API testiranje

Bešlić, Kristina

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

API TESTIRANJE

Diplomski rad

Kristina Bešlić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Kristina Bešlić
Studij, smjer:	Sveučilišni diplomski studij DRC - Programsko inženjerstvo
Mat. br. pristupnika, god.	D-1186R, 07.10.2021.
JMBAG:	0165076805
Mentor:	doc. dr. sc. Ivan Vidović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Tomislav Matić
Član Povjerenstva 1:	doc. dr. sc. Ivan Vidović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	API testiranje
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti dostupne alate i okruženja za automatizirano testiranje API-ja. Nakon provedenog istraživanja potrebno je odabrati jedan alat ili okruženje te ga usporediti s ostalim postojećim rješenjima. U praktičnom dijelu rada potrebno je primjeniti odabrani alat ili okruženje na proizvoljno odabranoj internet aplikaciji koja omogućava testiranje API-ja.

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	06.05.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	21.5.2024
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	17.06.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 17.06.2024.

Ime i prezime Pristupnika:	Kristina Bešlić
Studij:	Sveučilišni diplomski studij DRC - Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	D-1186R, 07.10.2021.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: API testiranje**

izrađen pod vodstvom mentora doc. dr. sc. Ivan Vidović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis pristupnika:

SADRŽAJ

1. UVOD

Tehnologija je postala neizostavan dio svakodnevice čovjeka. Ljudi su svaki dan okruženi velikim brojem informacija iz različitih izvora. Velik dio informacija dobiva se na zahtjev. Kako bi se moglo „surfati“ ili pogledati vremenska prognoza, kroz sučelje aplikacije mora se poslati zahtjev kako bi se u odgovoru dobile zahtijevane informacije. Tema ovog diplomskog rada je testiranje aplikacijskog programskog sučelja koje se odnosi upravo na testiranje ponašanja zahtjeva koji korisnik ili aplikacija preko API-ja šalju poslužitelju. Testiranje se može napraviti na različitim razinama, npr. na razini sigurnosti, performansi, responzivnosti itd. Nužno je testirati i podatke koji se dobiju u odgovoru poslužitelja kako bi se osigurala ispravnost podataka koji će biti prikazani korisniku. U ovom diplomskom radu odabrana su dva API-ja koja su testirana na različitim razinama, ali se fokus stavio na testiranje podataka dobivenih u odgovoru. Također, pokrivena je i sigurnosna razina koja je izrazito bitna za svakog korisnika. S povećanjem količine podataka te kompleksnosti istih javlja se potreba za automatiziranim testiranjem. Ručno testiranje postaje izrazito zahtjevno te podložno ljudskim pogreškama. Stoga se u ovom radu vrši automatizirano testiranje kroz Postman API platformu.

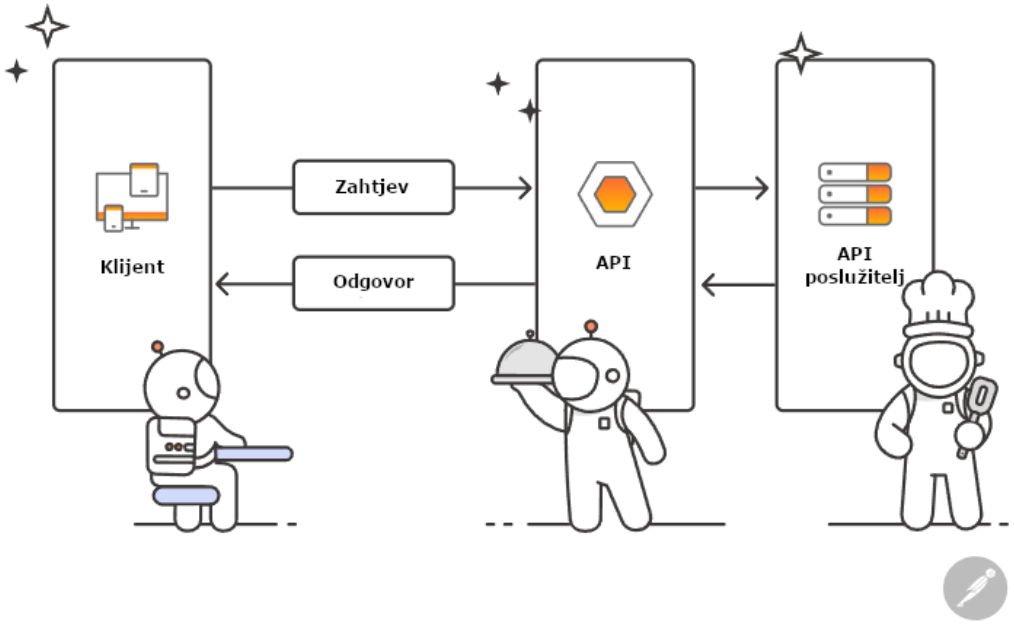
Vremenska prognoza jedna je od stavki koje korisnici pametnih telefona često koriste na dnevnoj razini. U ovom diplomskom radu jedan od testiranih API-ja je upravo API za dohvaćanje podataka o vremenskoj prognozi. Testirani su podaci iz odgovora kako bi se potvrdila ispravnost istih. Kako bi se osmislili i ispravno napisali testni slučajevi, potrebno je proučiti dokumentaciju za korišteni API te istražiti koje su očekivane vrijednosti za određene parametre iz odgovora. Tako je primjerice osigurano da je temperatura u očekivanom rasponu. Drugi testirani API je GitHub API. Kroz testiranje GitHub API-ja korištene su metode za dohvaćanje, kreiranje, izmjenu i brisanje resursa. Takve metode su općenito najčešće korištene u API testiranju. Postoje i druge metode koje su manje poznate i rjeđe se koriste.

Nakon uvodnog poglavlja, u drugom poglavlju ovog rada opisano je što je aplikacijsko programsko sučelje (API), u kojem je odnosu s klijentom i poslužiteljem te način na koji komunicira s njima. Također, opisani su dijelovi svakog zahtjeva, metode te statusni kodovi koji se mogu dobiti u odgovoru. Zatim su opisane vrste i načini testiranja API-ja. Navedeni su i ukratko opisani alati koji olakšavaju testiranje te izazovi koji se javljaju u doba umjetne inteligencije i strojnog učenja. Treće poglavlje opisuje Postman, API platformu koja je korištena za testiranje. Nadalje, ukratko se opisuju i dva testirana API-ja, Weather API i GitHub API. U četvrtom

poglavlju navedeni su i prikazani testni scenariji i testni slučajevi za svaki scenarij, dok je peto poglavlje rezervirano za rezultate testiranja oba API-ja. Završno poglavlje čini zaključak rada.

2. APLIKACIJSKO PROGRAMSKO SUČELJE

Aplikacijsko programsko sučelje (engl. *application programming interface*), API, set je protokola koji omogućavaju komuniciranje i izmjenu podataka između različitih programskih komponenti. [1] Kroz zahtjeve i odgovore dijele se podaci između aplikacija, sustava i uređaja. Prema pravima pristupa postoje 3 kategorije API-ja: privatni, partnerski i javni. [2] Privatni su korišteni unutar organizacije, partnerski između poslovnih partnera, dok javni API-ji imaju otvoren pristup. Prema [1], postoje 4 komponente odgovorne za proces slanja zahtjeva te vraćanja odgovora korisniku. Jedna od njih je API klijent koji šalje zahtjev API poslužitelju. API poslužitelj odgovoran je za provjeru autentifikacijskih podataka, validiranje ulaznih podataka te dohvaćanje i manipuliranje podacima. Slika 3.1. prikazuje jednostavan primjer slanja API zahtjeva između klijenta i poslužitelja na primjeru restorana. Klijent naručuje hranu, odnosno zahtijeva neki resurs. API se ponaša kao konobar koji prima narudžbu te ju prosljeđuje kuharima, odnosno prosljeđuje zahtjev API poslužitelju. API poslužitelj, koji na slici predstavlja kuhare, skuha hranu te ju konobar odnese klijentu, odnosno API poslužitelj obrađuje zahtjev i šalje odgovor klijentu preko API-ja.



Slika 3.1. Prikaz komunikacije između klijenta i poslužitelja.

Zahtjev može biti pokrenut na više načina, na primjer korisnik može unijeti pojam za pretraživanje ili pritisnuti gumb (engl. *button*). Također, može biti pokrenut vanjskim događajima, primjerice dolaskom obavijesti (engl. *notification*) vanjske aplikacije. API zahtjev će se ponašati i izgledati drukčije ovisno o tipu API-ja, ali svi zahtjevi sadržavaju komponente poput krajnje točke (engl. *endpoint*), metode, parametara, tijela zahtjeva (engl. *request body*) i zaglavlja zahtjeva (engl.

request header). API *endpoint* je URL (engl. *uniform resource locator*) preko kojeg se pristupa određenom resursu. Metode zahtjeva su tipovi operacija koje klijent želi napraviti na određenom resursu, poput dohvaćanja, kreiranja, ažuriranja i brisanja podataka. Parametri su vrijednosti predane u API *endpoint*-u kako bi se pružili određeni podaci API-ju koji će ih potom poslati severu koji će ih obraditi. Mogu biti uključeni u API zahtjev kao dio URL-a ili poslani u tijelu zahtjeva. Zaglavlja zahtjeva su parovi ključ-vrijednost (engl. *key-value pairs*) koji pružaju dodatne informacije o zahtjevu, primjerice tip sadržaja. Tijelo je glavni dio API zahtjeva jer sadrži sve podatke potrebne za kreiranje, ažuriranje ili brisanje resursa. Nakon slanja zahtjeva sa svim potrebnim parametrima, API poslužitelj ga obrađuje i šalje odgovor API klijentu. Odgovor obično sadržava zaglavlje i tijelo odgovora te statusni kod (engl. *status code*). HTTP zaglavlja odgovora vrlo su slična zaglavljima zahtjeva, međutim dodatno sadržavaju informacije o odgovoru poslužitelja. Tijelo odgovora sadrži podatke koje je klijent zahtijevao ili poruku o pogrešci ukoliko podaci u zahtjevu nisu ispravni ili ne postoji mogućnost obrađivanja zahtjeva. HTTP statusni kodovi su troznamenkasti kodovi koji indiciraju uspješnost obrade zahtjeva. Dijele se u 5 skupina. Skupine kreću rednim brojevima od 1 do 5, a najčešći kodovi su 200, 201, 401, 403, 404 itd. Statusni kod će biti vraćen u ovisnosti o metodi poslanog zahtjeva. Najčešće korištene metode su GET, PUT, POST i DELETE. Također, osim navedenih metoda postoje i PATCH, HEAD, OPTIONS, TRACE i CONNECT koje nisu toliko često korištene. U tablici 3.1. navedene su metode API zahtjeva i statusni kodovi koji se mogu dobiti uz pojedinu metodu. [3]

Tablica 3.1. Metode i statusni kodovi.

Naziv metode	Opis metode	Statusni kod	Opis statusnog koda
GET	Dohvaća resurse	200 OK	Resurs je dohvaćen
		404 Not Found	Poslužitelj ne podržava *endpoint*
		400 Bad Request	Kriva sintaksa u zahtjevu
POST	Kreira novi resurs	201 Created	Resurs je kreiran
		204 No Content	Nepostojeći resurs za predan URL
PUT	Ažurira postojeći resurs (zamijenjen je cijeli sadržaj resursa)	200 OK	Resurs je dohvaćen
		204 No Content	Nepostojeći resurs za predan URL
PATCH	Ažurira postojeći resurs (ažurira samo izmijenjene parametre resursa)	2xx	Zahtjev je uspješno zaprimljen i obrađen
DELETE	Briše resurs	2xx	Ovisno o API-ju, neki vraćaju 200 OK ili 204 No Content
HEAD	Dohvaća informacije o resursu, ali ne vraća tijelo odgovora	200 OK	Resurs je dohvaćen

2.1. Testiranje API-ja

Prema [4], testiranje uključuje detaljnu analizu API dokumentacije kako bi se razumjele funkcionalnosti te očekivani ulazni i izlazni parametri. Također, uključuje i pisanje testnih slučajeva koji provjeravaju različite funkcionalnosti API-ja, izvršavanje testnih slučajeva te uspoređivanje dobivenih rezultata s očekivanim rezultatima. Na kraju testiranja potrebno je analizirati rezultate i identificirati probleme koji trebaju biti ispravljeni, ukoliko postoje. Testiranje uključuje testiranje funkcionalnosti API-ja kako bi se osiguralo da se ponaša kao što je očekivano. Takva vrsta se naziva funkcionalno testiranje. Uz funkcionalno, postoji i sigurnosno testiranje koje osigurava da su informacije zaštićene, zatim testiranje performansi, interoperabilnosti API-ja s ostalim sustavima te mogućnosti korištenja API-ja od strane programera. Prema [5], postoje različiti tipovi testiranja kao što su jedinično testiranje (engl. *unit testing*), integracijsko testiranje (engl. *integration testing*), *end-to-end* testiranje, testiranje opterećenja (engl. *load testing*), već navedeno sigurnosno testiranje te *fuzz* testiranje. Prilikom API testiranja testira se točnost podataka, vrijeme dohvaćanja podataka, dupliciranje ili nedostatak funkcionalnosti, provjera autorizacije, provjera postojanosti sigurnosnih problema, problema performansi te izdržljivosti sustava prilikom velikog opterećenja. Postoje razni alati za testiranje API-ja, a neki od njih su Postman, Katalon Studio, Soap UI, Parasoft, REST assured, Ping API itd. Dio prethodno navedenih alata opisan je u idućem potpoglavlju. U ovom radu za testiranje korišten je Postman. Prema [6], UI (engl. *user interface*) testovi su neefikasni u validaciji API funkcionalnosti i često ne pokrivaju sve aspekte koji trebaju biti pokriveni testiranjem. API testiranje omogućava testiranje u ranoj fazi razvoja proizvoda jer ne ovisi o korisničkom sučelju, tako zahtjevi koji ne vraćaju ispravne vrijednosti neće biti prikazani u UI sloju (engl. *layer*). Prednost takvog pristupa je što se već u početku eliminira većina postojećih *bug*-ova. Izazovi API testiranja su potreba za definiranjem dobrog seta parametara koji će biti korišteni pri testiranju te različitih kombinacija parametara kako bi se pokrili svi testni slučajevi. Također, postoje grupe zahtjeva koje mogu zahtijevati da zahtjevi budu pozvani u točno određenom redoslijedu. Postoji mogućnost da testirani API radi ispravno kada se testira individualno, ali kada se više API-ja uključi u aplikaciju moguće je da se dogodi neočekivano ponašanje.

2.2. Alati za testiranje API-ja

2.2.1. Katalon Studio

Prema [7], Katalon Studio automatizirano je testno okruženje koje omogućava kreiranje i izvršavanje testova za različite tipove aplikacija poput web, API, mobilnih i desktop aplikacija.

Prednosti Katalon Studio-a su jednostavnost i lakoća implementacije, jednostavna i brza instalacija te dostupna besplatna verzija. Glavni nedostatak je što je zatvorenog koda.

2.2.2. Soap UI

Prema [8], Soap UI alat je za SOAP (engl. *Simple Object Access Protocol*) i REST (engl. *Representational State Transfer*) API testiranje. Otvorenog je koda. Omogućava kreiranje i izvršavanje automatiziranog funkcionalnog, regresijskog i testiranja opterećenja. Glavne prednosti su intuitivno sučelje, podržana automatizacija testiranja, testiranje vođeno podacima (engl. *data-driven testing*) i skriptiranje. Nedostaci su ograničeno testiranje performansi i ograničena podrška za formate podataka koji nisu XML ili JSON. Također, veliki nedostatak je kompleksno postavljanje i konfiguracija *mock* usluga.

2.2.3. JMeter

Prema [9], Apache JMeter je alat otvorenog koda primarno dizajniran za testiranje web aplikacija. Kasnije je proširio mogućnosti testiranja i na ostale usluge poput SOAP/REST, FTP (engl. *File Transfer Protocol*), baza podataka itd. Prednosti su što omogućava dobro testiranje performansi i izdržljivosti jer omogućava simulaciju velikog prometa na poslužitelju, grupi poslužitelja ili mreži. Na taj način se postiže dobro testiranje izdržljivosti i snage sustava. Međutim, nedostaci se javljaju kod pokretanja testova velikog opsega jer se značajno troše resursi memorije i procesora. Također, u takvim slučajevima se povećava i vrijeme odziva sučelja.

2.2.4. Usporedba Postman-a s navedenim alatima

Postman je alat čije je sučelje izrazito intuitivno za korištenje. Nudi vrlo jednostavno kreiranje i korištenje *mock* poslužitelja za razliku od prethodno opisanog Soap UI-a. Omogućava izrazito dobru organizaciju API zahtjeva kroz kolekcije (engl. *collections*) i okoline (engl. *environment*). Nedostatak Postman-a u odnosu na JMeter je što ne omogućava testiranje performansi na toliko visokoj razini.

2.3. Izazovi testiranja API-ja u doba umjetne inteligencije

U današnje vrijeme, testiranje API-ja postaje još bitnije zbog većeg rizika od novih *bug*-ova te sigurnosnih rizika. Razvojem umjetne inteligencije i strojnog učenja API testiranje postaje izazovno zbog toga što umjetna inteligencija može kreirati API-je koji su kompleksniji za testiranje nego tradicionalni ručno kreirani API-ji. Nadalje, moguće je kreiranje API-ja koji su osjetljivi na napade, stoga su sigurnosni rizici veći no ikada. No, postoje i dobre strane umjetne inteligencije. Benefiti su poboljšana pokrivenost testnih slučajeva, predviđanje i prevencija

pogreške, automatizacija ponavljajućih zadataka te učinkovito rukovanje kompleksnim testnim slučajevima. [10]

3. KORIŠTENI ALATI I TEHNOLOGIJE

U ovom poglavlju opisani su testirani API-ji te API platforma koja je olakšala testiranje istih.

3.1. Postman

Postman je API platforma koja sadrži alate za olakšavanje upravljanja i praćenja životnog vijeka API-ja. [11] Životni vijek API-ja sastoji se od dizajna, testiranja, dokumentiranja i simuliranja ponašanja do dijeljenja kreiranog API-ja. Prema [12], Postman API klijent je alat koji omogućava istraživanje, debugiranje i testiranje API-ja, ali i definiranje kompleksnih API zahtjeva. Istraživanje je nužno kako bi korisnik odlučio odgovaraju li informacije koje pruža određeni API njegovim potrebama. Takva vrsta istraživanja može biti izrazito komplicirana ukoliko dokumentacija nije dobro napisana ili nije ažurirana. Postman olakšava istraživanje API-ja jer nudi mogućnost izvršavanja kompleksnih API zahtjeva te vraćanja odgovora u izrazito kratkom vremenu. Također, omogućava prilagođavanje parametara kako bi se vidjelo ponašanje API-ja s različitim ulaznim vrijednostima (engl. *input values*). API klijent automatski detektira jezik odgovora te povezuje i formatira tekst unutar tijela odgovora (engl. *response body*) kako bi korištenje podataka iz odgovora bilo olakšano. Također, sadrži i podršku za autentifikacijske protokole (engl. *authentication protocols*) poput OAuth 1.2/2.0, AWS Signature, Hawk i ostale kako bi korisnici mogli sigurno i brzo pristupati API-jima. API klijent omogućava i organiziranje zahtjeva u kolekcije kako bi se olakšalo recikliranje zahtjeva te automatizirano testiranje grupe zahtjeva koja se nalazi u kolekciji. Testiranje je ključni dio životnog vijeka API-ja. Omogućeno je stalno praćenje ponašanja korištenog API-ja i otklanjanje pogrešaka u vrlo kratkom vremenskom periodu. Bez API klijenta, testiranje bi bilo vrlo teško i podložno ljudskim pogreškama. Ručno API testiranje nije prikladno u velikim organizacijama niti na vremenski ograničenim projektima. [13] Zahtjevi koji ovise o izlaznim vrijednostima prethodno izvršenih zahtjeva izrazito su izazovni za testiranje bez određenog alata. Mogućnost kreiranja varijabli na različitim razinama pojednostavljuje testiranje. Varijable mogu biti kreirane na razini radne okoline (engl. *workspace*), okoline i kolekcije, a korištene u autorizaciji, tijelu zahtjeva, testovima, kao URL parametri itd. Postman nudi mogućnost kreiranja sesijskih varijabli (engl. *session variables*) koje omogućavaju pohranu osjetljivih podataka poput API ključeva (eng. *API keys*). Takva vrsta varijabli pohranjena je lokalno na računalu korisnika te nije dijeljena s drugim korisnicima koji imaju pristup tom radnom okruženju. Postman nudi korištenje testnih skripti te skripti koje se izvršavaju prije slanja zahtjeva (engl. *pre-request script*). One omogućavaju pisanje testnih slučajeva za dinamičke podatke, slanje zahtjeva koji sadrže dinamičke parametre, korištenje dinamičkih podataka u

zahtjevima itd. Automatizacija API testova odnosi se na korištenje testnog alata kako bi se testovi izvršili određen broj puta ili u određenim vremenskim razmacima. Ti alati su posebno korisni timovima programera koji se bave agilnim razvojem proizvoda. Omogućen je brz razvoj proizvoda dok se kontinuirano i sistematično verificira da API radi kako je očekivano.

3.2. GitHub API

GitHub API može biti korišten za izradu skripti i aplikacija koje automatiziraju procese, integriraju se s GitHub-om te proširuju GitHub. Svaki REST API *endpoint* je individualno dokumentiran i kategoriziran prema resursu na koji primarno utječe. Prema [14], velik broj REST API *endpoint*-a zahtijeva autentifikaciju (engl. *authentication*) ili vraća dodatne informacije za autentificirane korisnike. Dodatno, moguće je poslati veći broj zahtjeva. Postoji više načina za dobivanje autentifikacijskog tokena, jedan od njih je da se kreira osobni pristupni token (engl. *personal access token*) na GitHub profilu. Taj način korišten je u ovom radu. Autentifikacija s korisničkim imenom i lozinkom nije omogućena. Postoje razni *endpoint*-i za GitHub API, a neki od korištenih u ovom radu odnose se na dohvaćanje repozitorija (engl. *repository*), grana (engl. *branches*), *commit*-ova, kreiranje novog repozitorija itd.

3.3. Weather API

Za dohvaćanje podataka o vremenskoj prognozi korišten je OpenWeather API. Kako bi se koristio ovaj API potrebna je registracija te kreiranje pristupnog tokena. Za ispravno korištenje ovog API-ja potrebno je detaljno proučiti dokumentaciju. Prema [15], dokumentacija je neophodni vodič sa stvarnim primjerima i opsežnim opisom svakog API zahtjeva, odgovora i parametara. Također, podaci se ne osvježavaju u trenutnom vremenu, stoga API pozive ne bi trebalo izvršavati više od jednom u vremenskom periodu od 10 minuta. OpenWeather API nudi razne *endpoint*-e, poput dohvaćanja trenutnih podataka o vremenskoj prognozi ili dohvaćanja podataka o vremenskoj prognozi za sljedećih 5 dana. Navedeni *endpoint*-i su korišteni u praktičnom dijelu ovog rada. Uz njih postoje i drugi, poput *endpoint*-a za povijesne ili statističke podatke, podatke o klimatskim promjenama itd.

3.4. Geocoding API

OpenWeather razvio je i Geocoding API, koji je prema [16], jednostavan alat za olakšano pretraživanje lokacija kada se radi s geografskim imenima ili koordinatama. Podržava dva načina; vraćanje koordinata za predan geografski naziv grada te obrnut slučaj u kojem se vraća geografski

naziv za predane koordinate. U praktičnom dijelu ovog rada korišten je kako bi se dohvatile koordinate za pojedini naziv grada.

4. DIZAJN TESTNIH SLUČAJEVA

Prilikom dizajna testnih scenarija i testnih slučajeva potrebno je upoznati se s dokumentacijom pojedinog API-ja, poznavati statusne kodove te mjerne jedinice u kojima su iskazane vrijednosti parametara u dobivenom odgovoru. Kako bi se mogli razumjeti pojedini testni slučajevi u tablicama 5.3. i 5.5., u tablici 5.1. bit će navedene vrijednosti statusnih kodova s objašnjenjima istih. [17] Također, potrebno je navesti kako brzina odgovora ovisi o kompleksnosti zahtjeva i veličini odgovora, ali ključan faktor je i brzina interneta. Zbog navedenog postoji mogućnost da će testni slučajevi koji se odnose na brzinu odziva u nekim iteracijama proći, dok će u drugima pasti, u ovisnosti o zauzetosti resursa mreže na koju je klijent spojen. U nastavku su navedeni i objašnjeni zahtjevi, testni scenariji i testni slučajevi napisani za pojedini testni scenarij za dva testirana API-ja – GitHub API i Weather API.

Tablica 5.1. Statusni kodovi i detaljan opis za kodove korištene u testnim slučajevima.

2xx – zahtjev klijenta je uspješno zaprimljen i obrađen od strane poslužitelja	
200	OK – zahtjev je uspješan, poslužitelj je klijentu vratio tražene podatke
201	Kreiran – zahtjev je uspješan, poslužitelj je kreirao novi resurs
204	Nepostojeći sadržaj – zahtjev je uspješan, ali poslužitelj nije vratio podatke klijentu
4xx – pogreška u zahtjevu klijenta	
401	Nedopušten pristup – klijent nema pristup zahtijevanom resursu
422	Nemogućnost obrade zahtjeva - poslužitelj ne može obraditi zahtjev klijenta jer sadrži nevažeće podatke

4.1. GitHub API

U tablici 5.2. navedene su metode, *endpoint*-i i opisi zahtjeva. Također, naveden je popis testnih scenarija za testiranje GitHub API-ja i korišteni podaci u tim testnim scenarijima. U tablici 5.3. nalazi se popis testnih slučajeva te opis istih za pojedine testne scenarije.

Tablica 5.2. Popis *endpoint*-a i testnih scenarija za GitHub API.

Metoda	**API *endpoint***
GET	https://api.github.com/users/{user}/repos
Opis zahtjeva:	
Dohvaćanje javnih repozitorija korisnika GitHub-a	
Korišteni podaci:	
{user} - KristinaBeslic	
Oznaka i naziv testnih scenarija:	
G-TS1	Get Public Repositories
G-TS12	Get Public Repositories After Changing Visibility of a Newly Created Repository
GET	https://api.github.com/user/repos
Opis zahtjeva:	
Dohvaćanje privatnih i javnih repozitorija autentificiranog korisnika GitHub-a	
Korišteni podaci:	
GitHub pristupni token	
Oznaka i naziv testnih scenarija:	
G-TS2	Get All Repositories
G-TS6	Requesting User Repositories With Incorrect Token
G-TS9	Get All Repositories After Creating a New Repository
G-TS13	Get All Repositories After Changing Visibility of a Newly Created Repository
POST	https://api.github.com/user/repos
Opis zahtjeva:	
Kreiranje novog repozitorija iz Postman-a	
Korišteni podaci:	
GitHub pristupni token	
Naziv repozitorija: APItesting	
Oznaka i naziv testnih scenarija:	
G-TS7	Create New Repository
G-TS8	Create Repository With The Same Name As Newly Created One
GET	https://api.github.com/repos/{user}/{repository}/commits
Opis zahtjeva:	
Dohvaćanje *commit*-ova odabranog javnog repozitorija	
Korišteni podaci:	
{user} - KristinaBeslic	
{repository} - android-vjestina-tmdb	
Oznaka i naziv testnih scenarija:	
G-TS3	Get Single Public Repository Commits

Metoda	API *endpoint*
GET	https://api.github.com/repos/{user}/{repository}/branches
Opis zahtjeva:	
Dohvaćanje grana odabranog javnog repozitorija	
Korišteni podaci:	
{user} - KristinaBeslic	
{repository} - android-vjestina-tmdb	
Oznaka i naziv testnih scenarija:	
G-TS4	Get Single Public Repository Branches
GET	https://api.github.com/repos/{user}/{repository}/commits
Opis zahtjeva:	
Dohvaćanje *commit*-ova odabranog privatnog repozitorija	
Korišteni podaci:	
GitHub pristupni token	
{user} - KristinaBeslic	
{repository} - vjezba	
Oznaka i naziv testnih scenarija:	
G-TS5	Get Single Private Repository Commits
GET	https://api.github.com/user/subscriptions
Opis zahtjeva:	
Dohvaćanje repozitorija na koje je pretplaćen autentificirani korisnik	
Korišteni podaci:	
GitHub pristupni token	
Oznaka i naziv testnih scenarija:	
G-TS10	Get a Repository Subscription
PATCH	https://api.github.com/repos/{user}/{repository}
Opis zahtjeva:	
Promjena vidljivosti (engl. *visibility*) repozitorija	
Korišteni podaci:	
{user} - KristinaBeslic	
{repository} - APItesting	
Oznaka i naziv testnih scenarija:	
G-TS11	Change Visibility of a Newly Created Repository
DELETE	https://api.github.com/repos/{user}/{repository}
Opis zahtjeva:	
Brisanje repozitorija	
Korišteni podaci:	
{user} - KristinaBeslic	
{repository} - APItesting	
Oznaka i naziv testnih scenarija:	
G-TS14	Delete Newly Created Repository

Tablica 5.3. Popis testnih slučajeva za testne scenarije iz tablice 5.2.

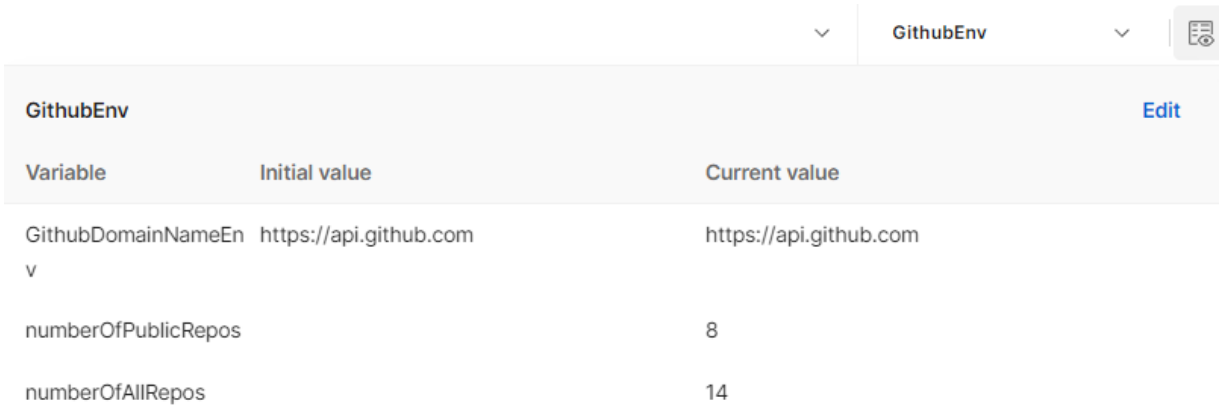
Oznaka testnog slučaja	Naziv testnog slučaja
G-TC1	**Status code is 200**
Opis testnog slučaja: Provjera je li statusni kod 200, odnosno je li zahtjev uspješno izvršen	
Očekivani rezultat: Statusni kod: 200 OK	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS1, G-TS2, G-TS3, G-TS4, G-TS 5, G-TS9, G-TS10, G-TS11, G-TS12, G-TS13	
G-TC2	**Response time is less than 500 ms**
Opis testnog slučaja: Provjera je li zahtjev izvršen u vremenskom periodu od 500 ms	
Očekivani rezultat: Vrijeme odgovora manje od 500 ms	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS1, G-TS2, G-TS3, G-TS4, G-TS5, G-TS6, G-TS7, G-TS8, G-TS9, G-TS10, G-TS11, G-TS12, G-TS13, G-TS14	
G-TC3	**Number of public repositories in JSON response is equal to the number of public repositories on the GitHub profile of the user**
Opis testnog slučaja: Provjera je li broj javnih repozitorija u odgovoru jednak broju javnih repozitorija na korisnikovom GitHub profilu	
Očekivani rezultat: Broj javnih repozitorija: 8	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS1	
G-TC4	**The JSON response contains the same number of repositories in a list as the authenticated user on GitHub**
Opis testnog slučaja: Provjera je li broj javnih i privatnih repozitorija u odgovoru jednak broju javnih i privatnih repozitorija na korisnikovom GitHub profilu	
Očekivani rezultat: Zbroj javnih i privatnih repozitorija: 14	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS2	
G-TC5	**The number of commits on the branch main is correct**
Opis testnog slučaja: Provjera je li broj *commit*-ova u odgovoru jednak broju *commit-ova* na grani *main* u repozitoriju *andorid-vjestina-tmdb* na GitHub profilu korisnika KristinaBeslic	
Očekivani rezultat: Broj *commit*-ova: 21	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS3	

Oznaka testnog slučaja	Naziv testnog slučaja
G-TC6	**The number and the names of branches are the same in a response and on the GitHub**
Opis testnog slučaja: Provjera jesu li broj i nazivi grana u odgovoru jednaki broju i nazivima grana u repozitoriju *andorid-vjestina-tmdb* na GitHub profilu korisnika KristinaBeslic	
Očekivani rezultat: Broj *commit*-ova: 2, nazivi grana: main, hw4	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS4	
G-TC7	**Status code is 401**
Opis testnog slučaja: Provjera je li statusni kod 401 (korisnik nema pristup zahtijevanom resursu, u ovom slučaju popisu repozitorija autentificiranog korisnika)	
Očekivani rezultat: Statusni kod: 401 Unauthorized	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS6	
G-TC8	**Status code is 201**
Opis testnog slučaja: Provjera je li statusni kod 201 (poslužitelj je uspješno izvršio zahtjev i kreirao novi resurs, u ovom slučaju repozitorij)	
Očekivani rezultat: Statusni kod: 201 Created	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS7	
G-TC9	**Created repository has name *APItesting***
Opis testnog slučaja: Provjera je li naziv novokreiranog repozitorija *APItesting*	
Očekivani rezultat: Naziv repozitorija: APItesting	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS7	
G-TC10	**Created repository is public**
Opis testnog slučaja: Provjera je li novokreirani repozitorij *APItesting* javan	
Očekivani rezultat: Vidljivost repozitorija: javan	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS7	

Oznaka testnog slučaja	Naziv testnog slučaja
G-TC11	**Status code is 422**
Opis testnog slučaja: Provjera je li statusni kod 422 (poslužitelj nije u mogućnosti izvršiti zahtjev zbog nevažećih podataka, u ovom slučaju naziva repozitorija)	
Očekivani rezultat: Statusni kod: 422 Unprocessable Entity	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS8	
G-TC12	**The JSON response contains API testing repository**
Opis testnog slučaja: Provjera sadrži li odgovor u kojemu su dohvaćeni javni repozitoriji korisnika KristinaBeslic novokreirani repozitorij *APItesting*	
Očekivani rezultat: Broj javnih repozitorija: 9	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS9	
G-TC13	**The user is subscribed to a newly created API testing repository**
Opis testnog slučaja: Provjera je li korisnik pretplaćen na novokreirani repozitorij *APItesting*	
Očekivani rezultat: Odgovor sadrži repozitorij naziva *APItesting*	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS10	
G-TC14	**The visibility of the newly created repository has been changed to private**
Opis testnog slučaja: Provjera je li vidljivost novokreiranog repozitorija promijenjena na vrijednost *private*	
Očekivani rezultat: Vidljivost repozitorija: privatan	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS11	
G-TC15	**The number of public repositories in the JSON response is equal to the number of public repositories for a particular user on GitHub after changing visibility of repository *API testing***
Opis testnog slučaja: Provjera je li broj javnih repozitorija nakon promjene vidljivosti novokreiranog repozitorija jednak broju javnih repozitorija na GitHub profilu korisnika KristinaBeslic	
Očekivani rezultat: Broj javnih repozitorija: 8	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS12	

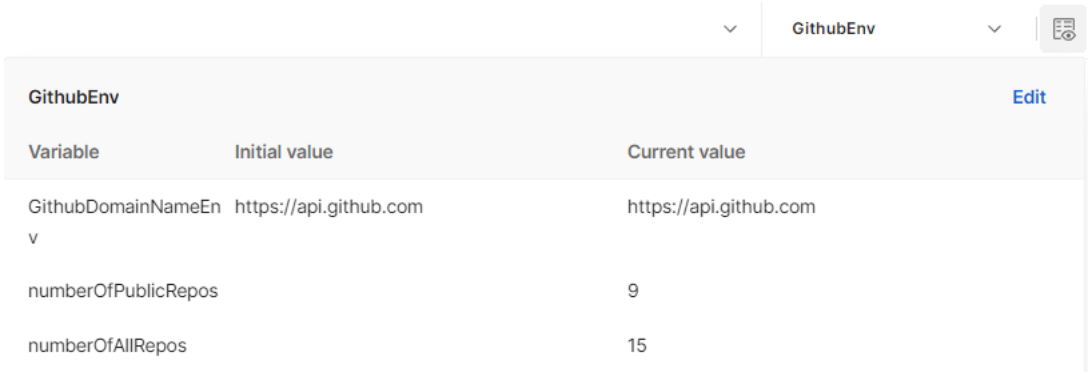
Oznaka testnog slučaja	Naziv testnog slučaja
G-TC16	**The JSON response contains the same number of repositories as the authenticated user on GitHub after changing visibility of a repository API testing**
Opis testnog slučaja: Provjera je li broj javnih i privatnih repozitorija nakon promjene vidljivosti novokreiranog repozitorija jednak broju javnih i privatnih repozitorija na GitHub profilu autentificiranog korisnika	
Očekivani rezultat: Zbroj javnih i privatnih repozitorija: 15	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS13	
G-TC17	**Status code is 204**
Opis testnog slučaja: Provjera je li vrijednost statusnog koda 204, odnosno je li repozitorij uspješno obrisan	
Očekivani rezultat: Statusni kod: 204 No content	
Popis testnih scenarija za koje se izvodi testni slučaj: G-TS14	

Zahtjevi u Postman-u smješteni su u kolekciju naziva GitHub API. Za tu kolekciju kreirana je okolina. Unutar okoline postoji mogućnost definiranja varijabli, što je korisno u slučaju kada se ista varijabla treba koristiti u više testnih scenarija. Okolina i definirane varijable za GitHub API prikazane su na slici 5.1. Ukoliko je potrebno, postoji mogućnost promjene vrijednosti varijabli, što je primijenjeno u testnom scenariju gdje se mijenja vidljivost repozitorija (G-TS11). Za određen API potrebno je koristiti odgovarajuću definiranu okolinu, u suprotnom postoji mogućnost pada testnih slučajeva zbog korištenja varijabli koje nisu definirane u okolini koja se koristi.



Slika 5.1. Prikaz okoline i varijabli.

Za svaki od testnih scenarija navedenih u tablici 5.2. napravljeni su testni slučajevi koji se odnose na provjeru statusnog koda te vremenskog perioda u kojem je poslužitelj odgovorio na klijentov zahtjev. Testni scenariji koji se odnose na dohvaćanje repozitorija korisnika sadržavaju i testne slučajeve koji provjeravaju je li broj repozitorija u odgovoru poslužitelja jednak stvarnom broju repozitorija na korisnikovom profilu na GitHub-u. Kreirane su dvije skupine takvih testnih scenarija, oni koji se odnose na repozitorije vezane uz autentificiranog korisnika (engl. *authenticated user*) te oni koji se odnose na repozitorije vezane uz neautentificiranog korisnika, odnosno bilo kojeg korisnika GitHub-a. Razlika u tim testnim scenarijima je što oni vezani uz autentificiranog korisnika u odgovoru poslužitelja sadržavaju sve repozitorije, neovisno o privatnosti repozitorija, dok za neautentificiranog korisnika poslužitelj u odgovoru vraća samo javne repozitorije. Takve vrste testnih scenarija se u tablici nalaze pod oznakama G-TS1, G-TS2, G-TS9, G-TS12, G-TS13. Testni scenarij G-TS9 sadrži i testni slučaj (G-TC12) u kojem se provjerava postojanje repozitorija u odgovoru poslužitelja nakon kreiranja novog repozitorija. U testnom scenariju G-TS7 u kojem se kreira novi repozitorij promijenjene su vrijednosti varijabli okoline (engl. *environment variables*) koje se odnose na broj repozitorija, što je vidljivo na slici 5.2. Programski kod 5.1. prikazuje programsko rješenje promjene vrijednosti varijabli okoline. Promjena vrijednosti varijabli odvija se nakon izvršavanja zahtjeva u odjeljku *Tests*.



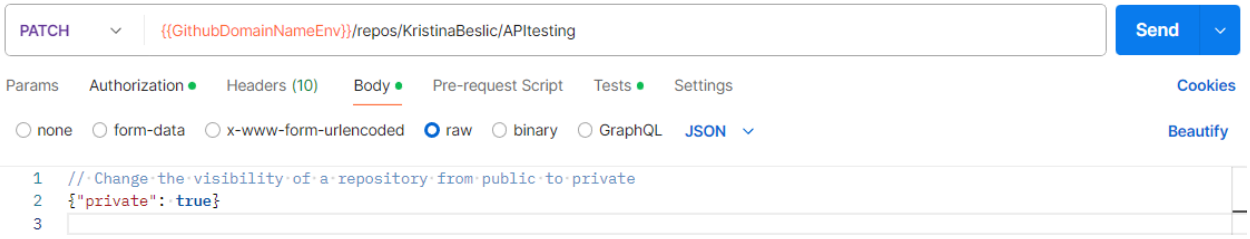
Slika 5.2. Prikaz varijabli okoline nakon kreiranja novog repozitorija.

```
pm.environment.set(
      "numberOfAllRepos",
      pm.environment.get("numberOfAllRepos") + 1
);

pm.environment.set(
      "numberOfPublicRepos",
      pm.environment.get("numberOfPublicRepos") + 1
);
```

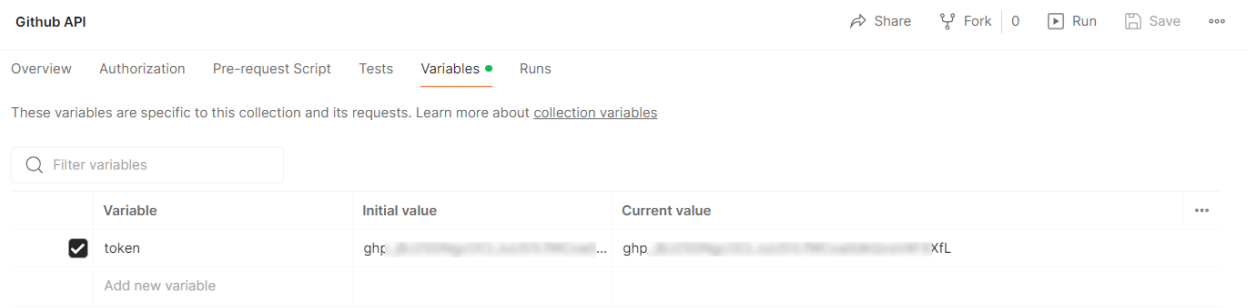
Programski kod 5.1. Prikaz modificiranja vrijednosti varijabli okoline nakon kreiranja novog repozitorija.

Testni scenarij G-TS10 provjerava je li korisnik pretplaćen na novokreirani repozitorij, dok testni scenariji G-TS12 i G-TS13 sadržavaju testne slučajeve (G-TC15, G-TC16) koji provjeravaju postojanje repozitorija u odgovoru poslužitelja nakon promjene vidljivosti repozitorija iz javnog u privatni koja se odvija u testnom scenariju G-TS11. Kao i u testnom scenariju G-TS7, u testnom scenariju G-TS11 izmijenjene su vrijednosti varijabli okoline na način da je vrijednost varijable *numberOfPublicRepos* umanjena za jedan. Također, u odjeljku *Body* mijenja se vrijednost parametra *private* iz *false* u *true*, što je vidljivo na slici 5.3. U trenutku kada je zahtjev poslan izvršena je promjena vidljivosti repozitorija.

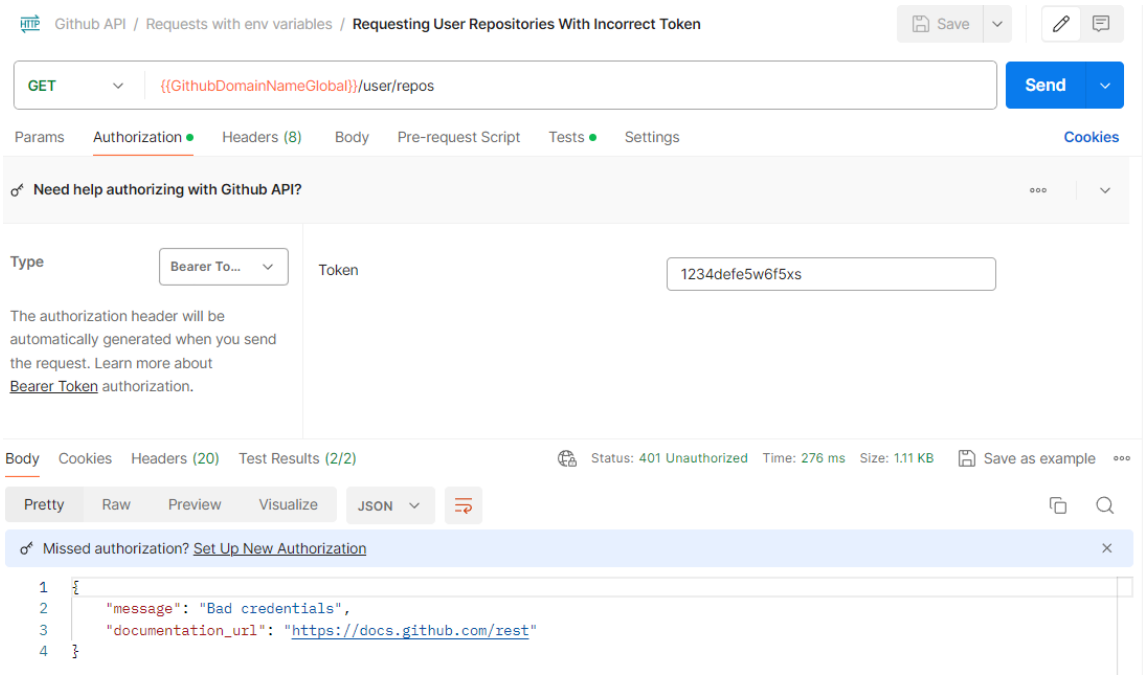


Slika 5.3. Promjena parametra za vidljivost novokreiranog repozitorija.

Testni scenariji G-TS3, G-TS4 i G-TS5 odnose se na dohvaćanje grana te *commit*-ova za privatne i javne repozitorije te sadržavaju testne slučajeve za provjeru broja istih u odgovoru poslužitelja u odnosu na broj grana i *commit*-ova na korisnikovom profilu na GitHub-u. Testni scenarij G-TS6 odnosi se na pokušaj dohvaćanja repozitorija za autentificiranog korisnika, ali s neispravnim GitHub pristupnim tokenom. Poslužitelj za taj testni scenarij vraća statusni kod 401 koji se odnosi na nemogućnost vraćanja podataka klijentu zbog neovlaštenog pristupa. Primjer ispravnog i neispravnog tokena vidljiv je na slikama 5.4. i 5.5. Pristupni token potreban je ukoliko se želi pristupiti privatnim podacima autentificiranog korisnika, u suprotnom nije potreban i moguće je izvršenje zahtjeva s uspješnim ishodom. Primjer testnog scenarija u kojem nije potreban pristupni token je G-TS1 gdje se dohvaćaju javni repozitoriji nekog korisnika GitHub-a.

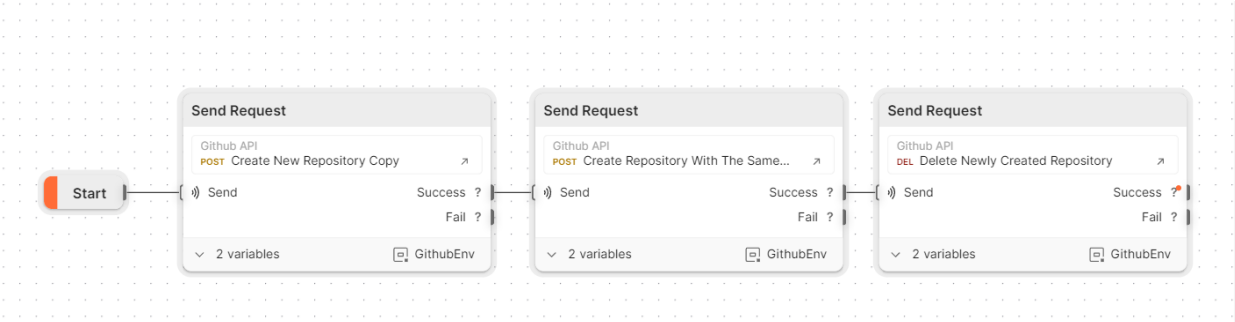


Slika 5.4. Ispravan pristupni token.



Slika 5.5. Pokušaj pristupa repozitorijima autentificiranog korisnika s neispravnim pristupnim tokenom.

U testnom scenariju G-TS7 koristi se POST metoda kojom je omogućeno kreiranje novog repozitorija. Kod kreiranja novog resursa, poslužitelj vraća statusni kod 201, a u navedenom testnom scenariju prilikom slanja zahtjeva potrebno je navesti i naziv repozitorija koji je u testnom scenariju G-TS7 postavljen na *„APItesting“*. Naziv novog repozitorija potrebno je postaviti u tijelu zahtjeva. Testni scenarij G-TS8 odnosi se na pokušaj kreiranja novog repozitorija istog naziva kao prethodno kreiran repozitorij. Budući da ne postoji mogućnost kreiranja više repozitorija istog naziva, očekivani statusni kod je 422 koji se odnosi na nemogućnost obrade zahtjeva. Nakon već navedenih testnih scenarija modificiranja vidljivosti novokreiranog repozitorija te provjere točnosti odgovora prilikom dohvaćanja privatnih i javnih repozitorija, vrši se brisanje novokreiranog repozitorija. Brisanje se vrši metodom DELETE nakon čijeg izvršavanja sever vraća statusni kod 204 koji se odnosi na uspješnost zahtjeva. U većini testnih scenarija korištena je metoda ulančavanja (engl. *API chaining*) u kojoj izlazna vrijednost jednog zahtjeva služi kao ulazna vrijednost sljedećeg zahtjeva. Primjerice, novokreirani repozitorij ne može biti izbrisan ako prethodno nije kreiran, odnosno, testni scenarij G-TS14 ne može se izvršiti prije testnog scenarija G-TS7. Za takve vrste povezanih zahtjeva Postman nudi alat *Flow* koji omogućava praćenje ponašanja ulančanih zahtjeva. Na slici 5.6. prikazan je primjer za testne scenarije G-TS7, G-TS8 i G-TS14. Nedostatak *Postman*-ovog *flow*-a je što ne može rukovati zahtjevima koji u sebi sadrže varijable okoline. Iz tog razloga, kako bi se prikazao način rada *flow*-a, potrebno je napraviti kopiju testnog scenarija G-TS7 u kojoj su izbrisane varijable okoline za promjenu broja privatnih i javnih repozitorija.



Slika 5.6. Prikaz *flow*-a za kreiranje i brisanje repozitorija.

4.2. Weather API

Za testne scenarije prilikom testiranja Weahter API-ja korištene su GET metode pomoću kojih su dohvaćene trenutna vremenska prognoza te vremenska prognoza za pet dana. Također, za iste su vizualizirani odabrani podaci pomoću Postman alata (engl. *tool*) za vizualizaciju podataka. U niže navedenim tablicama 5.4. i 5.5. prikazani su zahtjevi, odnosno testni scenariji te testni slučajevi osmišljeni za pojedine testne scenarije.

Tablica 5.4. Popis *endpoint*-a i testnih scenarija za Weather API.

Metoda	**API *endpoint***
GET	http://api.openweathermap.org/geo/1.0/direct?q={city name}&limit={limit}&appid={API key}
Opis zahtjeva:	
Dohvaćanje geografske dužine i širine iz naziva grada	
Korišteni podaci:	
{city name} - Osijek, New York	
{limit} - 1	
{API key} - Weather API pristupni token	
Oznaka i naziv testnih scenarija:	
W-TS1	Get Latitude and Longitude From the Given City Name
GET	https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
Opis zahtjeva:	
Dohvaćanje trenutačne vremenske prognoze iz geografske dužine i širine	
Korišteni podaci:	
{lat} - 45.5548793 (Osijek), 40.7127281 (New York), 14.73475430424682 (Tahoua)	
{lon} - 18.6953685 (Osijek), -74.0060152 (New York), 4.3110273439093305 (Tahoua)	
{API key} - Weather API pristupni token	
Oznaka i naziv testnih scenarija:	
W-TS2	Get Current Weather From Latitude and Longitude
W-TS4	Get Random Current Weather

Metoda	API *endpoint*
GET	https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
Opis zahtjeva:	
Dohvaćanje trenutačne vremenske prognoze iz naziva grada	
Korišteni podaci:	
{city name} - Osijek, New York	
{API key} - Weather API pristupni token	
Oznaka i naziv testnih scenarija:	
W-TS3	Get Current Weather From City Name
GET	https://api.openweathermap.org/data/2.5/forecast?q={city name}&appid={API key}
Opis zahtjeva:	
Dohvaćanje vremenske prognoze za 5 dana iz naziva grada	
Korišteni podaci:	
{city name} - Paris	
{API key} - Weather API pristupni token	
Oznaka i naziv testnih scenarija:	
W-TS5	Get Weather for 5 days From City Name
GET	https://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&appid={API key}
Opis zahtjeva:	
Dohvaćanje vremenske prognoze za 5 dana iz geografske dužine i širine	
Korišteni podaci:	
{lat} - 48.8588897 (Paris)	
{lon} - 2.3200410217200766 (Paris)	
{API key} - Weather API pristupni token	
Oznaka i naziv testnih scenarija:	
W-TS6	Get Weather for 5 days From Latitude and Longitude
GET	{mock domain}/data/2.5/forecast?lat={lat}&lon={lon}&appid={API key}
Opis zahtjeva:	
Dohvaćanje vremenske prognoze za 5 dana iz geografske dužine i širine s mock domenom	
Korišteni podaci:	
{lat} - 48.8588897	
{lon} - 2.3200410217200766	
{API key} - Weather API pristupni token	
Oznaka i naziv testnih scenarija:	
W-TS7	Get Weather for 5 days From Latitude And Longitude mock request

Tablica 5.5. Popis testnih slučajeva za testne scenarije iz tablice 5.4.

Oznaka testnog slučaja	Naziv testnog slučaja
W-TC1	**Status code is 200**
Opis testnog slučaja: Provjera je li statusni kod 200, odnosno je li zahtjev uspješno izvršen	
Očekivani rezultat: Statusni kod: 200 OK	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS1, W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC2	**Response time is less than 500 ms**
Opis testnog slučaja: Provjera je li zahtjev izvršen u vremenskom periodu od 500 ms	
Očekivani rezultat: Vrijeme odgovora manje od 500 ms	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS1, W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC3	**Content-Type is present**
Opis testnog slučaja: Provjera sadrži li zaglavlje tip sadržaja	
Očekivani rezultat: Tip sadržaja prikazan je u zaglavlju	
Popis zahtjeva koji sadrže testni slučaj: W-TS1, W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC4	**Response contains daily temperature in the form of a number**
Opis testnog slučaja: Provjera je li tip podatka dnevne temperature broj	
Očekivani rezultat: Tip podatka je broj	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC5	**Response contains a proper weather condition code**
Opis testnog slučaja: Provjera sadrži li odgovor ispravan kod za vremenske uvjete	
Očekivani rezultat: Odgovor sadrži jedan od navedenih kodova za vremenske uvjete; "Thunderstorm", "Drizzle", "Rain", "Snow", "Atmosphere", "Clear", "Clouds"	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	

Oznaka testnog slučaja	Naziv testnog slučaja
W-TC6	**The cloud coverage percentage is within acceptable bounds**
Opis testnog slučaja: Provjera je li podatak o postotku prekrivenosti oblacima unutar dozvoljenih vrijednosti	
Očekivani rezultat: Podatak o postotku prekrivenosti oblacima je između 0 i 100%	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC7	**The humidity percentage is within acceptable bounds**
Opis testnog slučaja: Provjera je li podatak o postotku vlažnosti zraka unutar dozvoljenih vrijednosti	
Očekivani rezultat: Podatak o postotku vlažnosti zraka je između 0 i 100%	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC8	**The visibility values are within acceptable limits**
Opis testnog slučaja: Provjera je li podatak o vidljivosti unutar dozvoljenih vrijednosti	
Očekivani rezultat: Podatak o vidljivosti je između 0 i 10 000 m	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC9	**The pressure values are within acceptable bounds**
Opis testnog slučaja: Provjera je li podatak o tlaku zraka unutar dozvoljenih vrijednosti	
Očekivani rezultat: Podatak o tlaku zraka je između 900 i 1050 hPa	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC10	**The response contains country and temperature is in expected range**
Opis testnog slučaja: Provjera sadrži li odgovor podatak o državi te je li temperatura u očekivanom rasponu	
Očekivani rezultat: Odgovor sadrži podatak o zemlji i temperatura iznosi između -50°C i +50°C ili -60°F i +120°F	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	
W-TC11	**The weather descriptions are valid**
Opis testnog slučaja: Provjera jesu li opisi vremenskih uvjeta ispravni	
Očekivani rezultat: Podaci za prikaz ikone, vrstu vremenskih uvjeta i opis istih su smisleni	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4, W-TS5, W-TS6, W-TS7	

Oznaka testnog slučaja	Naziv testnog slučaja
W-TC12	**The time from the response is not older than 30 minutes from the current time**
Opis testnog slučaja: Provjera je li podatak o vremenu stariji 30 minuta od trenutnog vremena	
Očekivani rezultat: Podatak o vremenu je unutar postavljenog vremenskog praga	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS2, W-TS3, W-TS4	
W-TC13	**There are 40 objects in a list**
Opis testnog slučaja: Provjera je li u listi 40 objekata	
Očekivani rezultat: Lista iz odgovora sadrži 40 objekata	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS5, W-TS6, W-TS7	
W-TC14	**Date is present in correct format**
Opis testnog slučaja: Provjera je li datum za svaki objekt iz liste prikazan u ispravnom formatu	
Očekivani rezultat: Datum je prikazan u ispravnom formatu	
Popis testnih scenarija za koje se izvodi testni slučaj: W-TS5, W-TS6, W-TS7	

U testnim scenarijima navedenima u tablici 5.4. djelomično je primijenjena metoda ulančavanja jer je za testne scenarije W-TS2 i W-TS6 potrebno prvo dohvatiti geografsku širinu i dužinu grada za koji se žele dobiti podaci o trenutnoj vremenskoj prognozi ili o vremenskoj prognozi za pet dana. Za dohvaćanje geografske širine i dužine iz naziva grada korišten je Geocoding API. Taj API korisniku za poslani naziv grada u zahtjevu, u odgovoru vraća naziv grada, državu te geografsku dužinu i širinu. Važno je napomenuti da se svaki zahtjev može izvršavati samostalno te nije potrebno koristiti metodu ulančavanja ukoliko se u pozivu zahtjeva ne koriste varijable okoline. Testni scenarij W-TS1 obrađen je pomoću Geocoding API-ja te *endpoint* prima naziv grada koji je u *Pre-request* skripti definiran i spremljen u kolekcijsku varijablu (engl. *collection variable*). U odgovoru zahtjev u testnom scenariju W-TS1 vraća već spomenute geografsku dužinu, širinu, državu te naziv grada. Geografska širina i dužina spremaju se u varijable okoline te su korištene u testnim scenarijima W-TS2, W-TS6 i W-TS7. Postavljanje vrijednosti varijabli okoline za geografsku širinu i dužinu prikazano je programskim kodom 5.2. Također, u zahtjevima nije nužno koristiti vrijednosti spremljene u varijablama, nego je u zahtjevu moguće i ručno unijeti vrijednosti za koje želimo dobiti odgovor.

```
var response = pm.response.json();

var collectionCityName;
postman.setEnvironmentVariable("lat", response[0].lat);
postman.setEnvironmentVariable("lon", response[0].lon);
postman.setEnvironmentVariable("city", response[0].name);
```

Programski kod 5.2. Dohvaćanje geografske širine i dužine pomoću Geocoding API-ja te spremanje istih u varijable okoline.

Testni scenarij W-TS2 ulančan je i ne može se pokrenuti samostalno ukoliko se koriste varijable okoline kao ulazne vrijednosti. One su postavljene i vraćene kao izlazne vrijednosti testnog scenarija W-TS1, u ovom slučaju konkretno je riječ o geografskoj dužini i širini. Testni scenariji W-TS2, W-TS3 i W-TS4 imaju iste vrste testnih slučajeva koji uz statusni kod i vrijeme odziva testiraju i jesu li dnevna temperatura i kod za vremenske uvjete (engl. *weather condition code*) u ispravnom formatu. Primjer navedenih testnih slučajeva prikazan je programskim kodom 5.3.

```
pm.test(Response contains daily temperature in the form of a number, function
() {
    pm.expect(response.main.temp).to.be.a(number);
});

pm.test("Response contains a proper weather condition code", function () {
    pm.expect(response.weather[0].main).to.be.oneOf([
        "Thunderstorm",
        "Drizzle",
        "Rain",
        "Snow",
        "Atmosphere",
        "Clear",
        "Clouds"
    ]);

});
```

Programski kod 5.3. Primjer testnih slučajeva za provjeru ispravnosti formata temperature i koda za vremenske uvjete.

Također, testirano je jesu li vrijednosti koje su izražene u postocima poput vlažnosti zraka i prekrivenosti oblacima unutar granica od 0 do 100 %. Nadalje, napisani su testni slučajevi koji provjeravaju je li vidljivost unutar granica navedenih u dokumentaciji te je li tlak zraka u očekivanom rasponu. Navedeni testni slučajevi prikazani su programskim kodom 5.4.

```
// Verify that the cloud cover percentage is within acceptable bounds (0-
100%)
pm.test("The cloud coverage percentage is within acceptable bounds",
function() {
    pm.expect(response.clouds.all).to.be.within(0, 100);
});

// Verify that the humidity percentage is within acceptable bounds (0-100%)
pm.test("The humidity percentage is within acceptable bounds", function() {
    pm.expect(response.main.humidity).to.be.within(0, 100);
```

```
});

// Verify that the visibility values are within acceptable limits (max.
10000m)
pm.test("The visibility values are within acceptable limits", function() {
    pm.expect(response.visibility).to.be.within(0, 10000);
});

// Verify that pressure values are within acceptable limits
pm.test("The pressure values are within acceptable bounds", function(){
    pm.expect(response.main.pressure).to.be.within(900, 1050);

});
```

Programski kod 5.4. Provjera ispravnosti podataka prekrivenosti oblacima, vlažnosti zraka, vidljivosti i tlaka zraka.

Za sve vrijednosti temperature provjereno je jesu li u očekivanom rasponu. U programskom kodu nalazi se izračun temperatura u stupnjevima Celzija, odnosno Fahrenheita, u ovisnosti o tome koja se mjerna jedinica koristi u zemlji za koju su dohvaćeni podaci. Budući da se u odgovoru poslužitelja dobiju podaci u Kelvinima, potrebno je preračunati iste u stupnjeve Celzija i Fahrenheita. Preračunavanje je potrebno izraditi zbog kasnije vizualizacije podataka koja je prikazana u idućem poglavlju. Programski kod 5.5. prikazuje uvjete i izračune granica očekivanog raspona za stupnjeve Celzija i Fahrenheita.

```
// Verify that the temperature is in expected range (-50°C to +50°C or -60°F
to +120°F)
pm.test("The response contains country and temperature is in expected range",
function () {
  var temperatureInKelvin = response.main.temp;
  var minTemperatureInKelvin = response.main.temp_min;
  var maxTemperatureInKelvin = response.main.temp_max;
  pm.expect(response.sys).to.have.property('country');
  if(country != 'US'
     && country != 'BZ'
     && country != 'PW'
     && country != 'BS'
     && country != 'KY'){
    pm.test("The country isn't US, BZ, PW, BS or KY", function() {
     pm.expect(country).to.not.have.any.keys(["US", "BZ", "PW", "BS", "KY"]);
     pm.test("The temperature is in the expected range", function() {
       temperatureInCelsius = Math.round(temperatureInKelvin - 273.15);
       minTemperatureInCelsius = Math.round(minTemperatureInKelvin - 273.15);
       maxTemperatureInCelsius = Math.round(maxTemperatureInKelvin - 273.15);

       pm.expect(temperatureInCelsius).to.be.within(-50, 50);
       pm.expect(minTemperatureInCelsius).to.be.greaterThan(-50);
       pm.expect(maxTemperatureInCelsius).to.be.below(50);
       boolCelsius = true;
     });
    });
  } else {
    pm.test("The country is US, BZ, PW, BS or KY", function() {
     pm.expect(country).to.be.oneOf(["US", "BZ", "PW", "BS", "KY"]);
     pm.test("The temperature is in the expected range", function() {
```

```
        temperatureInFahrenheit = Math.round(temperatureInKelvin * 9 / 5 -
459.67);
        minTemperatureInFahrenheit = Math.round(minTemperatureInKelvin * 9 / 5
- 459.67);
        maxTemperatureInFahrenheit = Math.round(maxTemperatureInKelvin * 9 / 5
- 459.67);

        pm.expect(temperatureInFahrenheit).to.be.within(-60, 120);
        pm.expect(minTemperatureInFahrenheit).to.be.greaterThan(-60);
        pm.expect(maxTemperatureInFahrenheit).to.be.below(120);
        boolFahrenheit = true;
      });
    });
  }
});
```

Programski kod 5.5. Preračun temperature i provjera valjanosti podataka.

Kreiran je testni slučaj za provjeru valjanosti dijela odgovora koji se odnosi na prikaz ikone, vrstu vremenskih uvjeta i opis istih. Dio odgovora koji sadrži kod ikone te vrstu i opis vremenskih uvjeta prikazan je na slici 5.7., dok je programskim kodom 5.6. prikazano testiranje odgovora za slike 5.7. Na isti način su napisane i provjere za ostale vremenske uvjete.


```
"weather": [
    {
        "id": 803,
        "main": "Clouds",
        "description": "broken clouds",
        "icon": "04n"
    }
],
```


Slika 5.7. Dio odgovora koji se odnosi na vremenske uvjete.

```
} else if(response.weather[0].main === Clouds) {
        pm.expect(response.weather[0].id).to.be.oneOf([
            801,
            802,
            803,
            804
        ]);
        pm.expect(response.weather[0].icon).to.be.oneOf([
            "02d", "02n",
            "03d", "03n",
            "04d", "04n"
        ]);
        pm.expect(response.weather[0].description).to.be.oneOf([
            "few clouds",
            "scattered clouds",
            "broken clouds",
            "overcast clouds"
        ]);
        if(response.weather[0].description === "few clouds"){
            pm.expect(response.clouds.all).to.be.within(11, 25);
            pm.expect(response.weather[0].id).is.eql(801);
        } else if(response.weather[0].description === "scattered clouds"){
            pm.expect(response.weather[0].id).is.eql(802);
            pm.expect(response.clouds.all).to.be.within(25, 50);
```

```
        } else if(response.weather[0].description === "broken clouds") {
            pm.expect(response.weather[0].id).is.eql(803);
            pm.expect(response.clouds.all).to.be.within(51, 84);
        } else if(response.weather[0].description === "overcast clouds"){
            pm.expect(response.weather[0].id).is.eql(804);
            pm.expect(response.clouds.all).to.be.within(85, 100);
        }
    }

});
```

Programski kod 5.6. Dio programskog rješenja koji se odnosi na provjeru vremenskih uvjeta sa slike 5.7.

Budući da se podaci sa svih meteoroloških postaja ne šalju u stvarnom vremenu, odnosno svake minute ili sekunde, potrebno je izračunati vremenske pragove unutar kojih se očekuje dobivanje podataka. Kako dan odmiče, tako vremenski period u kojemu se dobivaju podaci postaje veći, stoga je potrebno postaviti prag u vremenskom periodu od 30 minuta od trenutnog vremena. Donja granica praga je 30 minuta manja od stvarnog vremena, dok je gornja granica trenutno vrijeme. Programski kod 5.7. prikazuje kako je ostvareno programsko rješenje za izračun donje i gornje granice vremenskog praga.

```
var timestamp = response.dt * 1000; // Convert seconds to milliseconds
var date = new Date(timestamp);

pm.test("The time from the response is not older than 30 minutes from the
current time ", function(){
    var threshold30minutes = 1800000;
    pm.expect(timestamp).to.be.within(
        Date.now() - threshold30minutes,
        Date.now());
});
```

Programski kod 5.7. Programsko rješenje izračuna gornje i donje granice vremenskog praga.

Kako bi se mogli vizualizirati podaci na željeni način, potrebno je iskoristiti instancu klase *Date*, čije je kreiranje vidljivo u programskom kodu 5.7. Također, iz odgovora je potrebno pristupiti podacima za ikonu, brzinu vjetra, naziv grada, temperaturu i vlagu zraka te po potrebi prilagoditi njihov ispis kako bi se podaci vizualizirali na željeni način. U prikazu programskog koda 5.8. vidljivo je kako su prilagođeni podaci ispisa za datum i brzinu vjetra te je prikazana mjerna jedinica temperature u skladu s područjem za koje je dobiven odgovor.

```
var formattedDate = date.toLocaleDateString(
    'en-US',
    {year: 'numeric',
    month: '2-digit',
    day: '2-digit',
    hour: '2-digit',
    minute: '2-digit', hour12: false}
);
var iconUrl = "http://openweathermap.org/img/wn/" +
            response.weather[0].icon +
```

```
            ".png";
var wind = (response.wind.speed).toFixed(1);

var template = `
    <div style="display: flex; align-items: center; justify-content: space-
between; padding: 10px; border: 1px solid #ccc; border-radius: 5px;">
        <div style="flex: 0 0 auto; margin-right: 20px";>
            <img src="${iconUrl}"
            alt="Weather Icon"
            style="width: 50px; height: 50px;">
        </div>
        <div style="flex: 1;">
            <p style="margin: 0; font-size: 18px;">
                ${response.name}, ${formattedDate} h</p>
            <p style="margin: 0; font-size: 14px;">
                Temperature: {{response.main.temp}}</p>
            <p style="margin: 0; font-size: 14px;">
                Humidity: {{response.main.humidity}}%</p>
            <p style="margin: 0; font-size: 14px;">
                Wind: ${wind} km/h</p>
        </div>
    </div>
`;
if(boolCelsius){
    renderedTemplate = template.replace(
        '{{response.main.temp}}',
        temperatureInCelsius + "°C");
} else {
    renderedTemplate = template.replace(
        '{{response.main.temp}}',
        temperatureInFahrenheit + "°F");
}
// Set visualizer
pm.visualizer.set(renderedTemplate, {
    response: pm.response.json()
});
```

Programski kod 5.8. Prikaz programskog rješenja za vizualizaciju podataka iz odgovora.

Za testni scenarij W-TS4 kreirane su slučajne vrijednosti geografske dužine i širine čije je kreiranje prikazano programskim kodom 5.9. Vrijednosti su kreirane u skripti koja se izvršava prije slanja zahtjeva, a naziva se *Pre-request script*. Kreirane vrijednosti spremljene su u varijable okoline naziva *randLat* i *randLon* te su iskorištene u testnom scenariju za dohvaćanje vremenske prognoze za slučajno odabran grad.

```
// Generate random decimal number for latitude between -90 and 90
const minLat = -90;
const maxLat = 90;
const diffLat = maxLat - minLat;
const randomLatitude = (Math.random() * diffLat) + minLat;

// Generate random decimal number for longitude between -180 and 180
const minLon = -180;
const maxLon = 180;
const diffLon = maxLon - minLon;
const randomLongitude = (Math.random() * diffLon) + minLon;
```

```
var randLat = postman.setEnvironmentVariable("randLon", randomLatitude);
var randLon = postman.setEnvironmentVariable("randLon", randomLongitude);
```

Programski kod 5.9. *Pre-request* skripta za kreiranje slučajnih vrijednosti geografske širine i dužine.

Testni scenariji W-TS5 i W-TS6 imaju iste testne slučajeve, razlika je kao i kod testnog scenarija za dohvaćanje trenutačne vremenske prognoze, što se za testni scenarij W-TS5 podaci o vremenskoj prognozi dobivaju iz imena grada, dok se u testnom scenariju W-TS6 podaci dobivaju iz geografske dužine i širine. U ovim testnim scenarijima se u odgovoru dobiva vremenska prognoza za pet dana. Odgovor sadrži listu od četrdeset objekata, od kojih svaki objekt sadržava podatke koji su se dobivali i slanjem zahtjeva za dohvaćanje trenutne vremenske prognoze. Svaki objekt je vremenski udaljen tri sata od prethodnog, iz tog razloga postoji četrdeset objekata za vremensku prognozu za pet dana. Testni slučajevi su slični kao za prethodno navedene testne scenarije za dohvaćanje trenutačne vremenske prognoze. Razlika je u tome da su u svakom testnom slučaju testirane vrijednosti parametara za svih četrdeset objekata. Uz već spomenute i objašnjene testne slučajeve, izrađen je i testni slučaj koji provjerava jesu li podaci u odgovoru predstavljeni za pet dana, odnosno sadrži li lista četrdeset objekata. Taj testni slučaj prikazan je programskim kodom 5.10.

```
// Verify that there are 40 results in a list (5 days * (24 hours / 3 hours))
pm.test("There are 40 objects in a list", function () {
    var listLength = response.list.length;
    pm.expect(listLength).to.be.eql(40);
    pm.expect(listLength).to.be.eql(response.cnt);
});
```

Programski kod 5.10. Provjera broja objekata u listi.

Također, provjerava se je li datum zapisan u ispravnom formatu te je li datum u zadnjem objektu u listi točno pet dana udaljen od trenutačnog datuma. Prethodno navedenim osigurava se ispravnost dobivenih podataka, a programsko rješenje tog testnog slučaja prikazano je programskim kodom 5.11.

```
// Verify that the date field corresponds to the correct date and time for
each forecast entry
pm.test("Date is present in a correct format", function () {
    function addDaysToDate(date, days) {
        var result = new Date(date);
        result.setDate(result.getDate() + days);
        return result.toLocaleDateString('en-CA');
    }

    var date = (response.list[0].dt_txt).split(' ')[0];
    var startDate = new Date(date);
    var newDate = addDaysToDate(startDate, 5);
```

```
    // Verify that the last element in a list is 5 days apart from the date
of the first element in a list
    pm.expect(newDate).to.be.eql((response.list[39].dt_txt).split(' ')[0]);

    for(let i = 0; i < response.list.length; i++){
        pm.expect(response.list[i].dt_txt).
            to.match(/^\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}$/);
    }

});
```

Programski kod 5.11. Provjera ispravnosti formata datuma te ispravnosti zadnjeg datuma u listi.

Kako bi se provjerila i dokazala ispravnost testnih slučajeva, napravljen je testni scenarij u kojem se dohvaća vremenska prognoza za pet dana koji u odgovoru ima netočne podatke (engl. *mock data*). Takve vrste podataka koriste se kako bi se simuliralo ponašanje testnih slučajeva u slučaju dobivanja netočnih podataka. [18] Kako bi se izvršila takva vrsta zahtjeva, potrebno je izraditi *mock* poslužitelj te spremiti odgovor iz testnog scenarija za koji se želi kreirati lažni zahtjev (engl. *mock request*). Kada je odgovor spremljen, potrebno ga je ispraviti na željeni način te prilagoditi URL u zahtjevu. Umjesto pravog naziva domene u URL-u potrebno je staviti domenu od kreiranog *mock* poslužitelja. Ako su testni slučajevi dobro napisani te podaci izmijenjeni na dobar način, određeni testni slučajevi bi trebali pasti. U testnom scenariju W-TS7 testni slučajevi su izmijenjeni na način da prolaze s netočnim podacima, a primjer izmijenjenog testnog slučaja iz prikaza programskog koda 5.10. prikazan je u programskom kodu 5.12. Naime, podaci su izmijenjeni na način da je iz liste izbrisan jedan objekt te se u odgovoru testiranom u programskom kodu 5.12. u listi nalazi trideset devet objekata.

```
// Verify that there are 40 results in a list (5 days*(24 hours/3
hours))
pm.test("There are 40 objects in a list", function () {
    var listLength = response.list.length;
    pm.expect(listLength).to.be.eql(39);
    pm.expect(listLength + 1).to.be.eql(response.cnt);
});
```

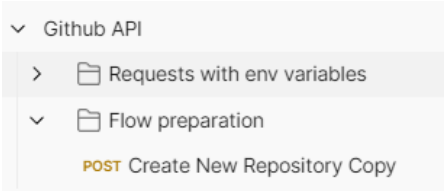
Programski kod 5.12. Primjer testnog slučaja prilagođenog *mock* podacima.

5. REZULTATI TESTIRANJA

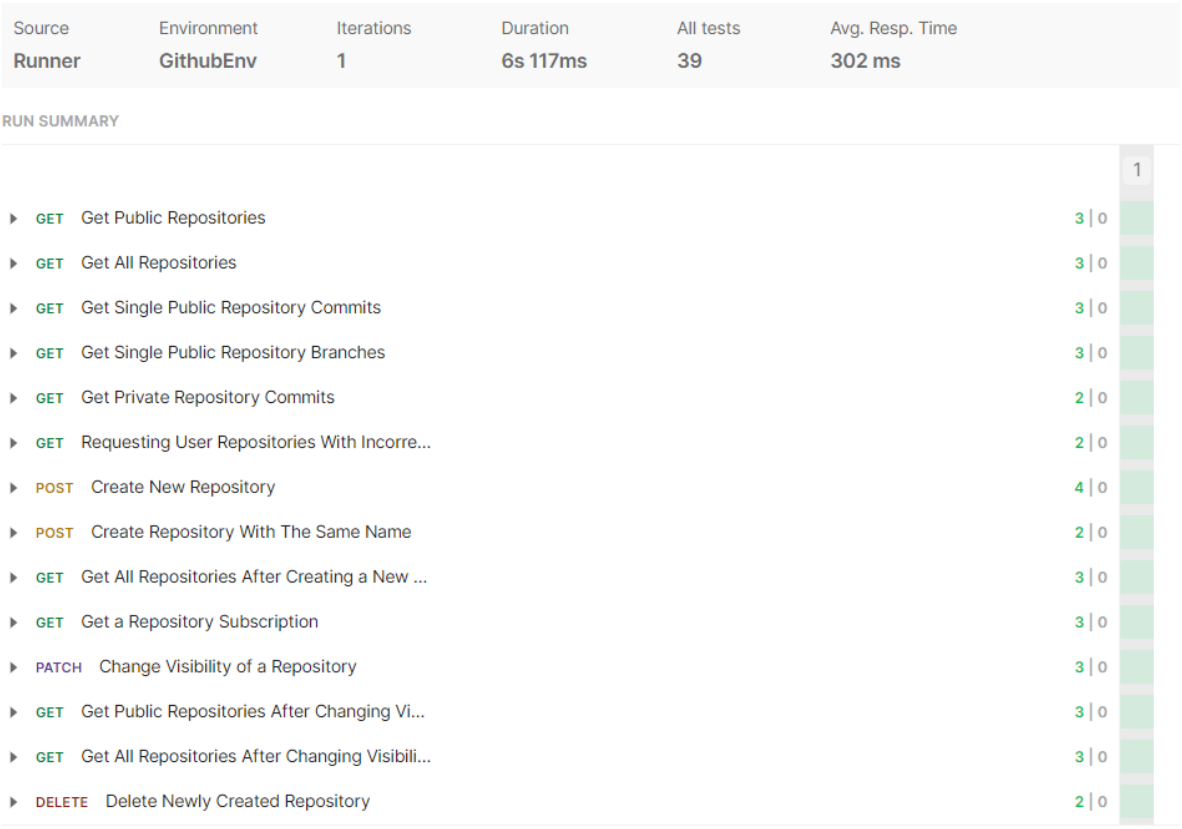
U ovom poglavlju prikazani su rezultati testiranja za GitHub i Weather API.

5.1. Rezultati testiranja za GitHub API

Postman nudi mogućnost automatiziranog testiranja. Omogućeno je pokretanje cijele kolekcije, ali i izrada mapa (engl. *folders*) koje je moguće samostalno pokrenuti. Kolekcija GitHub API podijeljena je u dvije mape, što je vidljivo na slici 6.1. U jednoj mapi nalaze se zahtjevi koji koriste varijable okoline, dok druga sadrži samo jedan zahtjev bez varijabli okoline. Taj zahtjev koristi se samo u *flow*-u zbog već navedenog problema s pokretanjem *flow*-a s varijablama okoline. Rezultati dobiveni automatiziranim pokretanjem mape nalaze se na slici 6.2.

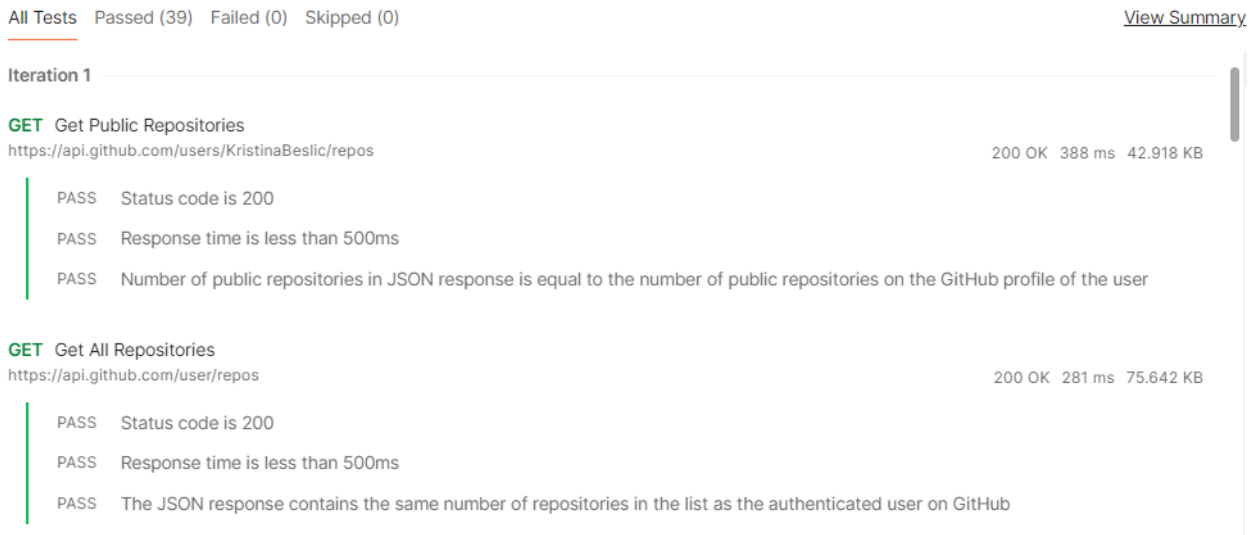


Slika 6.1. Organizacija zahtjeva za GitHub API.



Slika 6.2. Rezultati automatiziranog testiranja GitHub API-ja.

Postman nudi mogućnost ispisa svih testnih slučajeva i njihovih rezultata za pojedini testni scenarij. Također, za testne scenarije se u takvoj vrsti ispisa može vidjeti i *endpoint*, statusni kod, vrijeme potrebno za dohvaćanje odgovora te veličina odgovora. Za sve zahtjeve je vremenski limit izvršavanja postavljen na 500 ms, osim za kreiranje novog repozitorija za koji je postavljen na 1000 ms. Za kreiranje repozitorija uvijek je potreban veći vremenski period. Već je navedeno kako brzina odgovora na zahtjev između ostalog ovisi i o kompleksnosti zahtjeva, stoga se može zaključiti kako je zahtjev za kreiranje resursa kompleksniji nego zahtjevi za dohvaćanje, modificiranje i brisanje resursa. Na slici 6.3. vidljiva je vizualizacija rezultata za pojedinačne testne scenarije.

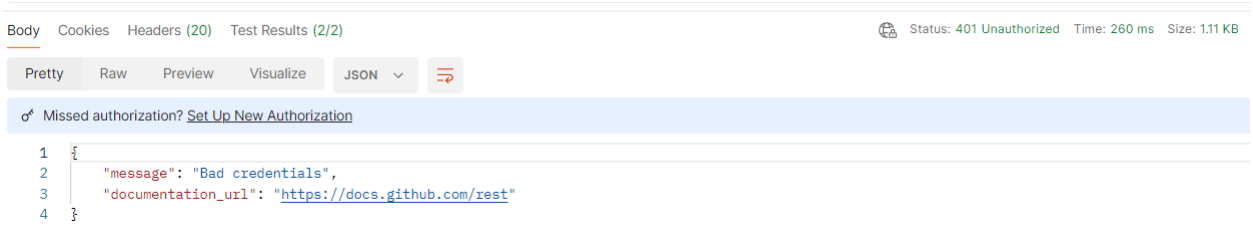


Slika 6.3. Vizualizacija rezultata automatiziranog testiranja kolekcije GitHub API.

Također, postoji mogućnost samostalnog pokretanja testnih scenarija, ali je potrebno obratiti pažnju jesu li testni scenariji povezani. Ukoliko ulazna vrijednost drugog testnog scenarija ovisi o izlaznoj vrijednosti prvog, a testni scenariji nisu pokrenuti dobrim redoslijedom, testni slučajevi ovisni o tim vrijednostima će pasti. U slučaju GitHub API-ja, ne postoji mogućnost brisanja repozitorija ako on ne postoji, odnosno nije prethodno kreiran. Takav slučaj prikazan je na slici 6.4.

Slika 6.4. Greška prilikom brisanja repozitorija.

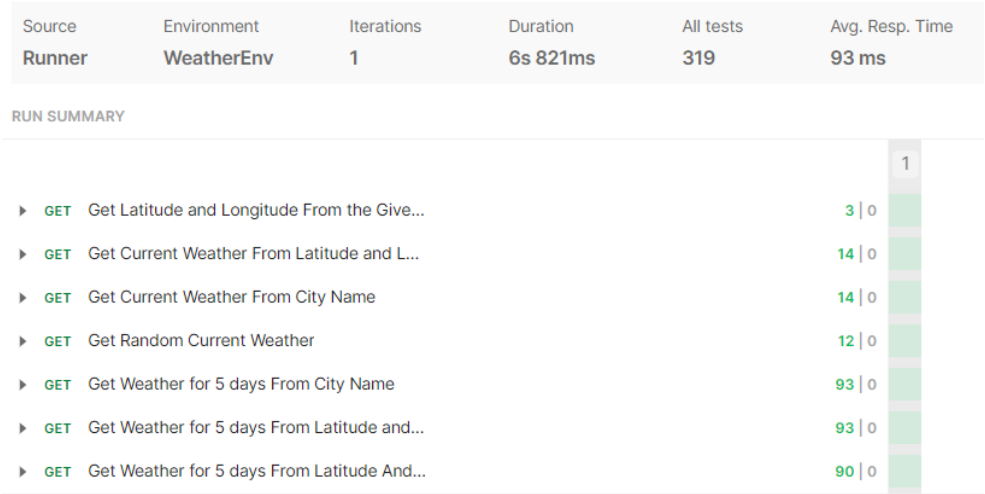
Kao što je već spomenuto, za neke zahtjeve potrebno je imati pristupni token. Tako zahtjevi za pristup javnim repozitorijima nekog korisnika ne zahtijevaju pristupni token, dok zahtjevi za pristup privatnim repozitorijima zahtijevaju. Pristupni token na GitHub-u može zatražiti svaki korisnik GitHub-a za sebe. To je privatni token koji se ne smije dijeliti s drugima. Ukoliko se pristupni token podijeli s drugima, postoje sigurnosni rizici za profil korisnika. Tako se primjerice mogu brisati postojeći repozitoriji, kao što je učinjeno s DELETE metodom u zahtjevu za novokreirani repozitorij. Na slici 6.5. prikazano je kako se neispravnim pristupnim tokenom ne može pristupiti privatnim informacijama niti upravljati računom korisnika.



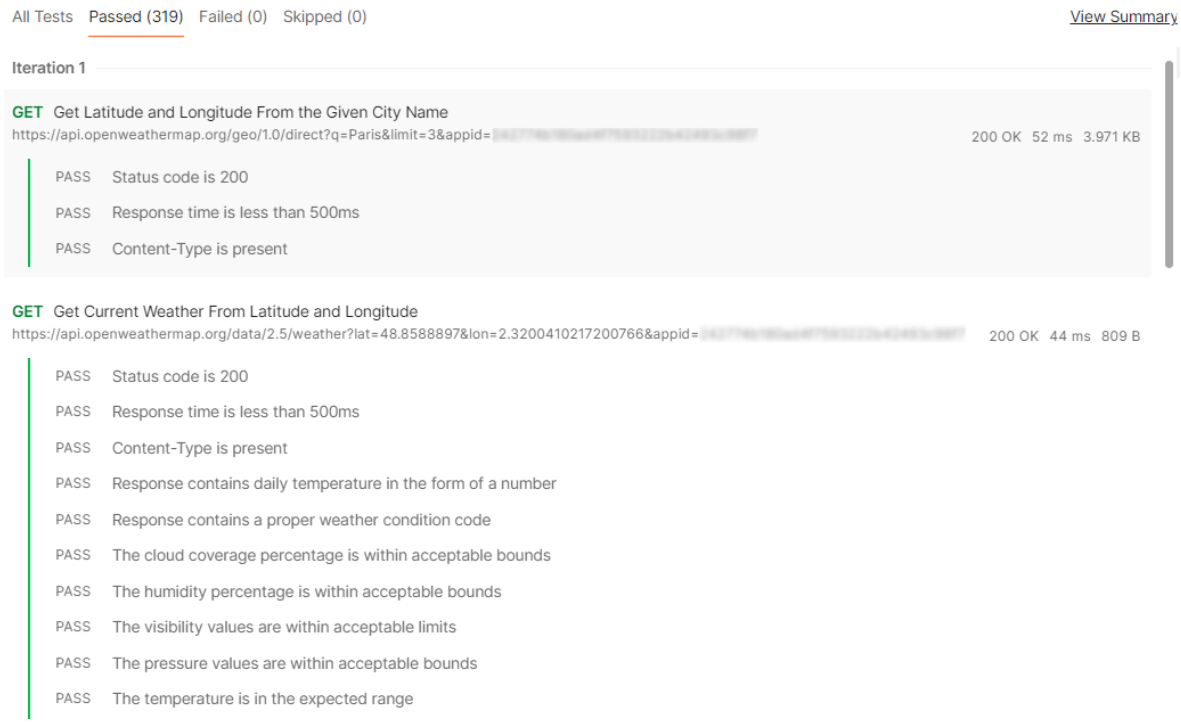
Slika 6.5. Odgovor u slučaju korištenja neispravnog pristupnog tokena.

5.2. Rezultati testiranja za Weather API

Za Weather API korišteni su zahtjevi za dohvaćanje podataka o trenutačnoj vremenskoj prognozi te vremenskoj prognozi za 5 dana. Postoji mogućnost pristupa podacima preko geografske dužine i širine grada ili preko naziva grada. Ukoliko se želi pristupiti geografskoj širini i dužini preko naziva grada potrebno je koristiti Geocoding API. Vremenski period od 500 ms je dovoljan prilikom dohvaćanja podataka u svim zahtjevima za Weather API. Na slici 6.6. nalazi se vizualizacija rezultata automatiziranog testiranja, dok je na slici 6.7. vidljiva vizualizacija rezultata istog za pojedinačne testne scenarije.

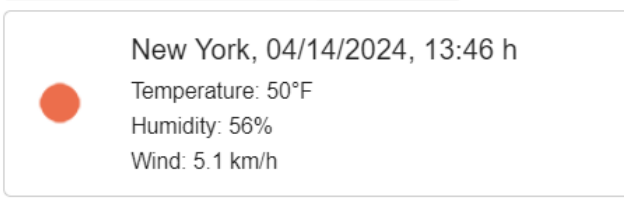


Slika 6.6. Rezultati automatiziranog testiranja Weather API-ja.



Slika 6.7. Vizualizacija automatiziranog testiranja dijela pojedinačnih testnih scenarija za Weaher API.

Za sve dohvaćene informacije o vremenskoj prognozi vizualizirani su podaci koji su prilagođeni mjernim jedinicama zemlje za koju su dohvaćeni podaci. Tako je za New York prikazana temperatura u stupnjevima Fahrenheita, što je vidljivo na slici 6.8., dok je za Osijek prikazana u stupnjevima Celzija, što je vidljivo na slici 6.9.

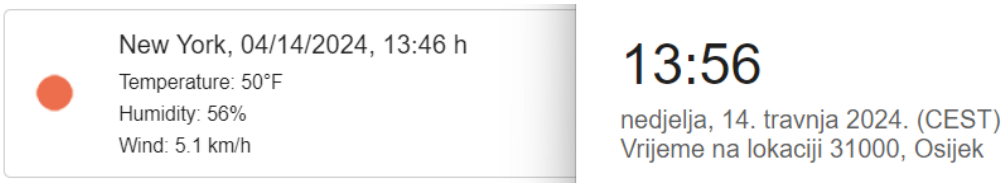


Slika 6.8. Vizualizacija trenutačne vremenske prognoze za New York.

Slika 6.9. Vizualizacija trenutačne vremenske prognoze za Osijek.

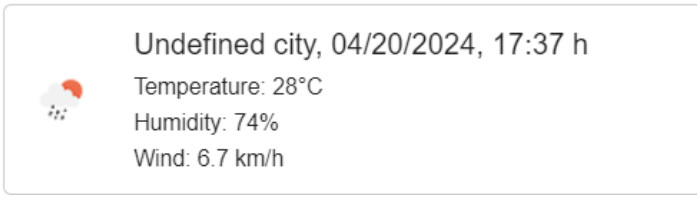
Na slici 6.10. vidljivo je kako podaci o trenutačnoj vremenskoj prognozi nisu dohvaćeni u stvarnom vremenu. Iz tog razloga, u testne slučajeve za provjeru je li trenutačna vremenska prognoza dohvaćena za ispravan dan i sat, potrebno je postaviti vremenski prag koji osigurava kako rezultati neće biti zastarjeli. U testnim slučajevima predstavljenima u ovom radu korišten je

vremenski prag od 30 minuta, u kojemu je gornja granica trenutačno vrijeme, dok je donja granica 30 minuta manja od trenutačnog vremena.



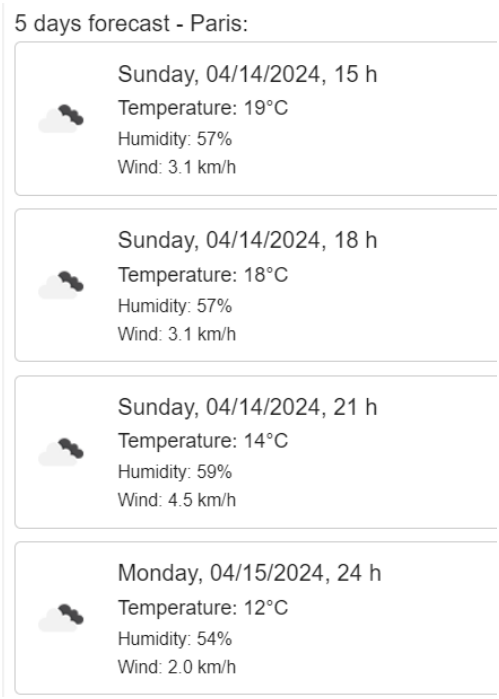
Slika 6.10. Prikaz vizualiziranih dobivenih podataka te vremena u kojem je zahtjev poslan i dobiven.

Prilikom kreiranja slučajnih vrijednosti geografske širine i dužine, moguće je dobiti podatke za grad za koji se u odgovoru ne dobije njegov naziv. U takvim slučajevima su vizualizirani podaci, ali bez naziva grada, što je vidljivo na slici 6.11. Također, temperatura je izražena u stupnjevima Celzija, budući da se za samo 11 država u svijetu koriste stupnjevi Fahrenheita.



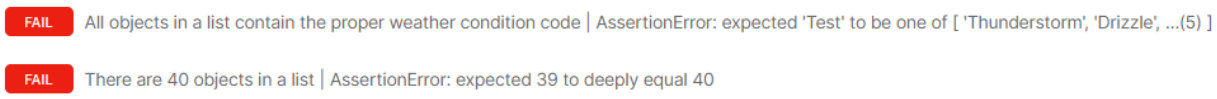
Slika 6.11. Vizualizacija podataka za nedefinirani naziv grada.

Na slici 6.12. vidljiv je dio vizualizacije podataka za Pariz za narednih 5 dana. Također, na isti način su vizualizirani podaci i za ostale gradove u svijetu.



Slika 6.12. Vizualizacija podataka za vremensku prognozu za 5 dana.

Napravljeni su i lažni testni scenariji kako bi se simuliralo ponašanje testnih slučajeva u slučaju dobivanja nerealnih podataka. Takva vrsta testnih scenarija potrebna je kako bi se provjerila ispravnost testnih slučajeva. Testni slučajevi koji koriste simulirane podatke su pali, kao što je i očekivano, jer su primljeni neispravni podaci. Na slici 6.13. prikazani su rezultati testnih slučajeva koji su u odgovoru dobili neočekivane podatke. Tako je u odgovoru za vremenske uvjete poslana neočekivana vrijednost parametra koja je bila „Test“. Također, u listi je obrisan jedan objekt te lista sada sadrži 39, umjesto 40 objekata.



Slika 6.13. Prikaz testnih slučajeva koji su u odgovoru dobili neočekivane vrijednosti parametara.

6. ZAKLJUČAK

U ovom diplomskom radu cilj je upoznati se s testiranjem API-ja na što više razina. Potrebno upoznati se s pojmom API-ja te shvatiti u kakvom je odnosu s klijentom i poslužiteljem. Također, potrebno je razumjeti što su *endpoint*-i i od kojih su parametara sastavljeni. Nadalje, potrebno je istražiti što se sve može testirati. U ovom radu, prva stavka je testiranje sigurnosti korištenih API-ja. Potrebno je testirati postoji li mogućnost neovlaštenog pristupa privatnim informacijama te izmjene istih. Sigurnost je jedna od ključnih stavki koja treba biti testirana. Uz sigurnost, potrebno je testirati i performanse, odnosno ponašanje API-ja u različitim scenarijima, koliko resursa troši, kakva je responzivnost te može li API podnijeti velik broj korisnika odjednom. Nužno je testirati i same podatke u odgovoru, na što je i stavljen fokus u ovom diplomskom radu. Takav pristup testiranju osigurava pronalaženje i ispravljanje pogrešaka prije nego one prouzroče probleme u konačnom isporučenom proizvodu. Kako bi se napisali dobri i ispravni testni scenariji i testni slučajevi potrebno je detaljno proučiti dokumentaciju za odabrane API-je. Također, potrebno je istražiti očekivane vrijednosti za pojedine parametre kako bi se osiguralo dobivanje realnih podataka. Tako je primjerice osigurano da temperatura ili tlak zraka ne mogu imati nerealne vrijednosti na temelju povijesnih podataka o istima. Prilikom pisanja testnih scenarija potrebno je uzeti u obzir njihovu međusobnu povezanost, ukoliko ona postoji. Takva pojava naziva se API *chaining* i osigurava izbjegavanje pogrešaka zbog neispravnog redoslijeda pozivanja zahtjeva. Svaki testni scenarij sastoji se od testnih slučajeva. Testni slučajevi moraju biti dobro osmišljeni kako bi bili jednostavni za razumjeti. To je jedan od izazova API testiranja. Potrebno je simulirati što više scenarija koji bi se mogli dogoditi u stvarnom svijetu. Svakako, izazov je i detaljno proučiti dohvaćene podatke te provjeriti njihovu ispravnost.

U radu su testirana dva API-ja, GitHub API i Weather API. Prilikom testiranja GitHub API-ja, velik broj testnih scenarija i testnih slučajeva temelji se na testiranju sigurnosti. Validira se autentifikacija te vraćeni podaci kako bi se provjerilo da ne postoji mogućnost neovlaštenog pristupa i izmjene privatnih informacija korisnika. Testira se i vrijeme odziva u svim testnim scenarijima. Pri tome se može zaključiti kako kreiranje resursa zahtijeva veći vremenski period nego dohvaćanje, ažuriranje ili brisanje resursa. Prilikom testiranja Weather API-ja, korištene su samo metode za dohvaćanje podataka. Može se primijetiti kako vrijeme odziva ovisi i o veličini odgovora. Tako dohvaćanje podataka o vremenskoj prognozi za 5 dana traje dvostruko duže nego dohvaćanje podataka o trenutnoj vremenskoj prognozi. Prilikom dohvaćanja podataka preko API-ja, potrebno je osigurati i dobru internetsku konekciju jer vrijeme odziva ovisi i o toj stavci.

Zaključno, prilikom API testiranja potrebno je pokriti različite aspekte testiranja od sigurnosnih, preko performansi do samih informacija u odgovoru. U ovom diplomskom radu ima prostora za nadogradnju testnih slučajeva za performanse sa stajališta dobivanja različitih količina podataka u odgovorima na poslane upite. Također, mogu se testirati performanse prilikom korištenja podataka od strane velikog broja korisnika kako bi se vidjelo ponašanje API-ja u takvim situacijama. Za takvu vrstu testiranja bilo bi potrebno koristiti drugi alat za testiranje, primjerice Apache JMeter.

LITERATURA

[1] Postman Inc., What is an API? [online], Postman Inc., dostupno na: https://www.postman.com/what-is-an-api/ [13.04.2024.]

[2] S. Jonnada, J. K. Joy, Measure your API Complexity and Reliability, 2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA), str. 104, Honolulu, Hawaii, 2019.

[3] Testfully Pty Ltd., 9 HTTP methods and how to use them [online], Testfully Pty Ltd., dostupno na: https://testfully.io/blog/http-methods/ [07.04.2024.]

[4] GeeksForGeeks, API Testing – Software testing [online], GeeksForGeeks, dostupno na: https://www.geeksforgeeks.org/api-testing-software-testing/ [13.04.2024.]

[5] A. Khan, S. Mudassar, Learn API Testing, First Edition, C# Corner

[6] A. S. Gillis, API testing [online], TechTarget, dostupno na: https://www.techtarget.com/searchapparchitecture/definition/API-testing [13.04.2024.]

[7] Katalon, About Katalon Studio [online], Katalon, dostupno na: https://docs.katalon.com/katalon-studio/about-katalon-studio [29.04.2024.]

[8] SmartBear, The World's Most Widely Used Open Source API Testing Tool [online], SmartBear, dostupno na: https://www.soapui.org/open-source/ [29.04.2024.]

[9] Apache Software Foundation, ApacheJMeter [online], Apache Software Foundation, dostupno na: https://jmeter.apache.org/ [29.04.2024.]

[10] Minds Task Knowledge Center, API Testing in the Age of Generative AI: What You Need to Know [online], LinkedIn, dostupno na: https://www.linkedin.com/pulse/api-testing-age-generative-ai-what-you-need-know [07.04.2024.]

[11] Postman Inc., What is Postman? [online], Postman Inc., dostupno na: https://www.postman.com/product/what-is-postman/ [13.04.2024.]

[12] Postman Inc., Tools [online], Postman Inc., dostupno na: https://www.postman.com/product/tools/ [14.04.2024.]

[13] Isha, A.Sharma, M. Revathi, Automated API testing, 2018 3rd International Conference on Inventive Computation Technologies (ICICT), str. 788, Coimbatore, India, 2018.

[14] GitHub Docs, About the REST API [online], GitHub Inc., dostupno na: https://docs.github.com/en/rest/about-the-rest-api/about-the-rest-api?apiVersion=2022-11-28 [17.02.2024.]

[15] OpenWeather, Best way to start and continue calling OpenWeather APIs [online], OpenWeather, dostupno na: https://openweathermap.org/appid [02.03.2024.]

[16] OpenWeather, Geocoding API [online], OpenWeather, dostupno na: https://openweathermap.org/api/geocoding-api [02.03.2024.]

[17] Postman Inc., What are HTTP status codes [online], Postman Inc., dostupno na: https://blog.postman.com/what-are-http-status-codes/ [17.02.2024.]

[18] Postman Inc., Mock your API using saved response examples [online], Postman Inc., dostupno na: https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/mocking-with-examples/ [19.03.2024.]

SAŽETAK

Tema ovog diplomskog rada je API testiranje. U teorijskom dijelu rada opisano je što je API te načini komuniciranja istog s klijentom i poslužiteljem. Opisane su i vrste i načini testiranja te su navedeni alati koji omogućavaju testiranje. U praktičnom dijelu osmišljeni su testni scenariji te napisani testni slučajevi koji provjeravaju ponašanje API-ja u različitim situacijama. Za testiranje je korištena API platforma Postman te dva API-ja koja su testirana na više razina. Implementacija testova realizirana je korištenjem skriptnog jezika JavaScript-a. Nakon testiranja napravljena je analiza rezultata.

Ključne riječi: API, API testiranje, JavaScript, Postman

ABSTRACT

API testing

The topic of this master's thesis is API testing. The theoretical part of the thesis describes what an API is and the ways of its communication with client and server. The thesis also covers different types and methods of testing, as well as the tools used to facilitate the testing process. In the practical part, test scenarios are designed and test cases are written to verify the behavior of the tested APIs in different situations. The Postman API platform is chosen for testing, along with two APIs tested at multiple levels. Test implementation is done using the JavaScript scripting language. After completing the testing phase, the results are thoroughly analyzed.

Keywords: API, API testing, JavaScript, Postman