* (taj dodatak danas više nije podržan), ali je s vremenom prerastao u samostalnu *desktop* aplikaciju. Dostupan je za preuzimanje sa službene stranice i moguće ga je instalirati na operacijske sustave *Windows*, *macOS* i *Linux* distribucije.



Slika 1 ***Početna stranica službene web stranice alata Postman***

U nastavku će se demonstrirati vrlo jednostavna instalacija na operacijski sustav *Windows 10*. Odabiremo odgovarajuću verziju za operacijski sustav (32-bitni ili 64-bitni) i klikom na tipku '*Download*' preuzimamo instalacijske datoteke.



Slika 2 ***Preuzimanje instalacijskih datoteka alata Postman***

Pokrećemo preuzetu instalacijsku datoteku. *Postman* se sam, bez ikakve interakcije s korisnikom, instalira i zatim pokreće.

Slika 3 ***Instalacija Postmana***

3 Korištenje alata *Postman*

3.1 Osnovne funkcionalnosti alata *Postman*

Postman je API klijent s raznim dodatnim funkcionalnostima, a koristi se za razvoj, testiranje, dijeljenje i dokumentiranje pretežito HTTP REST API-ja. Može se koristiti s registracijom ili bez nje, pri čemu je registracija korisna za timski rad. Za korištenje svih funkcionalnosti koje su demonstrirane u ovom dokumentu nije potrebna registracija.

Postman nudi velik broj funkcionalnosti i uistinu je riječ o bogatoj i potpunoj razvojnoj okolini za API-je. Neke od najvažnijih funkcionalnosti su:

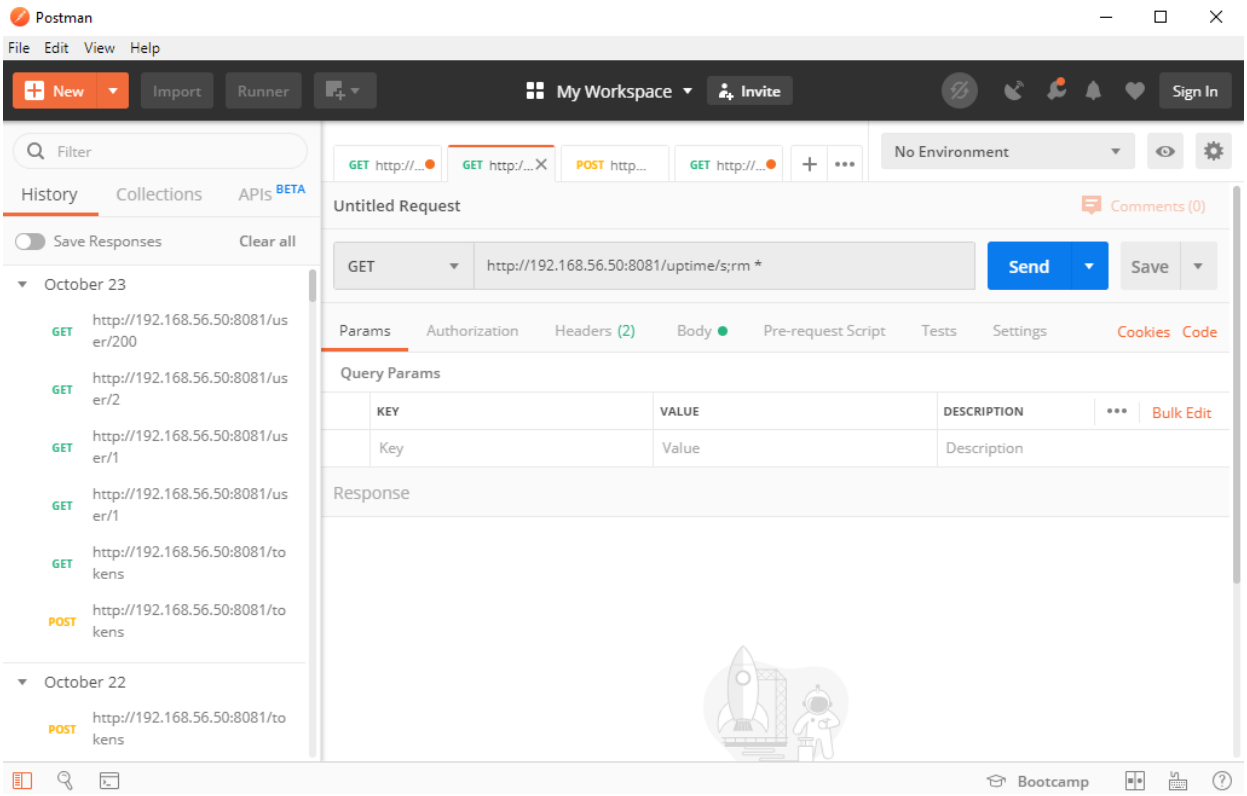
1. **Slanje zahtjeva**
   Postman se uobičajeno koristi za slanje zahtjeva REST API-jima, ali podržava slanje i SOAP i *GraphQL* zahtjeva.

2. **Automatizacija testiranja**
   Moguće je unaprijed pripremiti testove i zabilježiti odgovor API-ja na određeni zahtjev. Kad se dogodi neka izmjena programeri mogu jednostavno i brzo pokrenuti pohranjene testove i provjeriti podudaraju li se izlazi s očekivanima (tj. s izlazima kakvi su bili prije izmjene programskog kôda).

3. **Nadzor**
   U *Postmanu* je moguće definirati intervale u kojima će se periodički automatski poslati unaprijed pripremljeni zahtjevi API-ju i pratiti performanse i vrijeme potrebno za odgovor. Za ovu funkcionalnost potrebna je registracija.

4. **Dokumentiranje**
   U *Postmanu* se može jednostavno generirati dokumentacija za API koju i računala i ljudi lako čitaju.

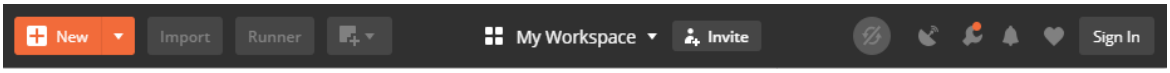
Kad se pokrene, *Postmanovo* korisničko sučelje izgleda kao što je prikazano na slici:



Slika 4 ***Korisničko sučelje alata Postman***

Tri osnovna gradivna bloka od kojih se sastoji korisničko sučelje su:

1. ***Header*** – u zaglavlju se nalaze postavke, opcije za stvaranje novih te učitavanje i pokretanje postojećih zahtjeva/kolekcija/okolina/dokumentacije itd. i još neke dodatne opcije poput prijave ili pregledavanja korisničkog profila.

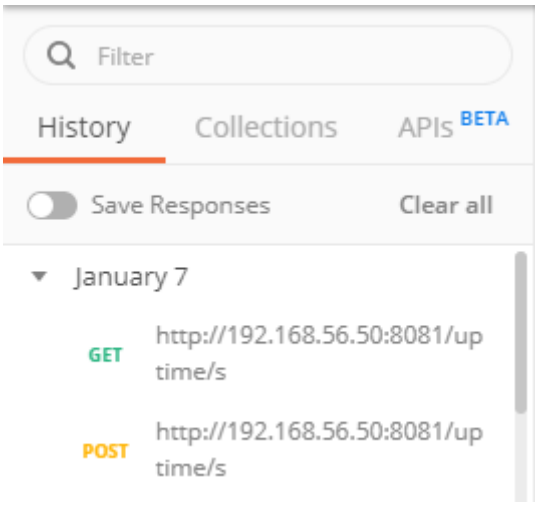


Slika 5 ***Zaglavlje (engl. header) alata Postman***

2. ***Side-bar*** – na *Side-baru* nalaze se dvije bitne opcije: povijest zahtjeva (engl. *History*) i kolekcije zahtjeva (engl. *Collections*). *Side-bar* se može ukloniti sa sučelja klikom na tipku '*View->Toggle Sidebar*'.

   Povijest zahtjeva sadrži kronološki (od najnovijih do najstarijih) poredane zahtjeve koji su se slali API-jima.

   Kolekcije smisleno grupiraju spremljene zahtjeve u logičke cjeline. Kolekciji se može dodijeliti ime, opis, a iznimno je korisno definirati autorizaciju, testove i varijable jer će se sve što je definirano na razini kolekcije automatski primijeniti na sve zahtjeve unutar nje (osim ako je na razini pojedinog zahtjeva definirano drukčije). Drugim riječima, dovoljno je definirati npr. autorizaciju na razini cijele kolekcije, umjesto da za svaki zahtjev posebno postavljamo autorizacijske tokene ili parametre. Kolekcije se mogu pokrenuti alatom *Runner*, podijeliti, pregledavati na *webu*, dodavati mape, kopirati, nadzirati (periodično pokretati kolekciju i pratiti rezultate i performanse)…



Slika 6 *Side-bar izbornik alata Postman*

3. ***Builder Window*** – Najveći dio korisničkog sučelja zauzima prozor za sastavljanje i slanje zahtjeva i analizu odgovora. Podrazumijevano (engl. *default*) će se dio prozora za zahtjev i dio za odgovor nalaziti jedan ispod drugog, ali klikom na tipku '*View->Toogle Two-Pan View*' mogu se postaviti jedan pored drugog.

3.1.1 Slanje zahtjeva REST API-jima

Osnova komunikacije na *webu* jest protokol HTTP koji se temelji na zahtjevima (koje klijent šalje poslužitelju) i odgovorima na zahtjev (koje poslužitelj šalje klijentu). REST je samo način kako će se koristiti protokol HTTP, i iz tog razloga potrebno je podsjetiti se dijelova HTTP poruka:

1. **Zaglavlje (engl. *Header*)**
   Svaka HTTP poruka obavezno sadrži zaglavlje koje omogućava klijentu i poslužitelju prosljeđivanje dodatnih informacija uz HTTP zahtjev ili odgovor. Osim standardnih (npr. *Content-Length, Content-Type, Accept, User-Agent, Allow*...), postoje i prilagođena HTTP zaglavlja koja programer sam definira za svoj API (poput *X-Auth-Token* kojeg ćemo sresti u primjerima). Tradicionalno su ovakva zaglavlja započinjala sufiksom '*X-*' kako bi klijent mogao raspoznati da je riječ o modificiranom zaglavlju, no konvencija o imenovanju je preporučila da se ipak ne koristi takvo imenovanje kako ne bi došlo do zabune jer su neki od modificiranih zaglavlja postali prihvaćeni kao standard. *Postman* je visoko prilagodljiv te podržava i standardna i modificirana zaglavlja.

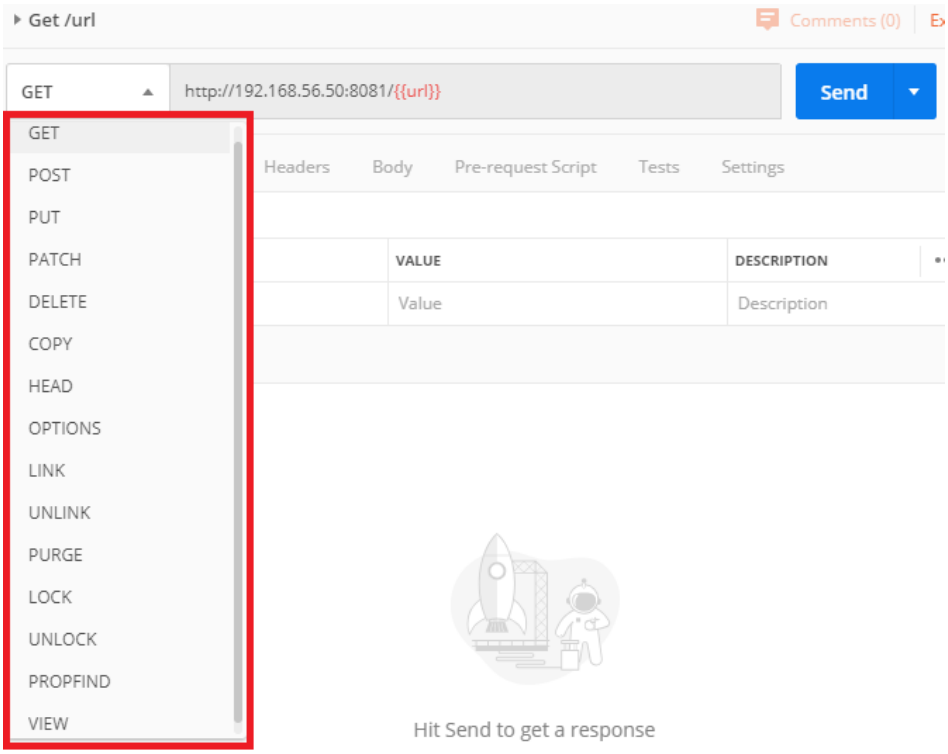
2. **Tijelo (engl. *Body*)**
   HTTP poruka može, ali i ne mora imati tijelo. U odgovoru, to je mjesto gdje se zahtijevani resurs šalje klijentu ili se šalje dodatni tekst s objašnjenjem ako je došlo do pogreške. Ako je riječ o tijelu zahtjeva, ono sadrži podatke koje klijent šalje ili datoteku koja će se učitati (engl. *upload*) na poslužitelj. Ako HTTP poruka sadrži tijelo, obično postoje linije u zaglavlju koje opisuju tijelo poruke, npr. *Content-Type* zaglavlje označava MIME tip podataka u tijelu kao što je *text/html* ili *image/gif*.

Važan dio HTTP zahtjeva je metoda, tj. HTTP glagol. Detaljniji opis standardnih metoda moguće je pronaći na poveznici, ali zasad ukratko recimo da su najčešće metode GET (za

dohvat resursa), POST (slanje podataka), PUT (dodavanje/ažuriranje resursa) i DELETE (brisanje resursa).

Osim standardnih metoda propisanih inačicom protokola HTTP 1.1, *Postman* podržava i još neke dodatne metode koje su dio proširenja *WebDAV*. Više o tim metodama može se pronaći na službenoj stranici, a jedan primjer je metoda COPY/MOVE za kopiranje/premještanje resursa s jednog URI-ja na drugi.

Metoda koja će se slati u zahtjevu se u *Postmanu* bira iz padajućeg izbornika kao što je prikazano na slici 7:



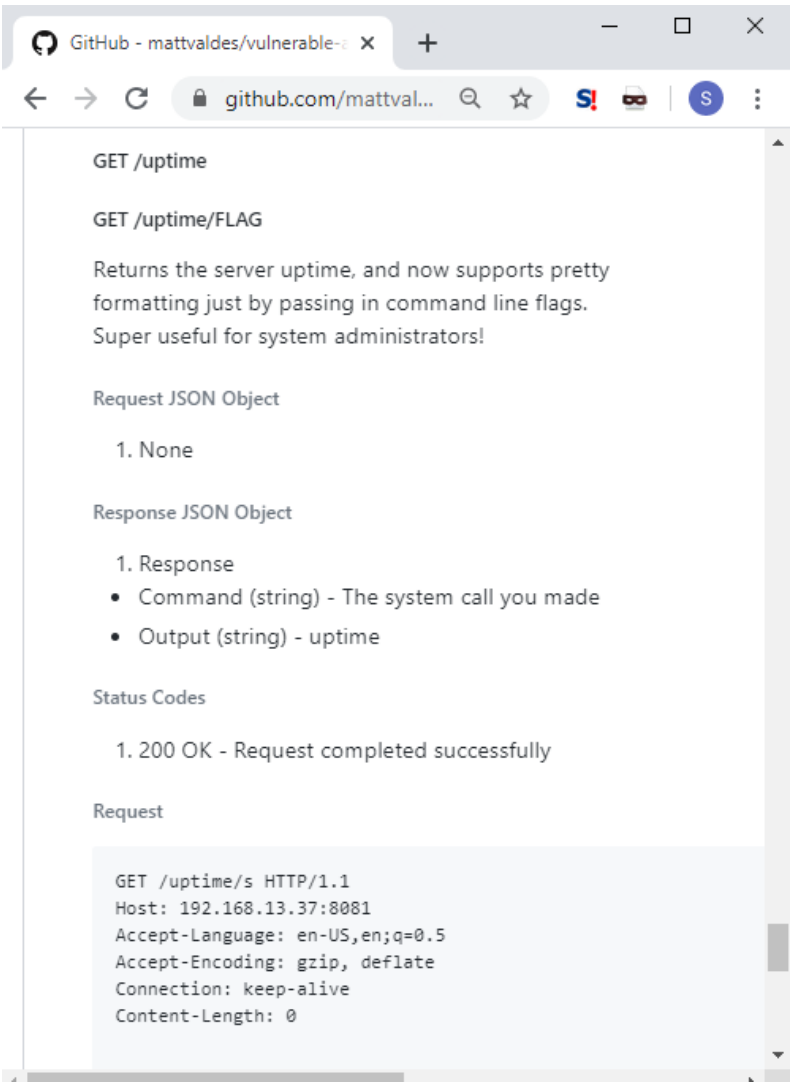
Slika 7 *HTTP metode koje podržava alat Postman*

Za primjere u dokumentu koristit će se projekt *Vulnerable API* čiji su izvorni kôd i dokumentacija dostupni u repozitoriju. *Vulnerable API* je razvijen kao primjer API-ja s nekim čestim ranjivostima na kojem se korisnici mogu upoznati s određenim propustima i pogreškama u API-jima.

Vulnerable API od korisnika traži autentifikaciju korisničkim imenom i lozinkom, nakon čega mu dodjeljuje (modificirani) autentifikacijski token. U bazi postoji više korisničkih zapisa, ali svaki korisnik smije pregledavati samo vlastite zapise slanjem zahtjeva `GET /user/USER_ID` i autentifikacijskog tokena koji mu je dodijeljen u polju zaglavlja *X-Auth-Token*. Ako je token ispravan i pripada korisniku čiji se zapis pokušava pregledati, korisniku će API odgovoriti zapisom.

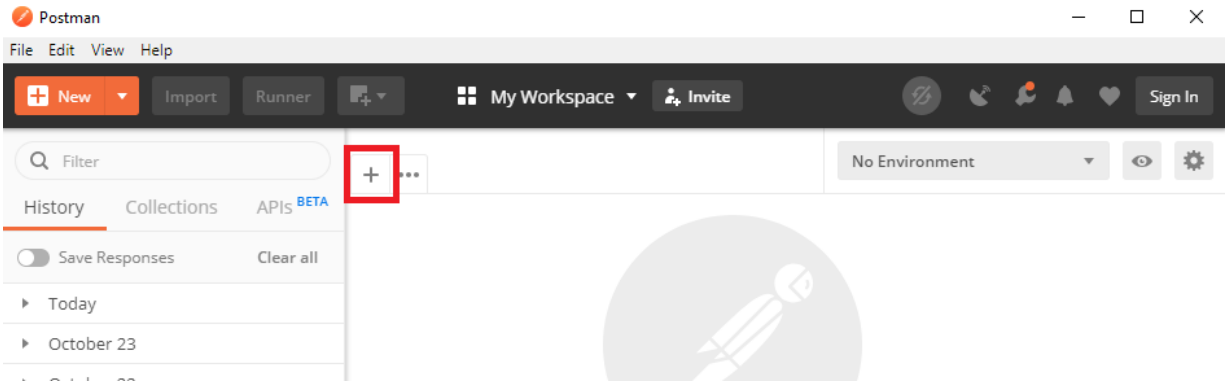
Kako bismo mogli sastaviti i poslati zahtjev, pogledajmo dokumentaciju *Vulnerable API*-ja u kojoj su navedene takozvane API krajnje točke (engl. *endpoints*), tj. URL-ovi na koje je potrebno poslati zahtjev kako bi se dobio odgovor. Na slici 8 prikazan je dio

dokumentacije koji se odnosi na izvršavanje naredbe *uptime* na poslužitelju na kojem je pohranjen API kojem šaljemo zahtjev.



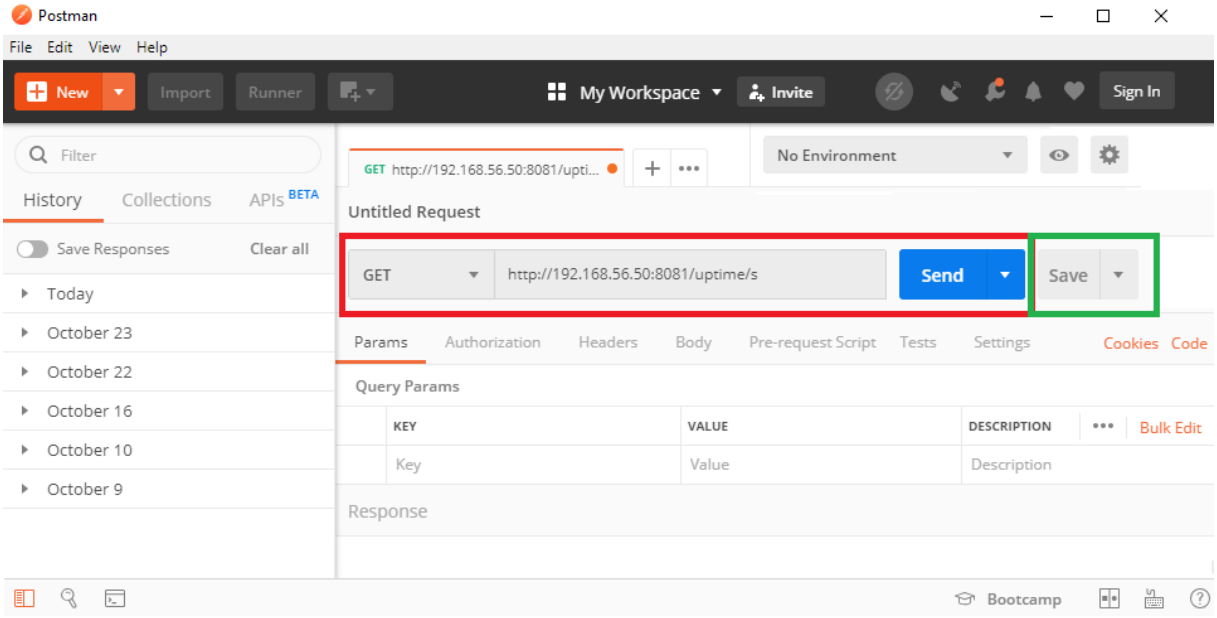
Slika 8 ***Isječak dokumentacije API-ja Vulnerable API***

Upućeni smo da moramo poslati zahtjev s metodom GET na *endpoint* `/uptime/FLAG` (pri čemu je *flag* zastavica koja definira kakvo formatiranje vremena treba prikazati korisniku). U *Postmanu* stvaramo novi zahtjev klikom na tipku '+' prikazanoj na slici 9, klikom na tipku '*File->New->Request*' ili tipku zaglavlja '*New->Request*'. Ako koristimo neki od dva posljednja načina, zahtjev moramo pohraniti u neku od postojećih kolekcija, tako da ćemo za sada, prije no što se upoznamo s kolekcijama, koristiti prvi način.



Slika 9 ***Stvaranje novog zahtjeva***

Nakon što odaberemo stvaranje novog zahtjeva, na zaslonu će se otvoriti *Builder Window* u koji upisujemo jednostavan GET zahtjev kao što smo pročitali u dokumentaciji. FLAG za formatiranje koji ćemo proslijediti API-ju neka za primjer bude '*s*' (čitanjem dokumentacije o naredbi `uptime` na operacijskim sustavima *Linux* saznat ćemo koje sve zastavice možemo postaviti).



Slika 10 ***Sastavljanje zahtjeva API-ju koji će nam prikazati vrijeme na poslužitelju***

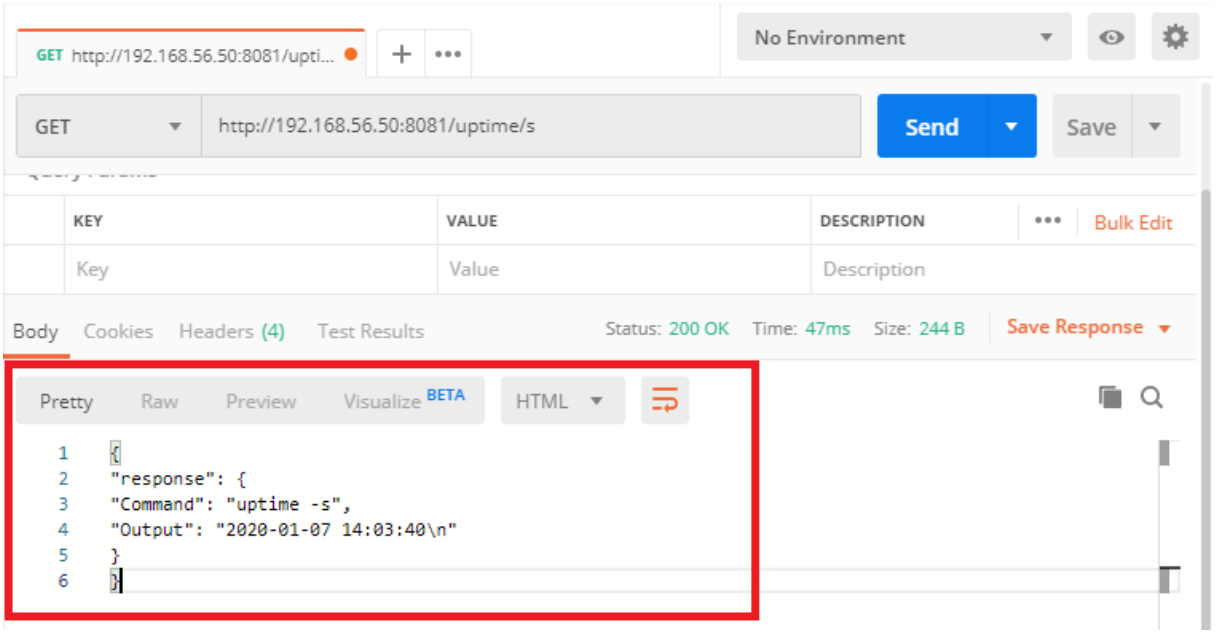
Zahtjev zatim možemo poslati API-ju klikom na tipku '*Send*' ili spremiti u neku od kolekcija klikom na tipku '*Save*'.

Kao što vidimo, ovo je vrlo jednostavan zahtjev za koji nije bilo potrebno postaviti autorizaciju, dodatne parametre, zaglavlja, tijelo, skripte ni testove koji će se izvršiti prije, odnosno nakon slanja zahtjeva.

Skripte koje se izvršavaju prije zahtjeva (engl. *Pre-request Scripts*) mogu se koristiti za postavljanje podataka ili varijabli u zahtjevu. Testovi se pokreću nakon što je primljen odgovor na poslani zahtjev API-ju, a testne skripte mogu raščlaniti detalje odgovora, kao što su npr. *Response Code*, *Cookies* i *Body* i pružiti informacije o rezultatima neuspjeha/uspjeha zahtjeva. Tijek izvođenja navedenih funkcionalnosti u *Postmanu* je:

Pre-Request Script → Request → Response → Test

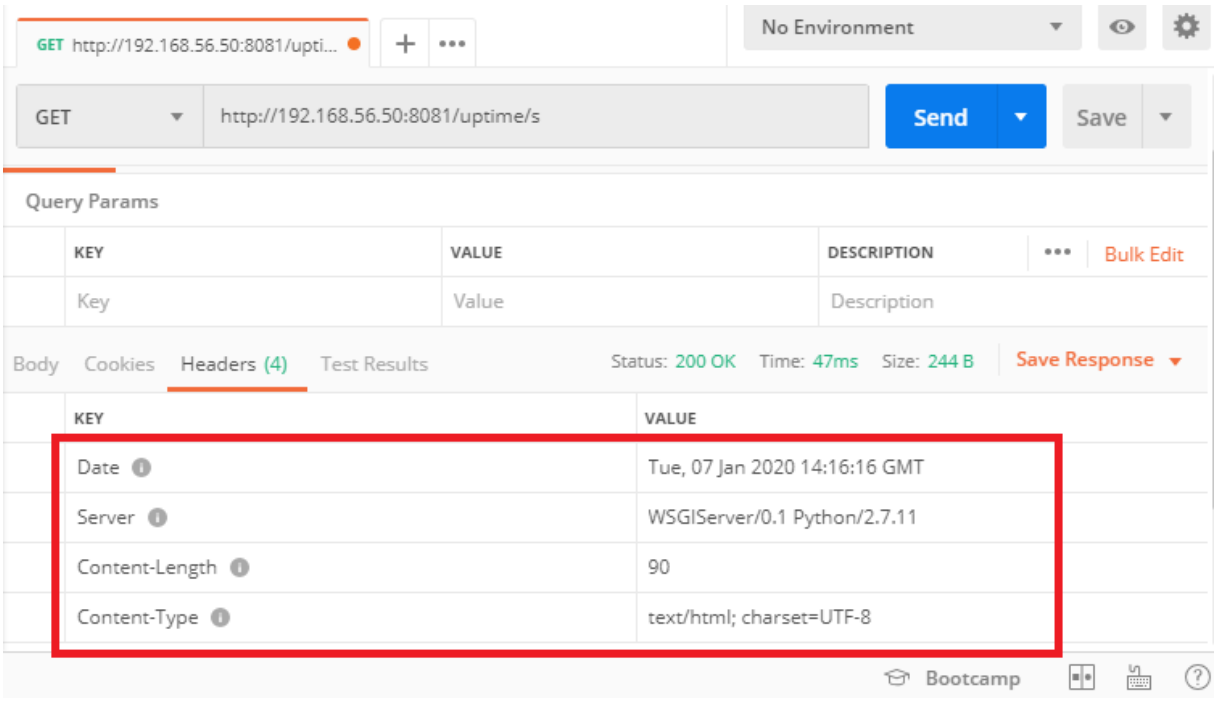
No, promotrimo zasad dobiveni odgovor na poslani zahtjev.



Slika 11 ***Dobiveni odgovor na poslani zahtjev***

Postoji nekoliko opcija formatiranja odgovora (*Pretty, Raw, Preview*) i padajući izbornik s različitim tipovima kodiranja/opisivanja podataka (JSON, HTML, XML...). Osim toga, dostupna nam je i metrika o odgovoru, poput HTTP statusnog kôda, vremena potrebnog za odgovor i veličine odgovora. Odgovor se može pretraživati klikom na ikonu povećala, a može se i spremiti klikom na tipku *'Save Response*'.

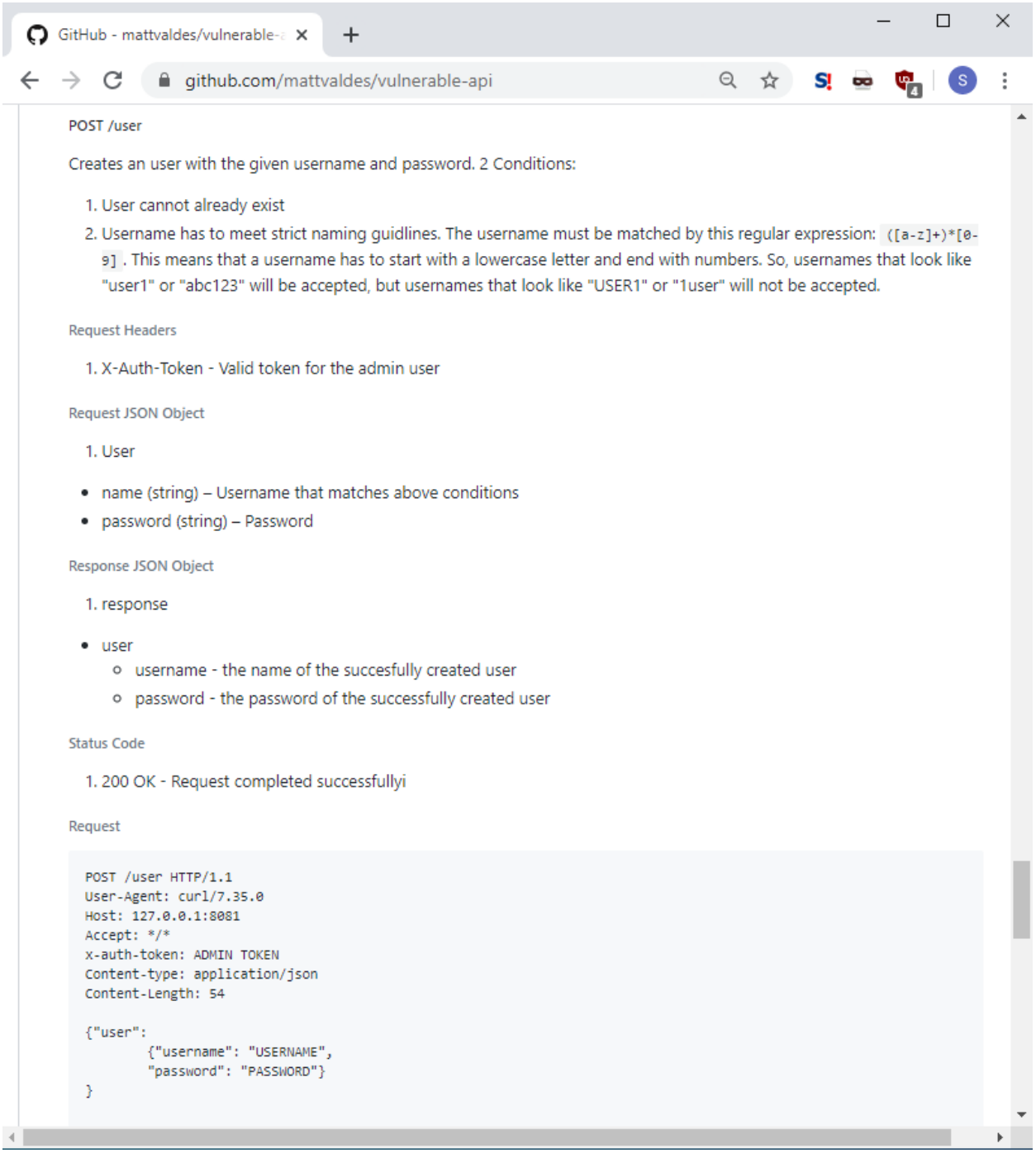
Osim odgovora, postoje kartice (engl. *tabs*) s nazivima *Body, Cookies, Headers i Test Results*. Kao što vidimo, te kartice pokazuju da je API, osim odgovora, vratio klijentu i četiri polja zaglavlja.



Slika 12 ***Zaglavlje koje je API poslao klijentu***

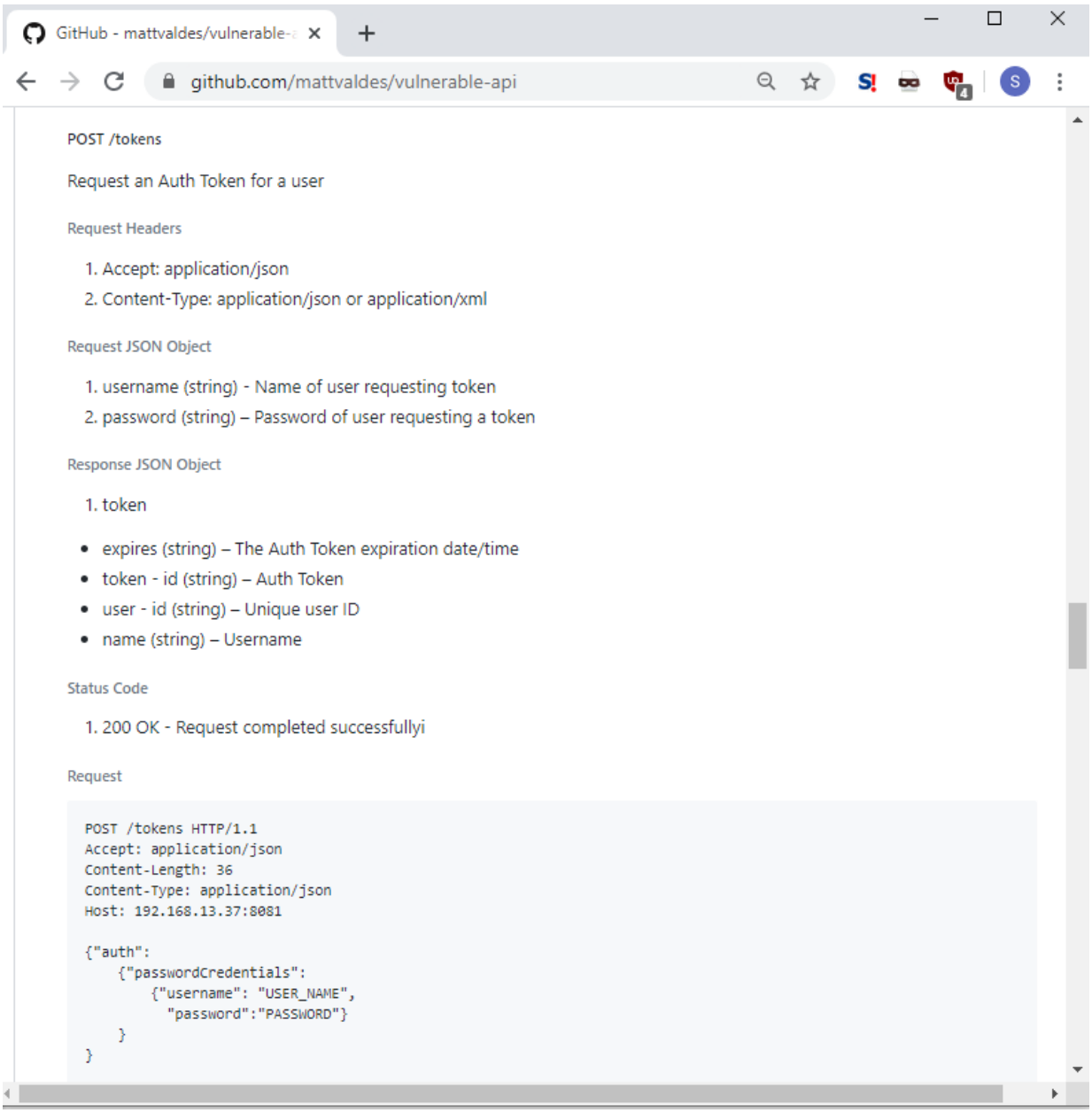
Ako bismo slali „složeniji zahtjev“, tj. zahtjev gdje administrator POST metodom želi dodati novog korisnika, morali bismo (prema uputama navedenim u dokumentaciji API-ja) postaviti dodatno i autorizacijska zaglavlja (kartica *Authorization*), i tijelo zahtjeva u kojem će se nalazit zapis kojeg želimo unijeti (kartica *Body*).

Pogledajmo dokumentaciju *Vulnerable API*-ja za takav zahtjev i upute kako sastaviti zahtjev da dobijemo željeni odgovor:



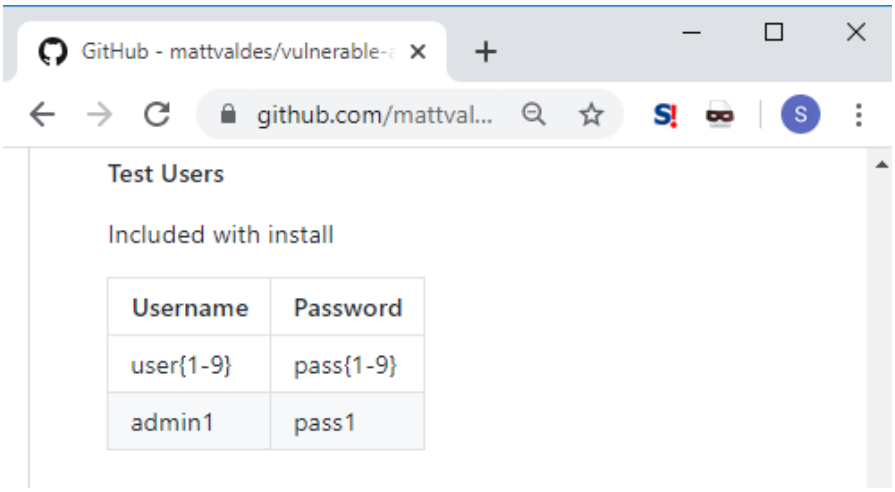
Slika 13 ***Dokumentacija za dodavanje novog korisnika u Vulnerable API***

Navedeno je da za uspješno slanje zahtjeva moramo, između ostalog, poslati zaglavlje *X-Auth-Token* s valjanim autentifikacijskim tokenom za administratora. Kako bismo dobili token za to zaglavlje, prvo moramo poslati zahtjev za dohvat autentifikacijskog tokena na *endpoint* `/tokens`, kao što je vidljivo iz dokumentacije:



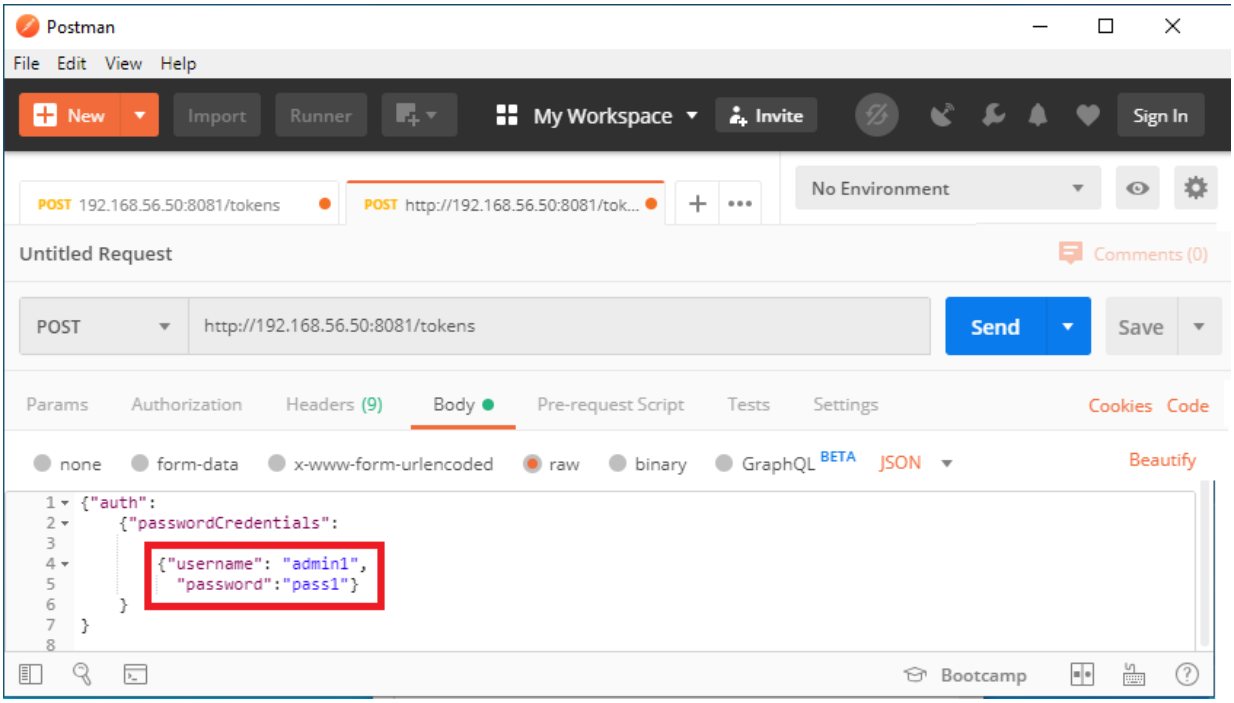
Slika 14 ***Dokumentacija za dohvat autentifikacijskog tokena***

Testne korisničke vjerodajnice kojima se možemo prijaviti u API također su navedene u dokumentaciji.



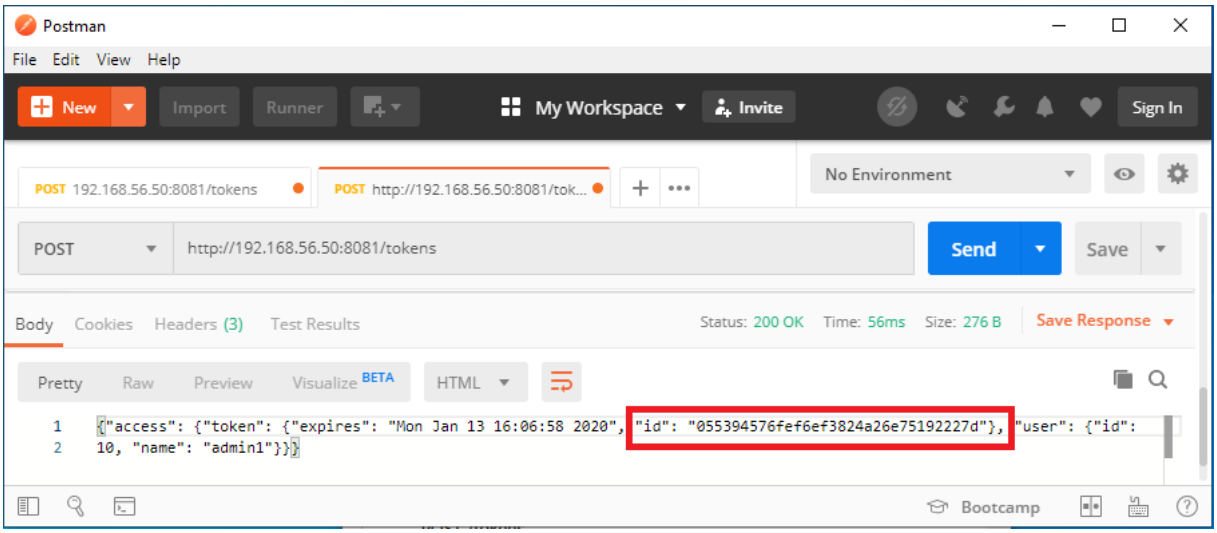
Slika 15 ***Vjerodajnice testnih korisnika za Vulnerable API***

Sastavljamo tijelo zahtjeva u *Postmanu*:



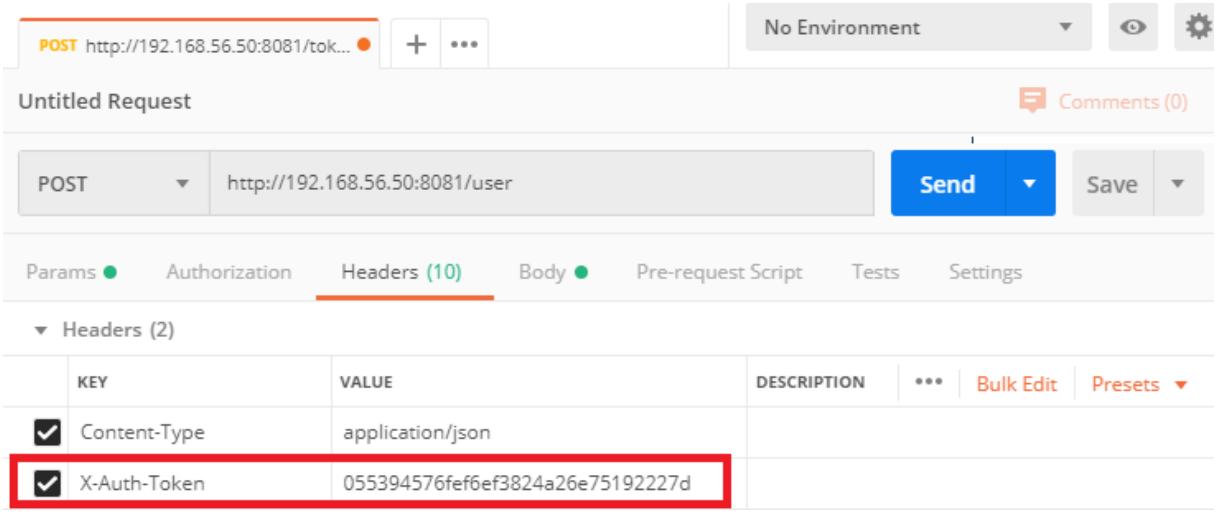
Slika 16 ***Sastavljanje zahtjeva koji će dodijeliti klijentu token ako su vjerodajnice ispravne***

Klikom na tipku '*Send*' šaljemo zahtjev API-ju i prikaže nam se dobiveni odgovor u kojem se nalazi i autentifikacijski token za administratora:



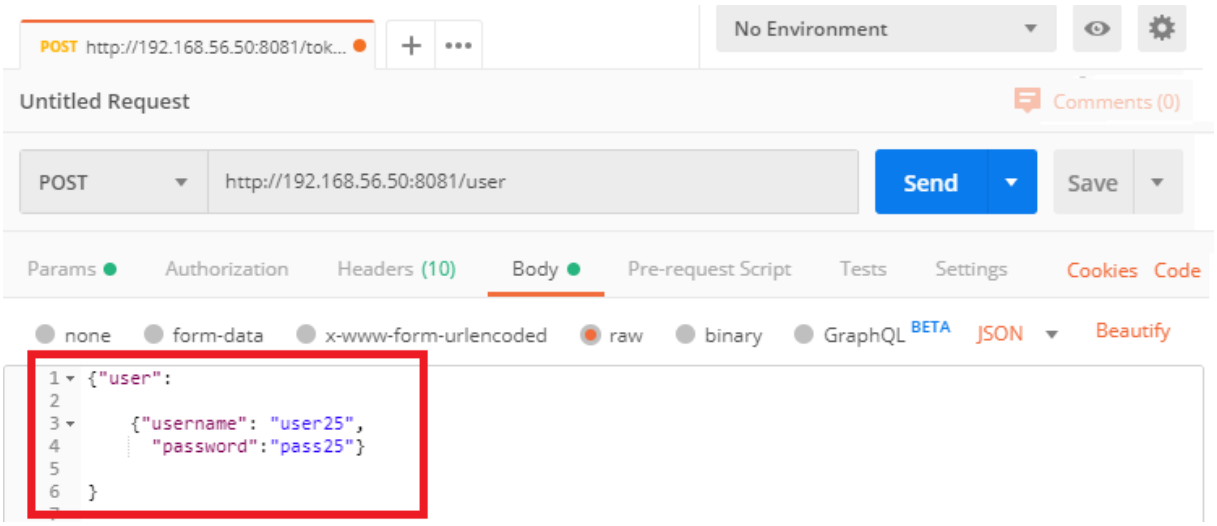
Slika 17 ***Odgovor na poslani zahtjev***

Sastavljamo novi zahtjev za *endpoint* `/user` kojim želimo dodati novog korisnika u bazu API-ja. Prepisujemo dobiveni autentifikacijski token u polje zaglavlja *X-Auth-Token* u kartici *Headers*.



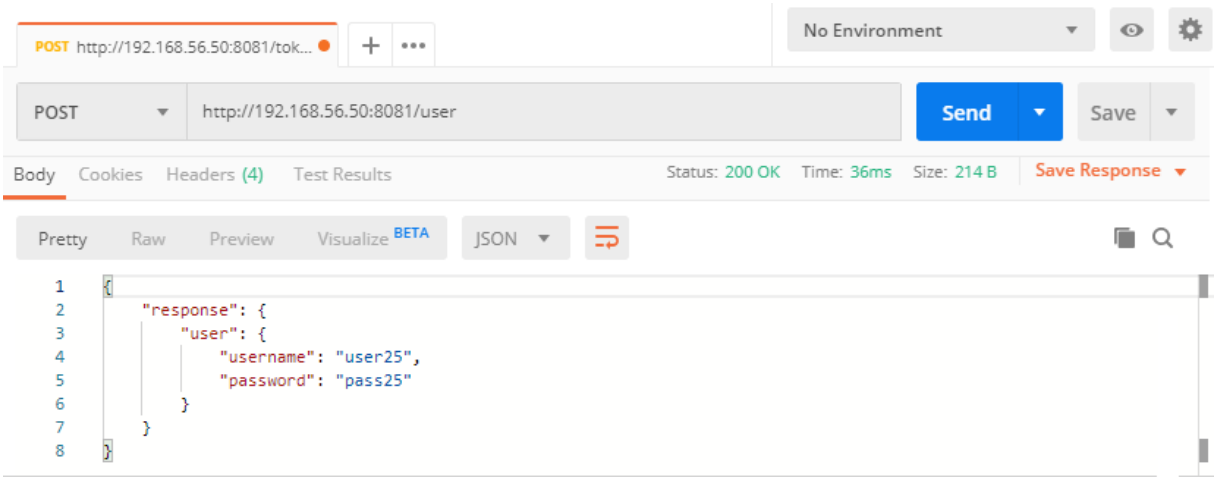
Slika 18 ***Postavljanje X-Auth-Token zaglavlja potrebnog za slanje zahtjeva***

Osim zaglavlja s autentifikacijskim tokenom, sastavljamo i tijelo zahtjeva u kojem se nalaze korisničko ime i lozinka novog korisnika kojeg želimo dodati u aplikaciju.



Slika 19 ***Sastavljanje tijela zahtjeva***

Klikom na tipku '*Send*' šaljemo zahtjev i primamo odgovor koji nam potvrđuje da je korisnik uspješno dodan.



Slika 20 ***Odgovor API-ja na poslani zahtjev***

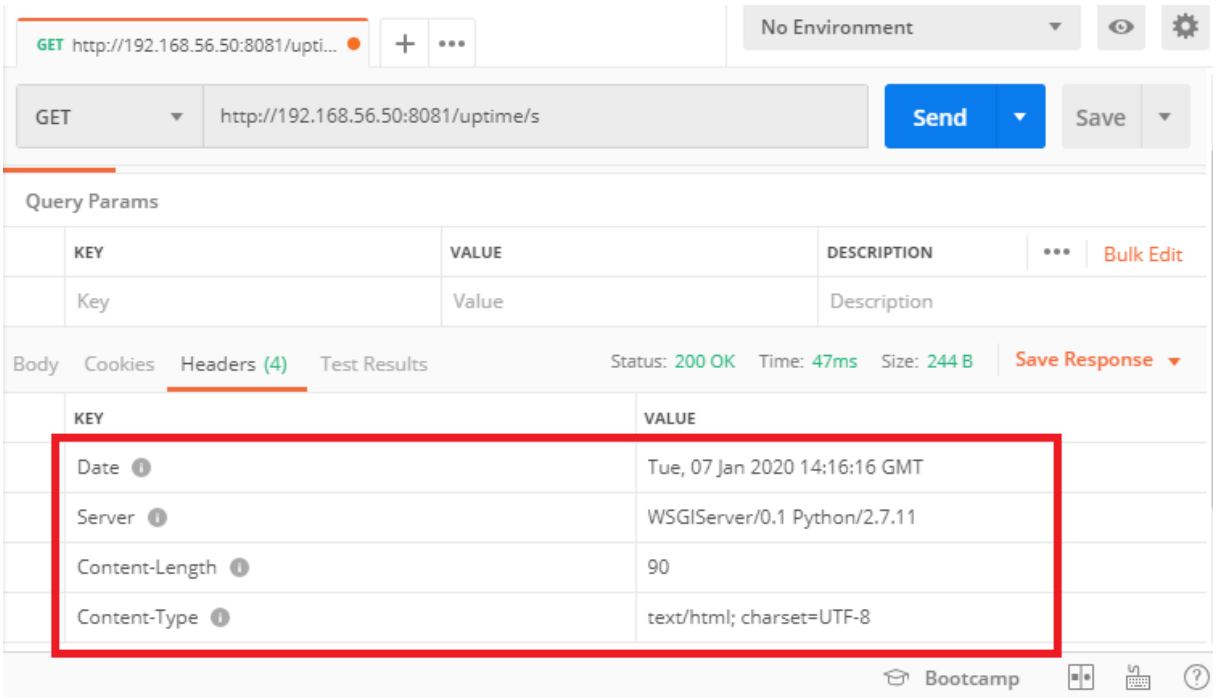
3.2 Sigurnosno testiranje korištenjem alata *Postman*

Kako bismo mogli testirati sigurnost API-ja alatom *Postman*, prvo se moramo upoznati s karakterističnim ranjivostima kako bismo zatim provjerili (tj. testirali) odnose li se takve ranjivosti i na naš API. Drugim riječima, kako bismo testirali ranjivosti, moramo znati što testirati i na koji način. Više o karakterističnim ranjivostima API-ja može se pronaći u već spomenutom dokumentu Nacionalnog CERT-a Sigurnost HTTP REST API-ja, a u nastavku ovog poglavlja pokazat ćemo jedan primjer sigurnosnog testiranja ranjivosti na izloženost osjetljivih informacija.

3.2.1 Izloženost osjetljivih informacija

Izloženost osjetljivih informacija je ranjivost koja napadaču omogućava dohvat osjetljivih informacija koje može izravno koristiti ili kojima će lakše napasti stranicu ili njene korisnike. Prisjetimo se slanja jednostavnog zahtjeva GET `uptime/s` iz prethodnog

poglavlja i odgovora kojeg nam je API vratio. Osim odgovora, API je vratio i zaglavlja od kojih je jedno *Server*, a sadrži informaciju o inačici poslužiteljskog softvera.



Slika 21 ***API nam je vratio informaciju o inačici poslužiteljskog softvera na kojem se nalazi***

Sad napadač zna o kojoj je inačici riječ i može pretraživati na *webu* postoji li koja ranjivost i spreman *exploit* za tu inačicu poslužiteljskog softvera.

Kad bi programer konfigurirao poslužitelj na način da ne prikazuje inačicu, ponovno slanje ovakvog zahtjeva ne bi odalo osjetljive informacije.

Još jedan primjer odavanja osjetljivih informacija bio bi slanje zahtjeva za resursom koji ne postoji, pri čemu bi se napadaču prikazala *web* stranica *404 Not Found* koja je često karakteristična za određeni poslužiteljski softver.

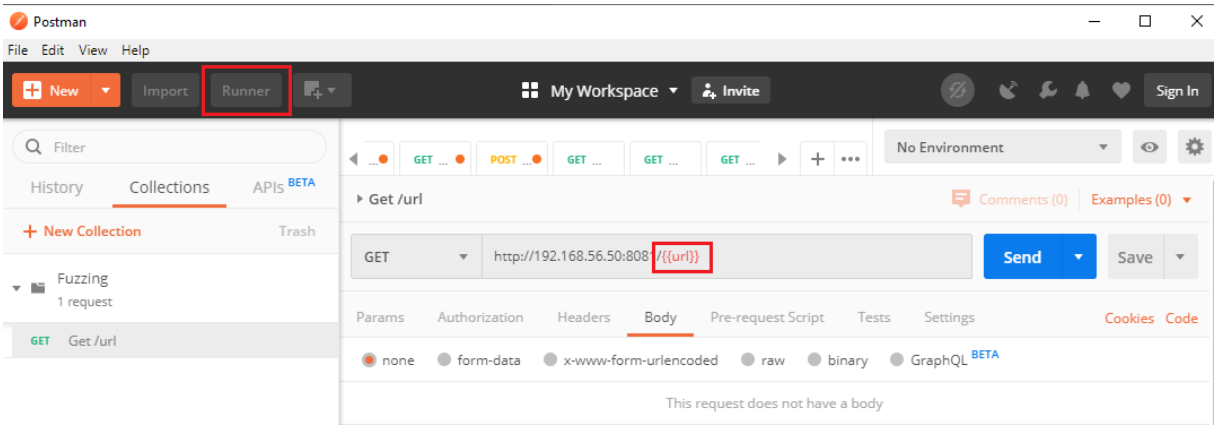
Zanimljivo je i slanje zahtjeva s metodama koje su dio *WebDav* proširenja HTTP protokola. Neki popularni PHP programski okviri čudno ili različito reagiraju na metode koje nisu dio standardnih metoda. Recimo *Laravel* PHP programski okvir koji je izuzetno popularan, za metodu PROPFIND diže *exception*, odnosno vraća *error* 500, dok kod nekih drugih to nije slučaj, čime ih je moguće otprilike identificirati.

3.2.2 *Fuzzanje* parametara

Fuzzanje parametara korisno je za provjeravanje kako aplikacija reagira na neočekivani unos, primjerice na razna umetanja (engl. *injection*) koja se mogu dogoditi gotovo svugdje gdje se prihvaća korisnički unos i općenito predstavljaju (prema istraživanju organizacije OWASP) najrizičniju ranjivost u *web* aplikacijama.

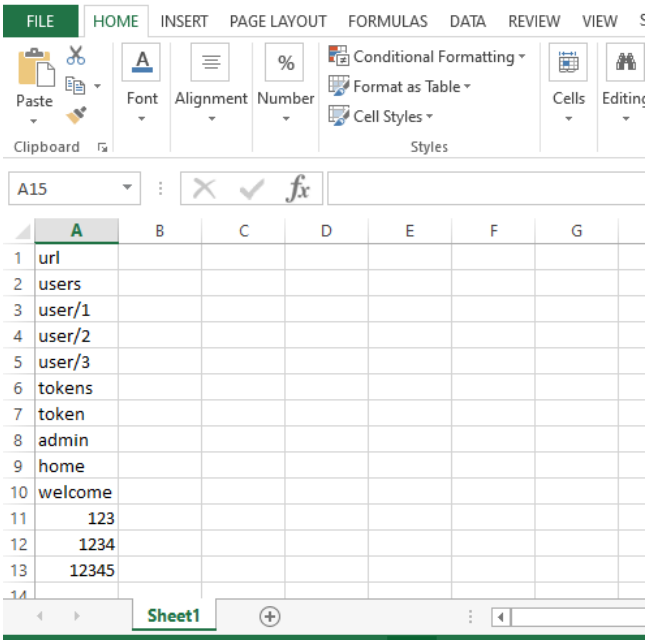
Pojam „*Fuzzanje* parametara“ odnosi se na automatsko isprobavanje različitih unaprijed definiranih ili nasumičnih vrijednosti određenog parametra.

Kako bi se moglo koristiti *fuzzanje*, potrebno je stvoriti novu kolekciju zahtjeva i unutar nje definirati zahtjev s varijablom koja se želi *fuzzati*. Varijable se u *Postmanu* označavaju dvostrukim vitičastim zagradama, npr. **{{url}}** kao što je prikazano na slici 2215.



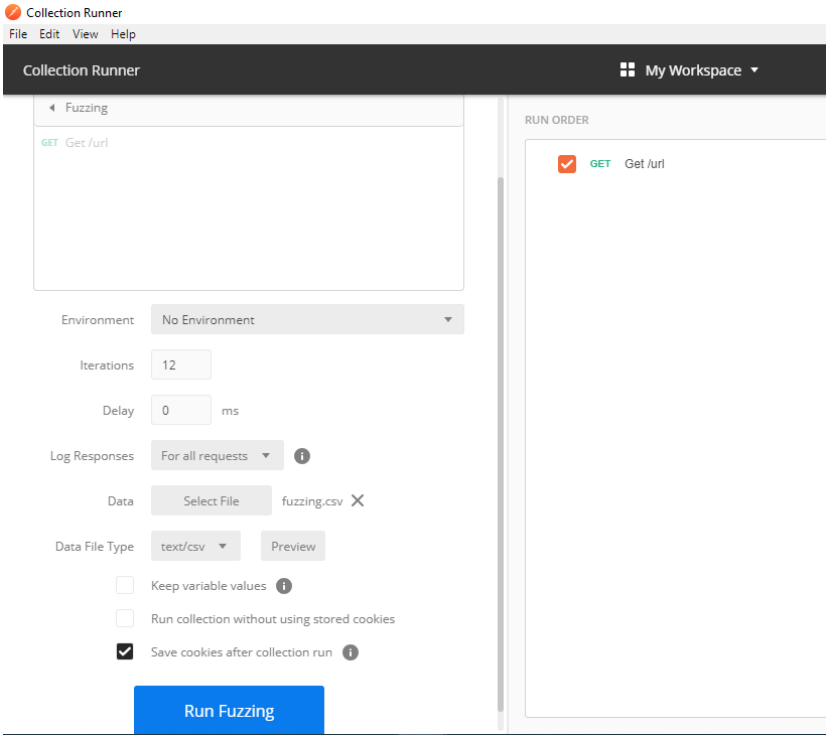
Slika 22 ***Varijabla {{url}} na koju će se primijeniti fuzzanje***

Ta se varijabla ne mora isključivo nalaziti u URL-u kao što je navedeno u primjeru, već se može nalaziti i u zaglavlju ili tijelu zahtjeva. Nakon toga se stvara datoteka (ili preuzme neka već postojeća koju je netko drugi pripremio i javno objavio) i upisuju različite vrijednosti koje će se izmjenjivati u parametru. U ovom primjeru stvorena je .csv datoteka s nekoliko kombinacija. U prvom retku datoteke mora se nalaziti naziv varijable, u ovom slučaju ***url***.



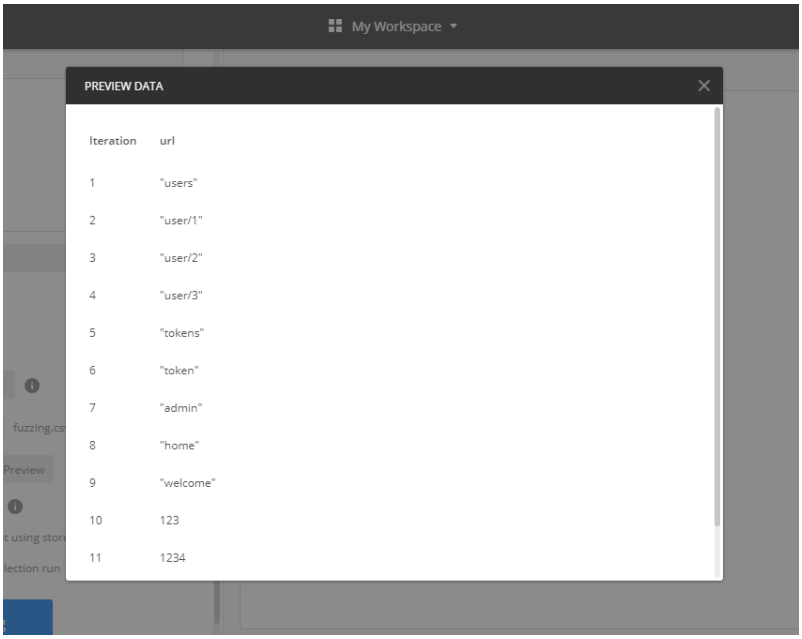
Slika 23 ***Definiranje vrijednosti koje će se isprobati u .cvs datoteci***

Nakon definiranja vrijednosti koje će se isprobavati u parametru (a to ne moraju biti samo riječi, već može biti i neki pokušaj umetanja SQL ili XSS kôda), pokreće se alat *Collection Runner*.



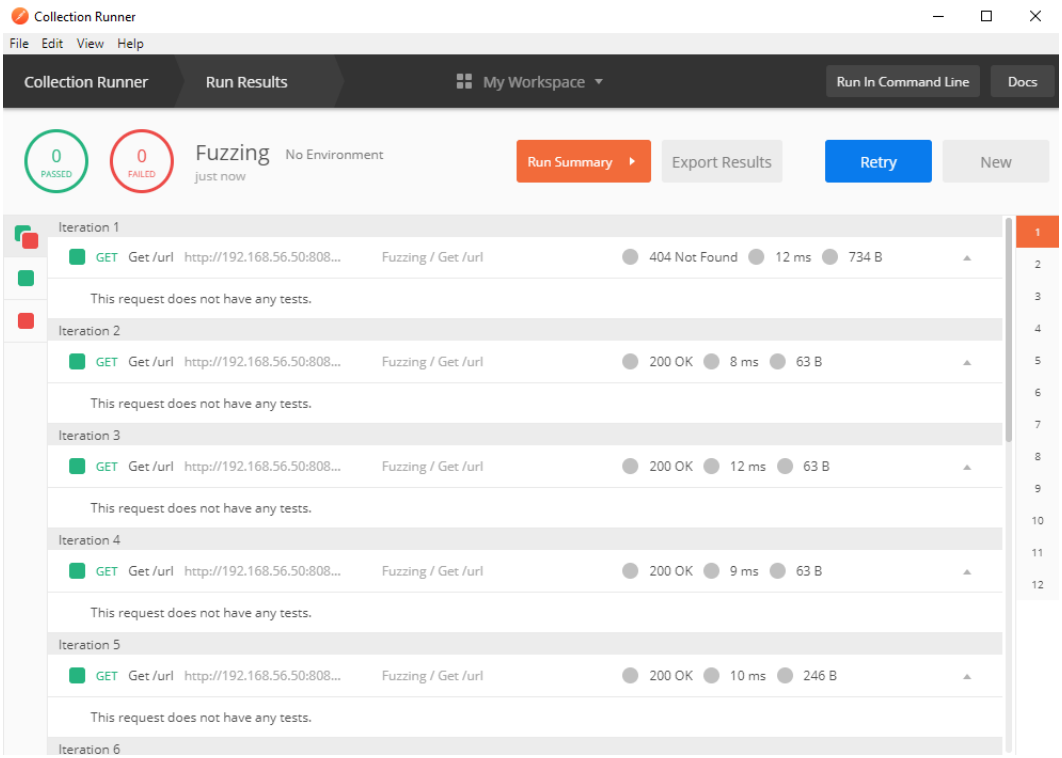
Slika 24 ***Postavke za alat Collection Runner***

Učitamo definiranu datoteku i označimo zahtjeve na koje želimo primijeniti *fuzzanje*. Klikom na tipku *'Preview'* provjeravamo je li se datoteka ispravno učitala.



Slika 25 ***Pretpregled vrijednosti koje će se isprobati u parametru {{url}}***

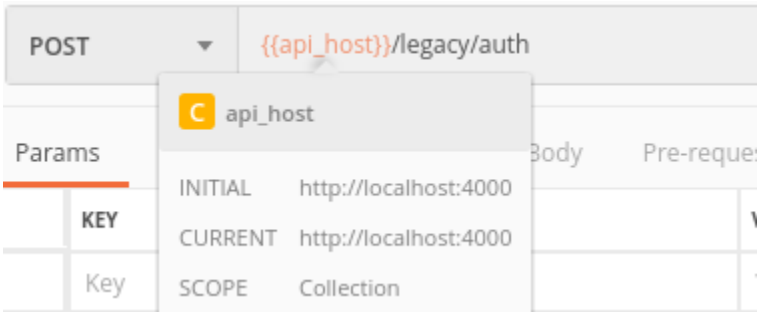
Pokrećemo *fuzzanje* klikom na tipku *'Run Fuzzing'* i u rezultatima vidimo kako je aplikacija odgovorila na zahtjeve.



Slika 26 ***Rezultati slanja zahtjeva na različite URL-ove***

Ova mogućnost korisna je kod pogađanja URL-ova (tj. *endpointa*) koji nisu dokumentirani (tj. tajni su jer su vjerojatno namijenjeni isključivo administratoru), a možda je moguće pristupiti nekim resursima preko njih bez potrebne autentifikacije. *Fuzzanje* je također korisno i kod testiranja ranjivosti na umetanje kôda, *brute-forcea* korisničkih vjerodajnica (tad bi se varijabla koja se *„fuzza"* nalazila u tijelu zahtjeva) itd.

Na primjeru *fuzzanja* vidjeli smo i da se u *Postmanu* mogu koristiti varijable. Osim za *fuzzanje*, varijable imaju još niz korisnih primjena (kao i kod programiranja) i u *Postmanu* postoji nekoliko vrsta varijabli. Jedan koristan primjer varijable je kod slanja zahtjeva na testnu i na produkcijsku okolinu. To su različiti poslužitelji s različitom IP adresom/domenom. Umjesto da svim zahtjevima koji su slani na testnu okolinu redom mijenjamo IP adresu na adresu produkcijskog poslužitelja, možemo promijeniti vrijednost varijable URL na novu IP adresu i svi će se zahtjevi poslati identično kao prije, ali na novu IP adresu.

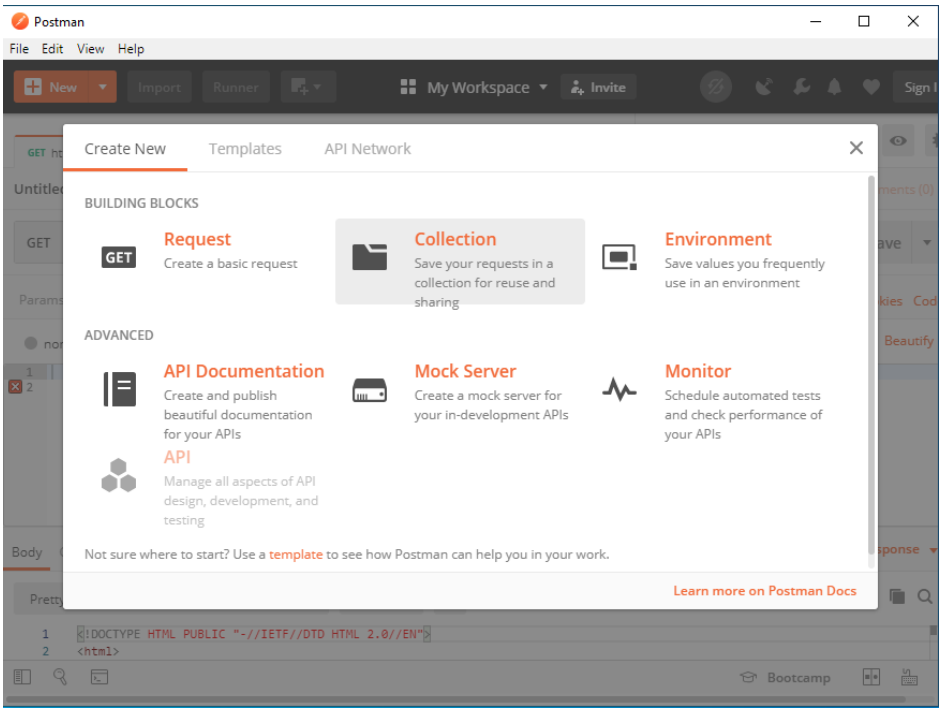


Slika 27 ***Umjesto adrese hosta upisujemo naziv varijable***

3.3 Automatizacija (sigurnosnog) testiranja alatom *Postman*

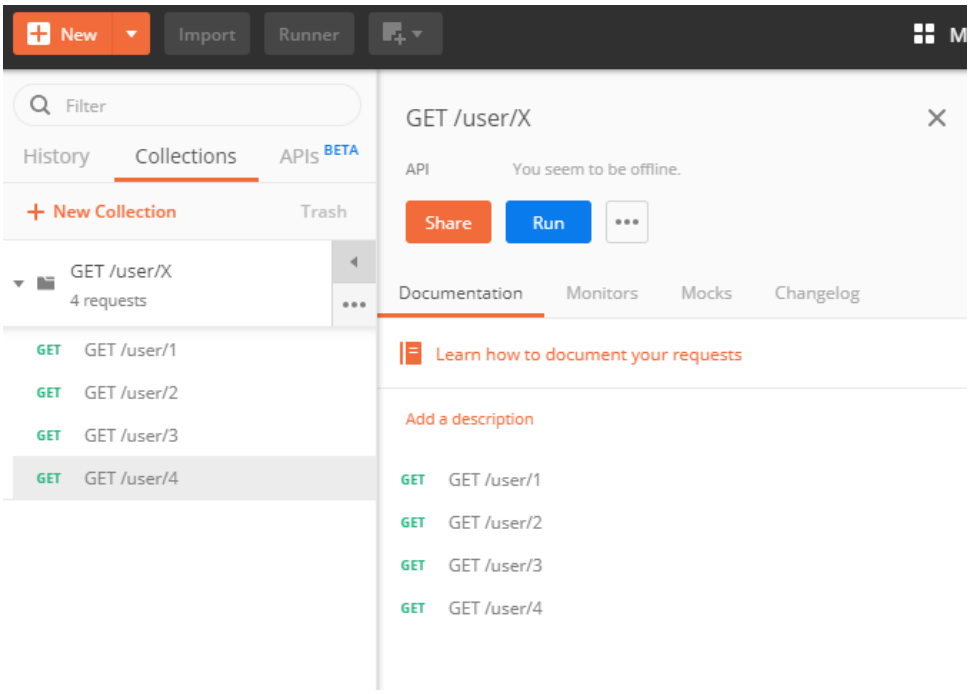
Za automatizaciju testiranja (a to se odnosi i na sigurnosno testiranje) alatom *Postman* moramo koristiti kolekcije (engl. *Collections*). Ponovimo, kolekcije su zbirke zahtjeva koji su grupirani u logičke cjeline. Pokretanjem kolekcije (za što se koristi alat *Runner* unutar *Postmana*) izvršit će se svi zahtjevi koji se nalaze u kolekciji, jedan za drugim.

Za demonstraciju ove funkcionalnosti stvorit ćemo kolekciju naziva *GET /user/X* koja će pohranjivati nekoliko zahtjeva koji dohvaćaju zapis za određenog korisnika ovisno o parametru X koji iterira od 1 do 4.



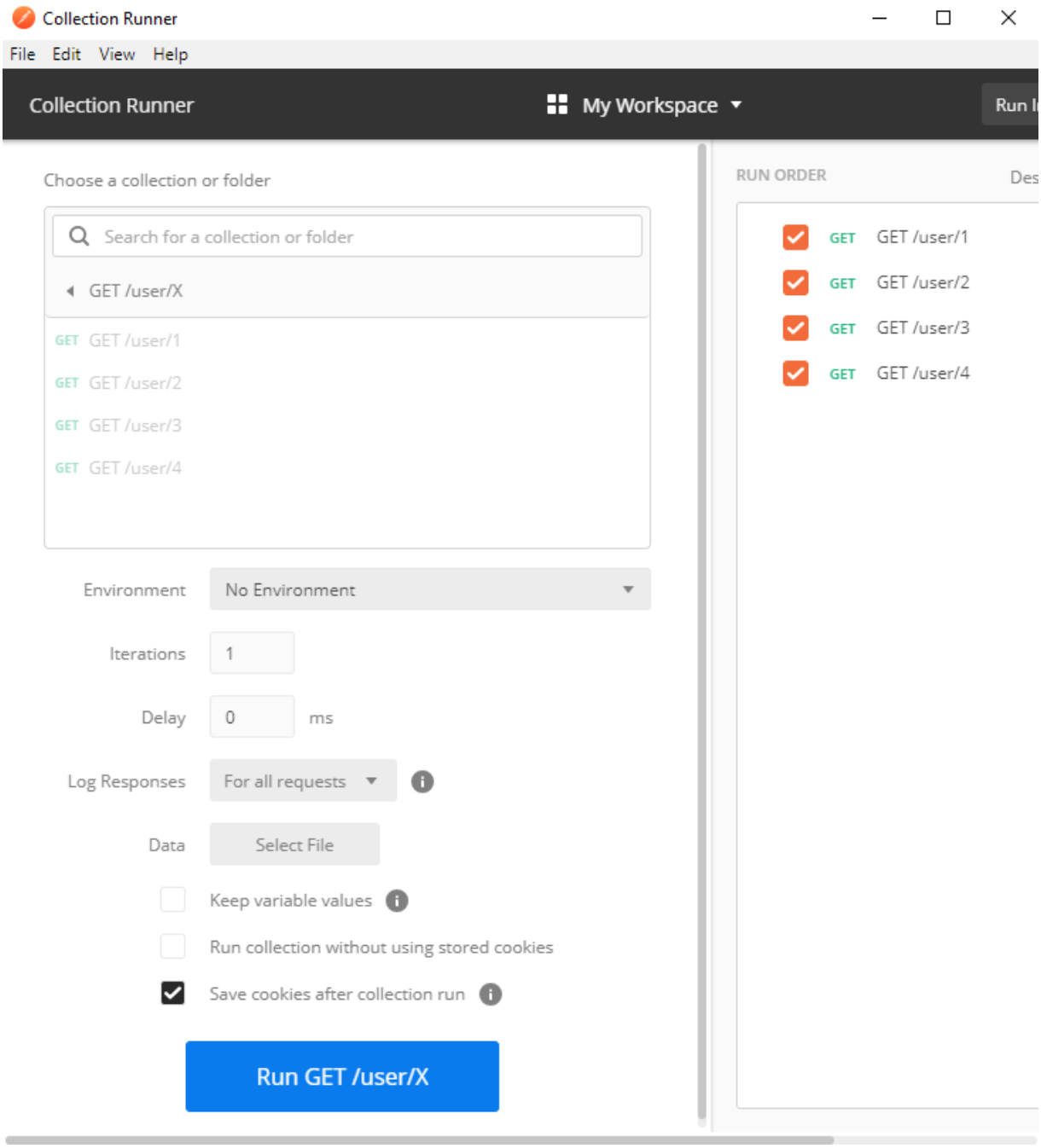
Slika 28 *Stvaranje nove kolekcije*

Stvorimo nekoliko zahtjeva i spremimo ih (vrlo je bitno kliknuti tipku *'Save'* kako bi se zahtjev spremio u kolekciju).



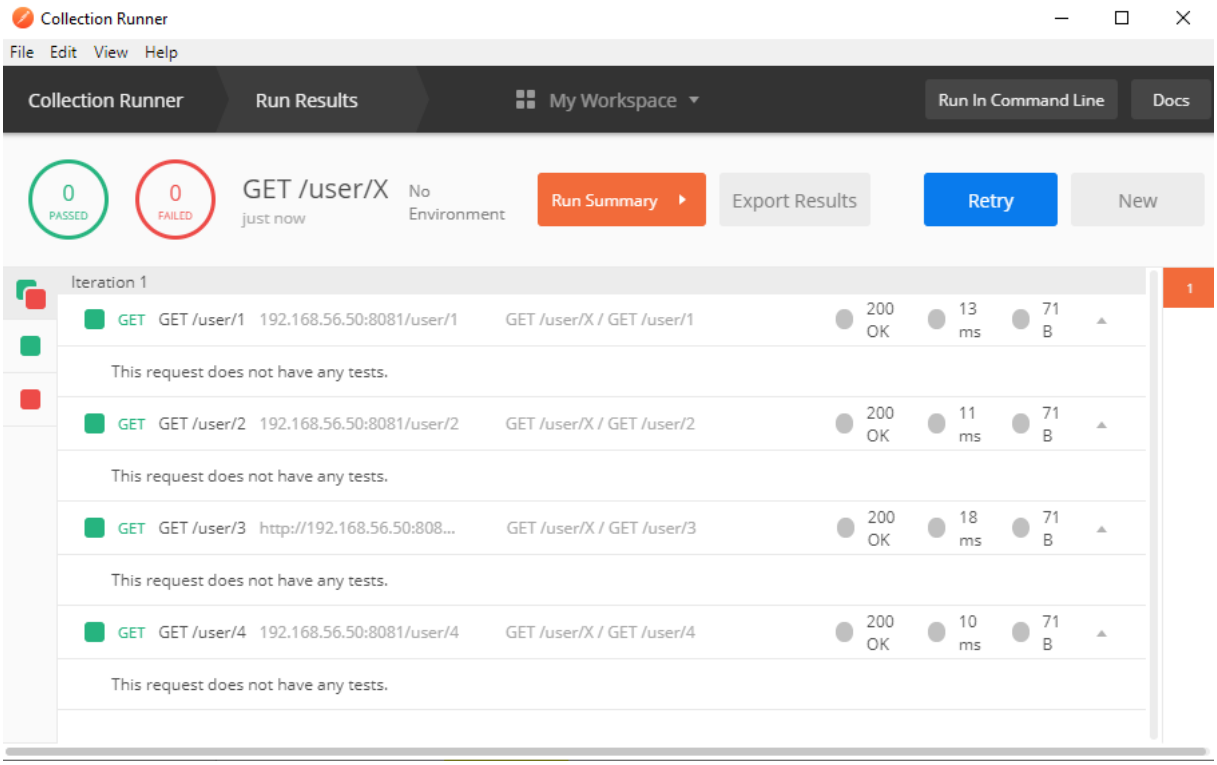
Slika 29 ***U kolekciji GET /user/X sad se nalazi nekoliko zahtjeva***

Pokrenimo kolekciju (tj. izvršimo sve zahtjeve koji se u njoj nalaze) klikom na tipku '*Run*'. Pokrenut će se alat *Collection Runner* u kojem je moguće konfigurirati dodatne postavke (broj iteracija, okruženje, varijable…).



Slika 30 ***Konfiguracija postavki alata Collection Runner***

Ostavimo podrazumijevane postavke i kliknimo na tipku '*Run GET /user/X*'.

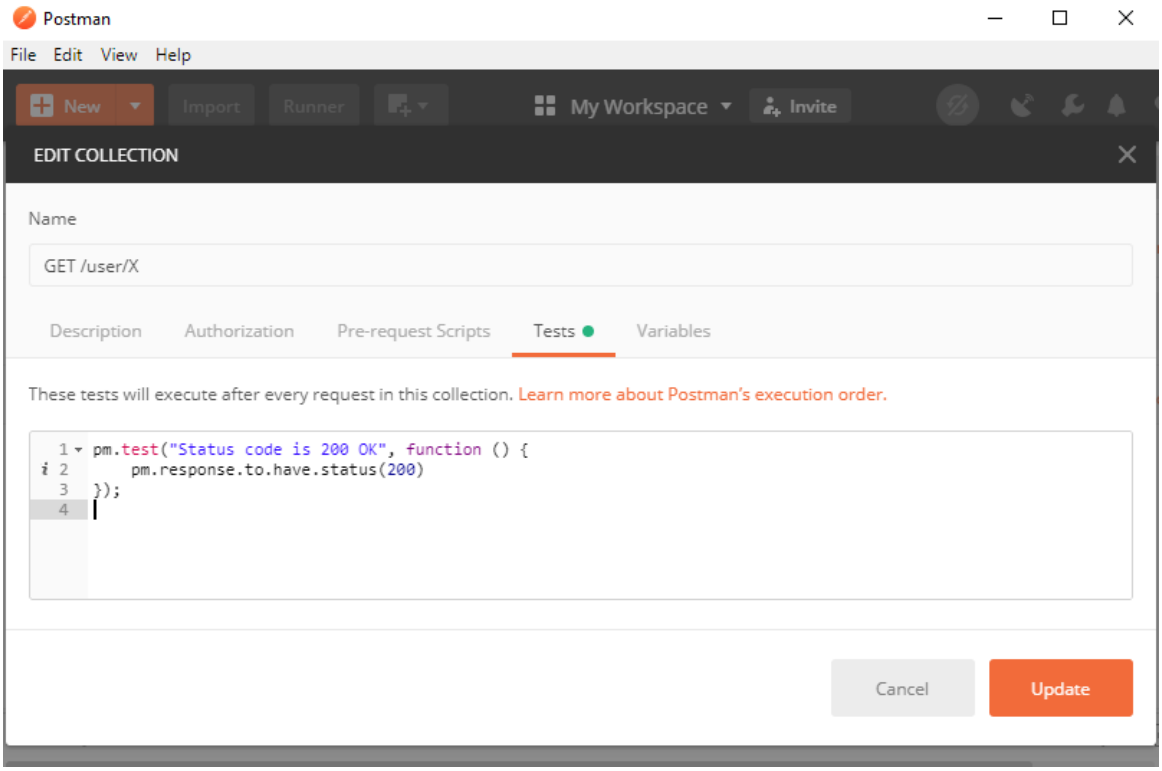


Slika 31 ***Rezultati izvršavanja kolekcije, tj. svih zahtjeva koji su u njoj pohranjeni***

Osim što možemo pokretati automatizirane testove i vidjeti rezultate njihovog izvršavanja (npr. za svaki zahtjev iz kolekcije vidimo statusni kôd odgovora, vrijeme potrebno za odgovor i veličinu odgovora), možemo usporediti dobiveni odgovor s očekivanim ili prethodnim.

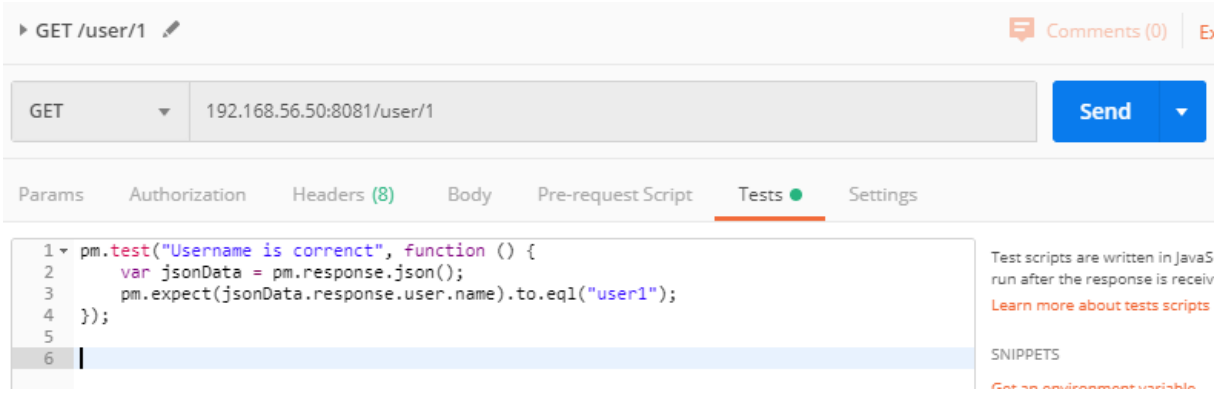
Uzmimo za primjer trivijalni scenarij gdje znamo korisnička imena koja se nalaze na određenom *endpointu* i želimo napisati testove koji će provjeravati je li zahtjev uspješno poslan i nalazi li se ispravno korisničko ime na određenoj lokaciji ili je došlo do nekih promjena.

Budući da se ista programska logika primjenjuje za provjeru je li zahtjev uspješno poslan, testnu skriptu za provjeru uspješnosti slanja zahtjeva možemo definirati na razini kolekcije, a ne pojedinačnog zahtjeva. Ta će se testna skripta onda primijeniti na sve zahtjeve pohranjene u kolekciji.



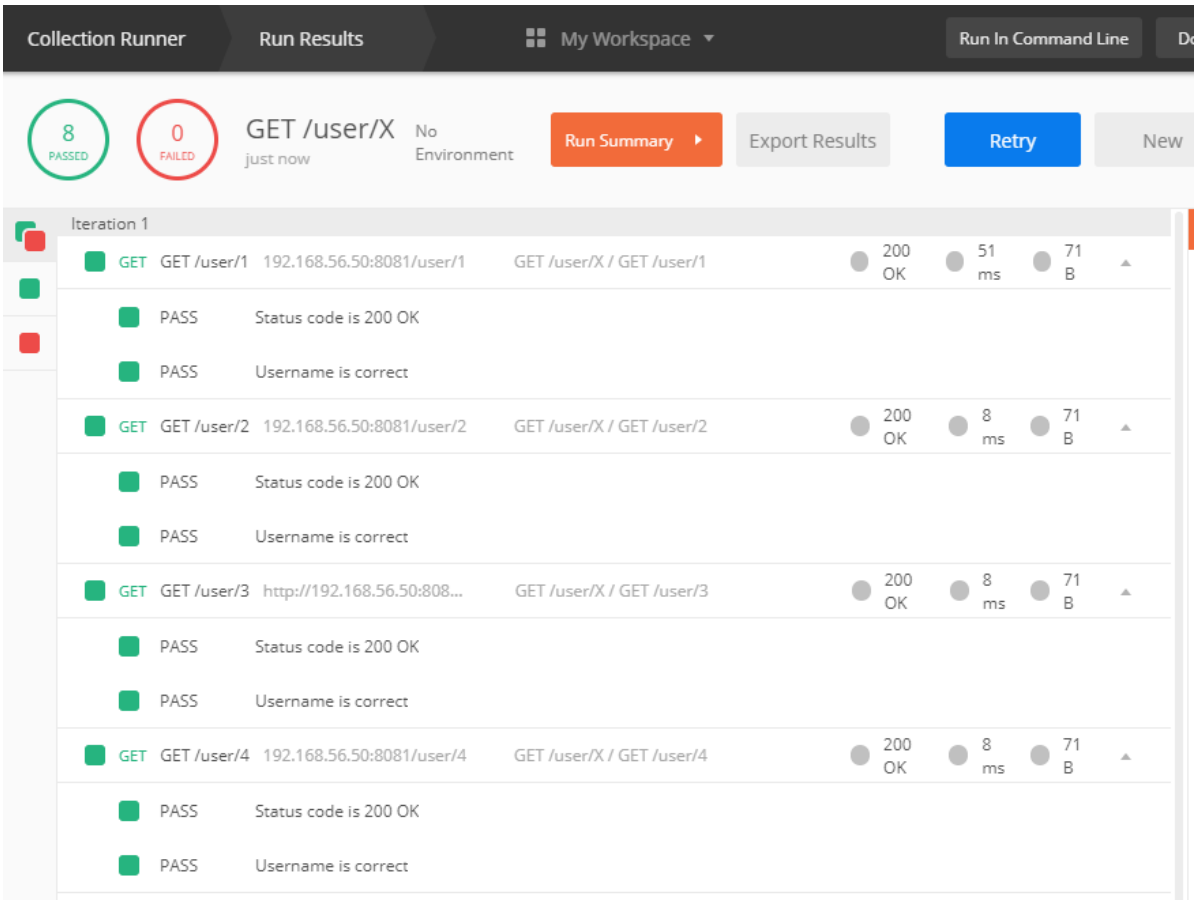
Slika 32 ***Testna skripta na razini kolekcije primijenit će se za sve zahtjeve u kolekciji***

Podatak koji provjeravamo (korisničko ime) nije isti na svakom *endpointu* pa ćemo napisati test za svaki zahtjev pojedinačno. Provjerimo nalazi li se na *endpointu* `/user/1` stvarno korisnik „user1“.



Slika 33 ***Skripta za svaki zahtjev pojedinačno provjerava ispravnost podataka na svakom endpointu***

Analogno napišimo skripte i za ostale zahtjeve (za korisnike od 1 do 4). Ponovno moramo provjeriti da smo spremili sve zahtjeve klikom na tipku '*Save*'. Pokrenimo izvršavanje kolekcije.

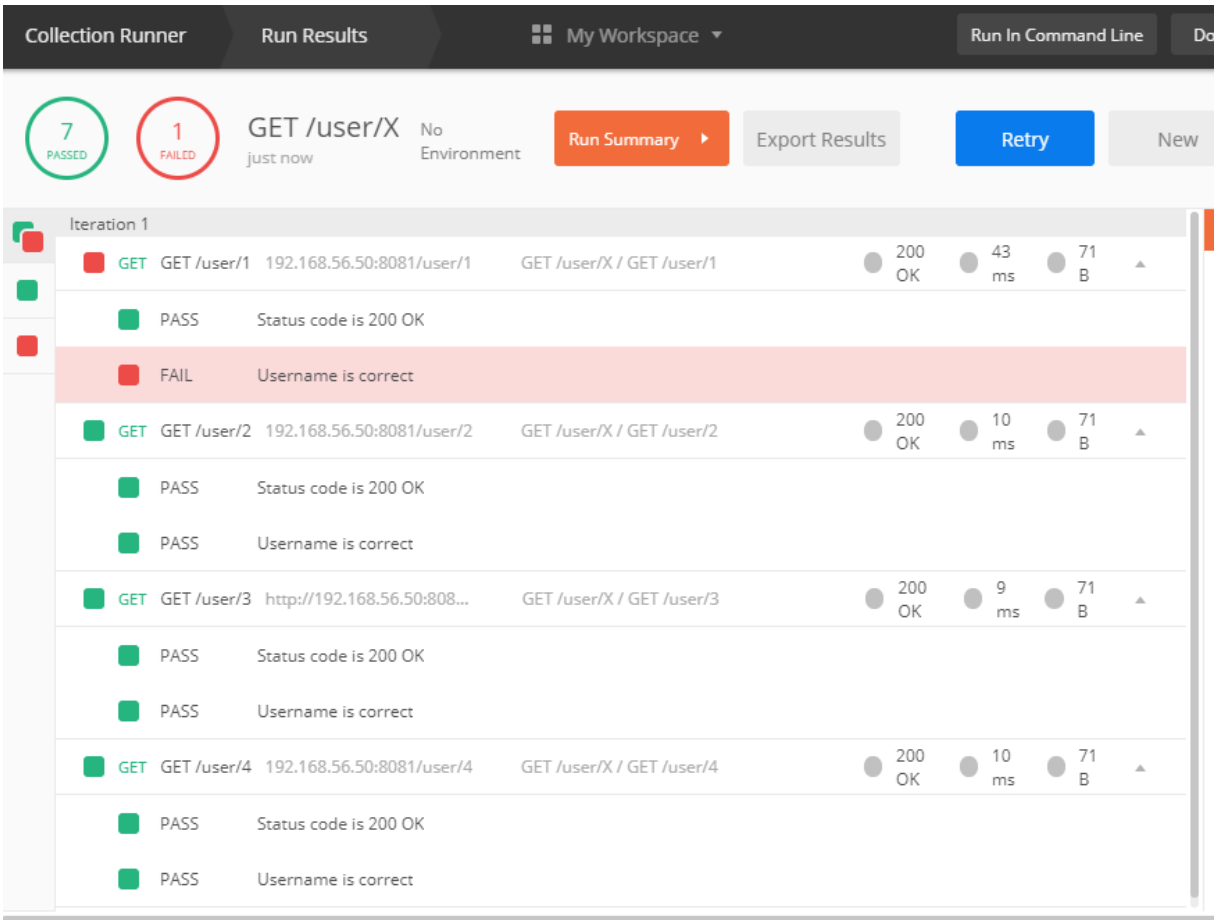


Slika 34 ***Rezultati izvršavanja kolekcije***

Vidimo rezultate testova ispod svakog pojedinog slanja zahtjeva. Testovi su uspješno prošli, što znači da se na svakom testiranom *endpointu* nalazi odgovarajući podatak.

Zamislimo situaciju da je došlo do izmjene kôda, neke neovlaštene aktivnosti i sl. i da je nekako obrisano ili izmijenjeno korisničko ime na *endpointu* `user/1`. Kad bi se to dogodilo, ponovno pokretanje kolekcije signaliziralo bi nam da je došlo do problema, tj. da se odgovor ne podudara s očekivanim i da je potrebno pobliže pogledati problem.

Osim s unaprijed definiranim vrijednostima (kao u ovom slučaju), odgovori se mogu uspoređivati s prethodnim odgovorima, s odgovorom nekog drugog zahtjeva, itd.



Slika 35 ***Zahtjev GET /user/1 nije prošao test***

Primjer scenarija u kojem bi automatizirano sigurnosno testiranje bilo korisno za *Vulnerable API* bio bi:

1) Znamo da je napadač uspijevao podmetnuti naredbu korištenjem funkcije *Vulnerable API*-ja *uptime* (detalji ovog napada opisani su u dokumentu Sigurnost HTTP API-ja).

2) Nakon što zakrpamo ranjivost, stvorimo kolekciju s nekoliko primjera slanja zahtjeva s različitim zastavicama, od kojih su neke legitimne (poput 's'), a neke zlonamjerne, tj. riječ je o umetanju naredbe (npr. '`uptime/s;pwd`'). Zahtjevi s legitimnim zastavicama se smiju izvršiti (statusni kôd 200 OK), a zahtjevi s umetnutim naredbama se moraju odbiti ili sanitizirati tako da se odbaci dio s umetnutom naredbom.

3) Stvorimo testove koji će provjeriti za svaki zahtjev je li se uspješno izvršio ili ne. Pokretanje kolekcije sa zahtjevima zakrpanom API-ju mora rezultirati prolaskom svih testova, tj. legitimni zahtjevi se izvrše, a zahtjevi s umetnutim naredbama ne izvrše.

4) Nakon izmjena u kôdu, objavljivanja nove inačice API-ja ili periodički, pokrećemo kolekciju s testovima i ako vidimo da je neki test s umetnutom naredbom uspješno prošao, znamo da ponovno moramo provjeriti API jer je izmjena kôda ponovno dovela do ranjivosti na umetanje (engl. *injection*).

4 Zaključak

Postmanovi autori na svojoj naslovnoj stranici izjavili su da je *Postman* jedini alat koji ima potpunu podršku za razvoj API-ja, i jasno je zašto to misle – *Postman* stvarno ima velik broj funkcionalnosti koje olakšavaju posao razvojnim programerima, ali i sigurnosnim stručnjacima. U ovom dokumentu demonstrirali smo neke od najvažnijih.

Osim nabrojanih, *Postman* ima još niz dodatnih funkcionalnosti poput *debuggiranja* kolekcija, slanja i primanja odgovora, pokretanja kolekcije iz naredbene linije (za što je potreban alat *Newman*), vizualizacija odgovora za grafički prikaz i lakše razumijevanje, razne statistike itd.

Kad je riječ o sigurnosnom testiranju, preporučljivo bi bilo da se ono provodi od početka razvoja softvera, ali nažalost zbog loše organizacije to se često tako ne provodi, nego se sigurnosno testiranje odradi samo neposredno prije objavljivanja konačnog proizvoda. Tu *Postman* također može pomoći jer se u njemu mogu stvoriti automatizirani testovi koji se mogu pokretati periodički ili nakon izmjena u kôdu. Detaljniju provjeru sigurnosti nije moguće automatizirati, već ju je potrebno je provesti ručno, a u tom slučaju korisna je kombinacija *Postmana* s nekim presretajućim HTTP posrednikom (engl. *intercepting proxy*) poput *Burp Suitea* ili OWASP ZAP-a.