Sveučilište Jurja Dobrile u Puli

Fakultet informatike

ANDREJ SABOL

Autentikacija i autorizacija pomoću JSON web tokena (JWT)

Diplomski rad

Pula, ___25. rujna__, _2019_ godine.

Sveučilište Jurja Dobrile u Puli

Fakutet informatike

ANDREJ SABOL

Autentikacija i autorizacija pomoću JSON web tokena (JWT)

Diplomski rad

JMBAG: ……., redoviti student

Studijski smjer: Informatika

Predmet: Kriptografija

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: doc. dr. sc. Siniša Miličić

Pula, ___25. rujna__, _2019_ godine.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani ____Andrej Sabol__, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____Sabol Andrej___

Pula, ___25. rujna__, _2019_ godine.

IZJAVA

o korištenju autorskog djela

Ja, _Andrej Sabol_ dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom _Autentikacija i autorizacija pomoću JSON web tokena (JWT)_ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____25. rujna 2019.____

Potpis

___Sabol Andrej___

SADRŽAJ

1. UVOD

JWT ili json web token je kompaktan i samostalan način sigurnog prijenosa informacija. Njihova uloga od početka korištenja bila je fokusirana na korištenje kod procesa autentikacije i autorizacije korisnika prilikom prijava na razne servise poput društvenih mreža, emaila, aplikacije i sl. Jedna od glavnih karakteristika JWT-a su jednostavnost korištenja, sigurnost i popularnost tehnologije.

Tema rada je autentikacija i autorizacija putem json web tokena.

Ciljevi istraživanja su:

- Definiranje pojmova autentikacije i autorizacije,
- Definiranje pojma JWT-a i njegove strukture,
- Ukazivanje na mogućnosti primjena JWT-a,
- Definiranje kriptografskih metoda korištenih JWT-om,
- Ukazivanje prednosti i nedostataka ove tehnologije,
- Praktičnim primjerom prikazati rad JWT-a kod autentikacije i autorizacije.

Za potrebe istraživanja korišteni su dostupni domaći i strani izvori podataka.

Rad se sastoji od pet poglavlja teorijskog sadržaja i dva poglavlja praktičnog sadržaja.

Rad započinje poglavljem o teorijskom okviru autentikacije i autorizacije. Prvo poglavlje sadrži definicije pojmova kao i opis rada sa primjerima.

Drugo poglavlje definira pojam json web tokena. Nakon definiranja pojma JWT-a poglavljem se razrađuje struktura JWT-a kao i njegova primjena. Potpoglavljima su definirani i opisani svi elementi koji grade strukturu JWT-a. Zadnji dio poglavlja spominje kriptografske algoritme korištene kod JWT-a koji su kasnijim poglavljem detaljnije opisani.

Treće poglavlje prikazuje primjere korištenja JWT-a kod autentikacije i autorizacije. Poglavljem su prikazana i opisana četiri koraka autentikacije. Drugim dijelom poglavlja detaljno je opisan rad JWT-a kod osiguravanja podataka korištenjem API-a. Zadnji dio poglavlja prikazuje izgled JWT-a u praksi i njegovu strukturu.

Četvrtim poglavljem obrađene su kriptografske metode JWT-a. Kod razrade ovog poglavlja uglavnom su korišteni RFC-ovi tehnologija kao izvor podataka. Četvrto poglavlje podijeljeno je na sedam potpoglavlja od kojeg svako predstavlja zasebnu kriptografsku metodu odnosno algoritam. Prvo potpoglavlje opisuje rad HMAC algoritma dok drugo i treće potpoglavlje opisuju rad RSA i EDRSA digitalnog potpisa. Četvrto potpoglavlje također opisuje kriptografsku metodu digitalnog potpisa, dok potpoglavlja pet, šest i sedam predstavljaju metodu enkripcije ključa.

Slijedećim poglavljem detaljno su analizirane prednosti i nedostaci korištenja JWT-a. Svaka od navedenih prednosti i nedostataka detaljno je obrazložena sadržanim poglavljem.

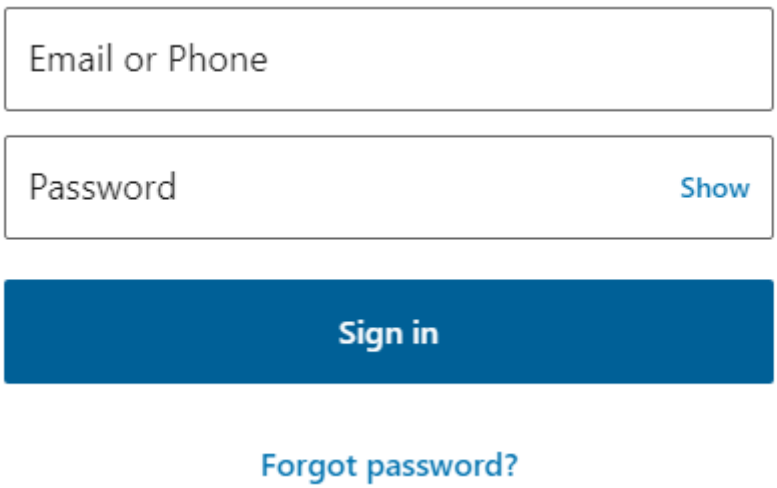
Sedmim poglavljem je opisan praktični dio zadatka ovog rada. Ovo poglavlje opisuje rad kreiranog primjera aplikacije koja predstavlja rad JWT-a kod autentikacije korisnika. Poglavlje sadrži detaljan opis svih programskih jezika, paketa i softvera korištenih kod izrade projektnog dijela. Također, mnogobrojnim slikama prikazan je rad i kod aplikacije.

2. POJAM AUTENTIKACIJE I AUTORIZACIJE

Autentikacija i autorizacija dva su pojma koja su učestalo nejasna kada je riječ o sigurnosti. Iako se koriste zajedno, oni imaju potpuno različite definicije i uloge kod sigurnosti nekog sustava. Pojam autentikacije odnosi se na provjeru identiteta korisnika, odnosno, provjerava da li je korisnik zapravo onaj koji tvrdi da je dok autorizacija provjerava da li taj korisnik ima prava pristupa resursu kojem pokušava pristupiti.[1] Iz navedenog vidimo kako se autorizacija i autentikacija zajedno koriste kod zaštite nekog sustava iako se njihova uloga u provjeri korisnika razlikuje.

2.1. Autentikacija

Autentikacija je postupak provjere korisnikovih akreditacija kako bi se utvrdilo da je korisnik zapravo onaj kojim se predstavlja. Najčešći oblik autentikacije je provjera korisnikovog imena i lozinke. Taj oblik se učestalo susreće kod prijave korisnika na razne društvene mreže, forume, aplikacije i sl. Jedan od takvih primjera je forma za prijavu na društvenu mrežu LinkedIn koja je prikazana slikom 1.



Slika 1. LinkedIn forma za prijavu korisnika

Izvor: LinkedIn https://www.linkedin.com/m/login/ (23.4.2019.)

[1] Siddiqui A., „Authentication vs Authorization“, 2018, https://medium.com/datadriveninvestor/authentication-vs-authorization-716fea914d55 (23.4.2019.)

Kod autentikacije se susrećemo s pojmom faktora autentikacije odnosno brojem provjera koje korisnik mora zadovoljiti kako bi potvrdio svoj identitet. Bazirano na stupnju zaštite autentikacija se dijeli na tri glavne skupine:

1. Jednofaktorna autentikacija,
2. Dvofaktorna autentikacija,
3. Višefaktorna autentikacija.

Jednofaktorna autentikacija je najjednostavnija forma autentikacije. Ona se bazira na provjeri korisničke lozinke u paru s korisničkim imenom. Ovaj stupanj autentikacije učestalo se koristi kod prijave korisnika na društvene mreže poput LinkedIna, Facebooka i dr. Takav primjer autentikacije bio je prikazan slikom 1 na prethodnoj stranici. Nešto sigurniji oblik autentikacije je dvofaktorna autentikacija, koja uz korisničko ime i lozinku zahtjeva dodatnu informaciju koju samo korisnik može znati. Najčešće korišteni oblik dvofaktorne autentikacije uz prijavu putem korisničkog imena i lozinke upotrebljava i token odnosno jednokratnu lozinku koju korisnik posjeduje. Primjer upotrebe dvofaktorne autentikacije je kod prijave na korisnički račun putem korisničkog imena i lozinke te upotrebe jednokratne lozinke koja se prilikom prijave može zaprimiti na e-mail račun ili u obliku SMS poruke na mobilni uređaj. Time se dodatno osigurava identitet korisnika koji se prijavljuje. Kod zaprimanja ili generiranja tokena odnosno jednokratne lozinke, razlikujemo dva modela: OOB (Out-of-Band) i non-OOB. Out-of-Band znači da se token generira ili zaprima putem različitog kanala u odnosu na kanal na kojem će se token koristiti dok non-OOB model koristi isti uređaj za generiranje ili zaprimanje tokena. Kao primjer OOB modela može se navesti prijava na e-mail putem osobnog računala gdje se token za prijavu zaprima u obliku SMS poruke na mobilni uređaj, dok kod non-OOB modela taj isti token bi se zaprimao na tom osobnom računalu.[2] Zadnji i najsigurniji oblik autentikacije je višefaktorna autentikacija koja koristi dvije ili više provjere identiteta korisnika koje su međusobno nevezane. Ovaj stupanj autentikacije često se koristi kod financijskih organizacija zbog dodatnog stupnja sigurnosti. Za potvrdu višefaktorne autentikacije najčešće se koriste biometrijske provjere kao što su otisak prsta, prepoznavanje lica, govor ili skeniranje

[2] Dias R., „The 5 factors of authentication“, 2017, https://medium.com/@renansdias/the-5-factors-of-authentication-bcb79d354c13 (23.4.2019.)

oka.[3] Mnogi moderni pametni telefoni dolaze s unaprijed ugrađenim čitačem otiska prsta ili funkcijom prepoznavanja lica te koriste višefaktornu autentikaciju za pristup uređaju. Uz višefaktornu autentikaciju razlikujemo i više stupanjsku autentikaciju, odnosno autentikaciju u više koraka. Primjer autentikacije u više koraka bila bi validacija korisničkog imena i lozinke. Tek kada su ta dva uvjeta zadovoljena, korisnik dobiva jednokratnu lozinku putem maila ili SMS poruke. Po pitanju sigurnosti ovaj tip autentikacije nije idealan jer potencijalnom napadaču otkriva ispravnost korisničkog imena i lozinke tako da prolazi validaciju i pruža token. Također ovaj tip autentikacije validacijom korisničkog imena i lozinke postaje jednofaktorna autentikacija koja zahtjeva samo unos jednokratne lozinke.[4]

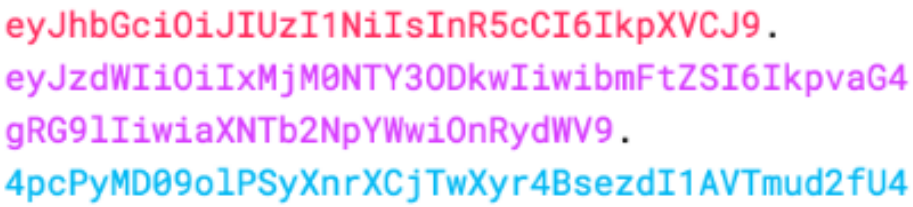
2.2. Autorizacija

Nakon uspješne validacije korisničkih podataka slijedi autorizacija. Autorizacija je postupak provjere korisnikovih prava pristupa, odnosno, putem autorizacije provjerava se da li korisnik ima prava pristupa određenom resursu npr. bazi podataka, informacijama, određenom dijelu aplikacije i sl. Postupak autorizacije učestalo se izvršava nakon autentikacije korisnika. Važno je napomenuti kako se autentikacija i autorizacije učestalo koriste zajedno kako bi se osigurao siguran rad sustava. Svaki pokušaj pristupa od strane korisnika koji je unio ispravnu korisničku oznaku i lozinku, ali nema prava pristupa, sustav će blokirati.[5] Primjer autorizacije odnosno kontrole pristupa mogu biti stranice nekog sveučilišta gdje se razlikuju prava student-korisnika i prava profesor-korisnika. Student-korisnik može imati prava pristupiti sadržaju predavanja, odnosno ima read only prava, dok profesor-korisnik može uz pristupanje sadržaju taj sadržaj i uređivati, odnosno, ima read and write prava.

[3] Khillar S., „Difference between Authentication and Authorization“, 2017, http://www.differencebetween.net/technology/difference-between-authentication-and-authorization/ (23.4.2019.)
[4] Dias Renan, op.cit. (30.4.2019.)
[5] Khillar S., op. cit. (30.4.2019.)

3. DEFINICIJA JWT-A

Kako bismo pravilno definirali JWT potrebno je prvo definirati što je RFC. RFC odnosno Request for Comment je formalni dokument kojeg izrađuje IETF (Internet Engineering Task Force) koji opisuje specifikacije određene tehnologije. Kada je RFC potvrđen on postaje službeni dokument o standardima tehnologije koju opisuje.[6] Neki od značajnijih i poznatijih RFC-ova su RFC 791 koji je nastao 1981. godine i definira internet protokol, RFC 1034 naslova „Domain Names – concepts and facilities" i RFC 793 odnosno Transmission Control Protocol RFC.[7] JWT ili JSON Web Token je otvoreni standard koji je definiran 2015. godine RFC-om 7519 u kojem je definirano kako je JWT kompaktan i samostalan način sigurnog prijenosa informacija u obliku JSON objekta čija je struktura također definirana RFC-om 7159.[8] Primjer jednog JWT-a prikazan je slikom 2.

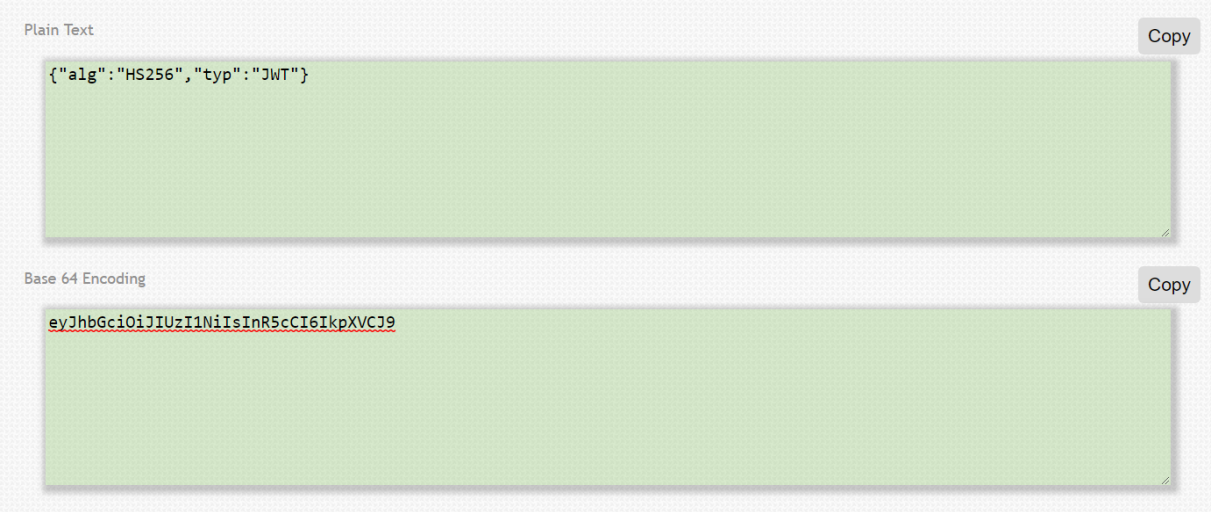


Slika 2. Šifriran oblik JWT-a

Izvor: JWT https://jwt.io/introduction/ (22.5.2019.)

Primjerom šifriranog oblika JWT-a sa slike 2 možemo vidjeti kako je struktura JWT-a zapravo jedan string odvojen dvjema točkama koje dijele taj string na 3 djela, odnosno 3 elementa koji grade strukturu JWT-a. Ti elementi koji sačinjavaju strukturu JWT-a su zaglavlje (eng. „Header"), sadržaj (eng. „Payload") i potpis (eng. „Signature"). Šifrirani oblik JWT-a ljudima izgleda kao besmisleni string, no kada bismo uzeli prvi dio tog stringa, odnosno zaglavlje JWT-a i base64url-dekodirali ga taj string bi poprimio oblik prikazan slikom 3.

[6] Request for Comment (RFC) https://www.techopedia.com/definition/27929/request-for-comments-rfc (20.5.2019.)
[7] What is an RFC?, 2018. https://flaviocopes.com/rfc/ (20.5.2019.)
[8] JSON Web Token (JWT) RFC 7519 https://tools.ietf.org/html/rfc7519 (22.5.2019.)



Slika 3. Base64url dekodirano zaglavlje JWT-a

Izvor: http://www.base64url.com/ (22.5.2019)

Takav oblik predstavlja JSON objekt koji se sastoji od para imena i vrijednosti.

3.1. Zaglavlje (JOSE Header)

JOSE (Javascript Object Signing and Encryption) zaglavlje kod JWT-a opisuje kriptografske operacije koje će se primjenjivati te opcionalno opisuje dodatna svojstva JWT-a u kojem se zaglavlje nalazi. JWT je poput apstraktne klase koja ne postoji sama, on mora bit JWS (JSON Web Signature) ili JWE (JSON Web Encryption) koji predstavljaju konkretnu implementaciju JWT-a.[9] Zaglavlje JWT-a tipično se sastoji od 2 para imena i vrijednosti. Prvi par predstavlja tip objekta i označava se imenom „typ“ dok je vrijednost tog para „JWT“. Kod JSON objekta to označava kako se radi o JSON web tokenu. Drugi par je imenovan „alg“ i označuje koji algoritam heširanja (eng. „hashing algorithm“) će biti korišten za izradu JWT komponente potpisa.[10] U primjeru prikazanom slikom 3 radi se o HMAC-SHA256 algoritmu. Heširanje algoritma nije nužno te vrijednosni par može biti postavljen na vrijednost „none“, koja označuje kako

[9] Siriwardena P., JWT,JWS and JWE for Not So Dummies! (part I), 2016. https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3 (22.6.2019.)

[10] Stecky-Efantis M., 5 Easy Steps to Understanding JSON Web Tokens (JWT), 2016. https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec (22.5.2019.)

se radi o nezaštićenom JWT-u. Uz „typ“ i „alg“ kod zaglavlja JWT-a još učestalo susrećemo parametar „cty“ (Content type) koji predstavlja informacije o strukturi JWT-a. Kod strukture zaglavlja postoje i drugi zahtjevi koji su opcionalni kod kreacije zaglavlja poput „iss“ (Issuer), „aud“ (Audience) i „sub“ (Subjekt) zahtjeva.

3.2. Sadržaj (Payload)

Sadržaj ili payload drugi je dio stringa šifriranog oblika JWT-a. On predstavlja set zahtjeva koje želimo prenijeti našim tokenom. Slikom 4 prikazani su zahtjevi sadržaja jednog JWT-a.


```
{
   "iss":"accounts.google.com",
   "sub":"110502251158920147732",
   "azp":"825249835659-np4sqv7erhu1211s.apps.googleusercontent.com",
   "email":"prabath@wso2.com",
   "at_hash":"zf86vNulsLB8gFaqRwdzYg",
   "email_verified":true,
   "aud":"825249835659-np4sqv7erhu1211s.apps.googleusercontent.com",
   "hd":"wso2.com",
   "iat":1401908271,
   "exp":1401912171
}
```


Slika 4. Primjer sadržaja JWT-a

Izvor: https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3 (23.6.2019)

Zahtjevi u JWT-u dijele se u 3 kategorije, a to su registrirani zahtjevi, javni zahtjevi i privatni zahtjevi. Registrirani zahtjevi su definirani u RFC-u JWT-a gdje se nalazi registar definiranih zahtjeva. Ti zahtjevi nisu obavezni, ali su definirani kako bi se olakšala upotreba i definirao set korisnih zahtjeva kod izrade JWT-a. Ti zahtjevi uz već prethodno spomenuti „iss“ (Issuer), „sub“ (Subjekt) i „aud“ (Audience) su „exp“ (Expiration Time), „nbf“ (Not Before), „iat“ (Issued At) i „jti“ (JWT ID). Svaki od unaprijed određenih zahtjeva definira pravila upotrebe JWT-a u čijem sadržaju je definiran, npr. putem „exp“ zahtjeva određuje se vrijeme kada JWT više nije valjan i nije siguran za

prihvaćanje, dok „nbf“ zahtjev određuje vrijeme prije kojeg se JWT ne smije prihvatiti za procesiranje. Zahtjevi mogu biti i svojevoljno definirani od strane korisnika JWT-a u obliku javnih zahtjeva. Takvi zahtjevi moraju biti registrirani u IANA (Internet Assigned Numbers Authority) „JSON Web Token Claims“ registru ili biti javno imenovani imenom koje se ne krši ograničenim imenima JWT-a. Neki primjeri takvih zahtjeva su „email“ koji predstavlja e-mail adresu, „picture“ odnosno profilna slika, „zoneinfo“ koja predstavlja vremensku zonu i mnogi drugi koji su prikazani slikom 5.

name	Full name	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
given_name	Given name(s) or first name(s)	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
family_name	Surname(s) or last name(s)	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
middle_name	Middle name(s)	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
nickname	Casual name	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
preferred_username	Shorthand name by which the End-User wishes to be referred to	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
profile	Profile page URL	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
picture	Profile picture URL	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
website	Web page or blog URL	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
email	Preferred e-mail address	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
email_verified	True if the e-mail address has been verified; otherwise false	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
gender	Gender	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
birthdate	Birthday	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
zoneinfo	Time zone	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
locale	Locale	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
phone_number	Preferred telephone number	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
phone_number_verified	True if the phone number has been verified; otherwise false	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
address	Preferred postal address	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
updated_at	Time the information was last updated	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
azp	Authorized party - the party to which the ID Token was issued	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 2]
nonce	Value used to associate a Client session with an ID Token	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 2]
auth_time	Time when the authentication occurred	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 2]
at_hash	Access Token hash value	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 2]
c_hash	Code hash value	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 3.3.2.11]
acr	Authentication Context Class Reference	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 2]
amr	Authentication Methods References	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 2]
sub_jwk	Public key used to check the signature of an ID Token	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 7.4]

Slika 5. Popis javno registriranih zahtjeva JWT-a

Izvor: https://www.iana.org/assignments/jwt/jwt.xhtml (22.5.2019)

Zadnji tip zahtjeva su privatni zahtjevi, odnosno zahtjevi koji su definirani između stranaka koje su se usuglasile na korištenje tih zahtjeva. Za privatne zahtjeve nije potrebno da budu registrirani i nisu potrebni pratiti regulaciju imena kako se ne bi kršili s ograničenim imenima.[11]

[11] Introduction to JSON Web Tokens https://jwt.io/introduction/ (22.6.2019)

3.3. Potpis (Signature)

Signature ili potpis treći je dio odvojen točkom šifriranog oblika JWT stringa. Potpis JWT-a sačinjavaju tri komponente:

1. Zaglavlje (Header),
2. Sadržaj (Payload),
3. Tajna (Secret).

Potpis JWT-a kreira se spajanjem kodiranog oblika zaglavlja i kodiranog oblika sadržaja te hashiranje korištenjem tajne. Zaglavlje i sadržaj se kodiraju navedenim algoritmom korištenim u zaglavlju JWT-a. U primjeru sa slike 3 koristi se HMAC-SHA256 algoritam. Slikom 6 prikazan je pseudokod dobivanja potpisa JWT-a.

```
// signature algorithm

data = base64urlEncode( header ) + "." + base64urlEncode( payload )

hashedData = hash( data, secret )

signature = base64urlEncode( hashedData )
```

Slika 6. Pseudokod procesa dobivanja potpisa JWT-a

Izvor: https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec (23.6.2019)

Slikom 6 iz pseudokoda procesa dobivanja potpisa možemo vidjeti kako proces započinje base64url šifriranjem zaglavlja i sadržaja, koji se zatim spajaju u string odvojen točkom i spremaju u varijablu data. Zatim se koristi varijabla data i tajni ključ u hash algoritmu, koji je u primjeru slike 3 HMAC-SHA256 algoritam i taj string se sprema u varijablu zvanu hashedData. Na kraju za dobivanje samog potpisa varijabla

hashedData se base64url šifrira i sprema u varijablu nazivom signature.[12] Kod JWT-a razlikujemo simetrični i asimetrični potpis.

Kod simetričnog potpisa servis generira jedan ključ koji se koristi kod izrade JWT-a kao u primjeru prikazanom slikom 6. Kod zaprimanja JWT-a sustav treba potvrditi integritet prije nego se koriste podaci JWT-a. Korištenjem istog ključa koji je korišten prilikom izrade JWT-a sustav izračunava algoritam koji je korišten prilikom šifriranja JWT-a – u ovom slučaju radi se o HMAC algoritmu. Primjer rada simetričnog potpisa prikazan je slikom 7.

Slika 7. Primjer korištenja simetričnog potpisa

Izvor: https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nobody-talks-about.html (23.6.2019)

Ukoliko se HMAC poklapa s onim iz potpisa radi se o pozitivnoj potvrdi integriteta. Simetrični potpis je poprilično jednostavan te se učestalo koristi kao primjer kod učenja JWT-a, no simetrični potpis sprječava dijeljenje JWT-a s drugim servisima zbog problema dijeljenja jednog ključa. Zbog tog problema koristi se asimetrični potpis. Kod asimetričnog potpisa koristi se par javnog i privatnog ključa. Potpis se generira korištenjem privatnog ključa koji je poznat samo izdavatelju JWT-a, dok se verifikacija vrši korištenjem javnog ključa koji je dostupan za dijeljenje s drugim servisima i strankama.[13] Takav sustav prikazan je slikom 8.

[12] Stecky-Efantis M., op. cit (23.6.2019)
[13] Ryck De P., The hard parts of JWT security nobody talks about, 2019. https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nobody-talks-about.html (24.6.2019)

Slika 8. Primjer korištenja asimetričnog potpisa

Izvor: https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nobody-talks-about.html (23.6.2019)

Ova shema značajno smanjuje rizik kršenja jedne od usluga u arhitekturi. Jedan od najvažnijih i najzahtjevnijih aspekata rada s JWT-om je upravljanje ključevima. Kriptografski ključevi korišteni kod potpisa JWT-a trebaju biti učestalo rotirani i izmjenjivani kako bi se uspješno održavala sigurnost sustava. Proces upravljanja ključevima je od iznimne važnosti kada je riječ o sigurnosti sustava no nije jednostavan problem za riješiti. U modernim kriptografskim sustavima ključ je najslabija karika sigurnosti tog sustava. Kod kriptografije razlikujemo tri primarna tipa ključeva:

1. Simetrični ključ,
2. Privatni ključ,
3. Hash ključ.

Simetrični se ključevi koriste kod šifriranja velikih količina podataka uz pomoć simetričnih algoritama poput 3DES (Triple Data Encryption Algorithm) ili AES (Advanced Encryption Standard). U simetričnom kriptografskom sustavu svatko tko ima ključ može dešifrirati podatke. Privatni ključevi su tajna polovica javnog i privatnog para ključeva koji se koriste kod asimetričnih kriptografskih sustava. Takvi sustavi koriste algoritme poput RSA (Rivest-Shamir-Adleman) i ECDSA (Elliptic Curve Digital Signature Algorithm). Svatko tko posjeduje privatni ključ može se predstaviti kao vlasnik tog ključa te ima pristup dešifriranju podataka. Zadnji primarni tip ključeva su hash ključevi koji koriste algoritme poput HMAC-SHA256 (keyed-hash message authentication code / Secure Hash Algorithm). Kod hash ključeva svatko tko posjeduje tajni ključ može promijeniti originalne podatke. Neke od slabosti kriptografskih sustava uključuju slabe ključeve, nepravilnu zaštitu ključeva, ponovno korištenje i ne rotiranje

ključeva i dr. Slabost ključa može se izbjeći korištenjem certificiranih generatora nasumičnih brojeva (eng. Random Number Generator) koji koriste prikladnu entropiju iz hardverskog izvora buke. Intervalnom rotacijom ključa kojom podrazumijevamo obnavljanje i izmjenu ključa osiguravamo problem prekorištenja istog ključa.[14]

[14] Stubbs R., Cryptographic Key Management – the Risks and Mitigation, 2018. https://www.cryptomathic.com/news-events/blog/cryptographic-key-management-the-risks-and-mitigations (24.6.2019)

4. UPOTREBA JWT-A KOD AUTORIZACIJE I AUTENTIKACIJE

Autorizacija i autentikacija česti su pojmovi koji se spominju zajedno u kontekstu s JWT-om. Zapravo JWT je izvrsna tehnologija za osiguravanje ispravnosti podataka, usto može sadržati velike količine podataka i kriptografski je potpisan te se smatra kriptografski sigurnim. Zbog toga JWT se učestalo koristi kod postupka autorizacije i autentikacije – npr. za API autentikaciju ili server-to-server autorizaciju. Ovim poglavljem bit će prikazane i detaljno objašnjene neke od upotreba JWT-a kod autentikacije i autorizacije.

4.1. Upotreba JWT-a kod autentikacije

Proces autentikacije kod JWT-a može biti razlomljen na četiri koraka:

1. Validacija korisnika na temelju baze podataka i generiranje zahtjeva („Claim“) baziranog na korisnikovoj ulozi,
2. Sadržaj („Payload“) koji sadrži tvrdnje ili druge podatke vezane za korisnika potpisuje se ključem za generiranje tokena i vraća se korisniku,
3. Korisnik šalje token sa svakim zahtjevom, obično u zaglavlju („Header“) ili putem kolačića (eng. Cookies) te se zaprimljeni token dešifrira za potvrdu zahtjeva,
4. Završetak identifikacije korisnika kojem je odobren pristup resursnom poslužitelju ovisno o njegovim zahtjevima.[15]

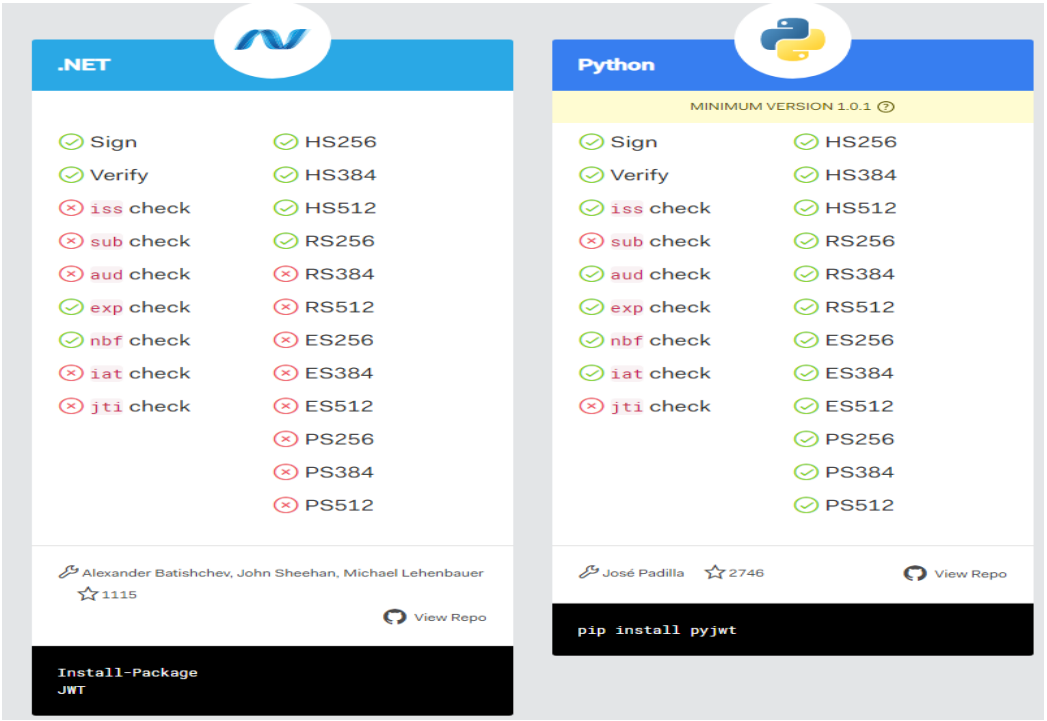
Prednost paradigme za provjeru autentičnosti temeljene na tokenu je ta da umjesto pohrane informacija potrebnih za provjeru autentičnosti ili autorizacije svakog korisnika u sesiji (eng. Session) se generira jedan ključ za potpisivanje koji se pohranjuje na autorizacijski poslužitelj. Korištenje jednog ključa za dešifriranje tokena i identifikaciju korisnika, bez obzira na broj korisnika, osigurava uštedu memorije i pridonosi skalabilnosti. Također postupak autorizacije može biti delegiran na bilo koji poslužitelj što ga čini potpuno odvojenim. Korisnici se identificiraju potvrđivanjem zahtjeva generiranih prvim korakom, ovisno o korisnikovim dopuštenjima. Tim zahtjevima se

[15] Singh A., JSON Web Token (JWT) with Web API, 2017., http://blogs.quovantis.com/json-web-token-jwt-with-web-api/ (09.7.2019.)

može vjerovati jer ih poslužitelj generira prvim korakom, te digitalno potpisuje koristeći jedan od algoritama poput HMAC SHA256, što ujedno osigurava da se prava ili zahtjevi nisu mijenjali u međuvremenu.[16]

4.2. Upotreba JWT-a kod API

Postoje mnogobrojne knjižnice (eng. Libraries) koje se brinu oko postupka generiranja i verificiranja tokena u gotovo svim programskim jezicima. Neki od učestalo korištenih su .NET, Python, Node.js, Java, JavaScript, Ruby, Go, C++, PHP, Swift i mnogi drugi.[17] Na službenim stranicama JWT-a nalazi se kompletan popis knjižnica u svim dostupnim programskim jezicima upotpunjen instrukcijama upotrebe i podrškom knjižnice. Primjerom slike 9 prikazane su dvije knjižnice s popisa, jedna u programskom jeziku Python, a druga u .NET-u.

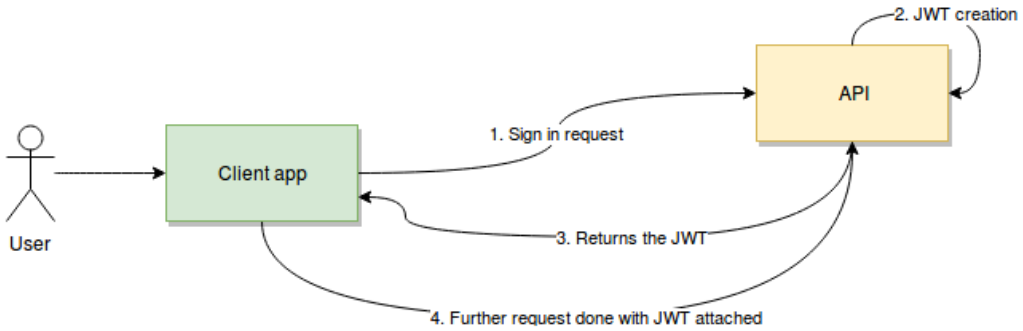


Slika 9. Popis knjižnica za generiranje i verifikaciju tokena

Izvor: https://jwt.io/ (10.7.2019.)

[16] Singh A., op.cit. (09.7.2019.)
[17] Libraries for Token Signing/Verification, https://jwt.io/ (10.7.2019.)

Također postupak generiranja i validacije tokena može se izvršavati bez pomoći knjižnica, te postoje mnogobrojni blogovi i tutorijali koji sadrže detaljan opis svakog koraka ovog postupka. Postupak korištenja JWT-a za zaštitu podataka kod API-a prikazan je slikom 10. Slika prikazuje klijent-server interakciju koja se može svesti na četiri jednostavna koraka.



Slika 10. Četiri koraka upotrebe JWT-a kod API-a

Izvor: https://blog.logrocket.com/how-to-secure-a-rest-api-using-jwt-7efd83e71432/ (10.7.2019.)

Prvim korakom aplikacija šalje zahtjev za prijavu kojim su sadržani korisnički podaci potrebni za prijavu kao što su korisnička oznaka i zaporka ili neki drugi oblik autentikacije. Nakon provjere ispravnosti podataka API kreira JWT i koristi tajni ključ za potpis JWT-a. Trećim korakom API vraća potpisani JWT klijentskoj aplikaciji. Naposljetku klijentska aplikacija zaprima token, potvrđuje autentičnost tokena i koristi ga za sve daljnje zahtjeve bez potrebe ponovnog slanja akreditiva. Taj postupak može biti prikazan i pravim primjerom jednostavne aplikacije koja služi za isplatu plaća zaposlenika. Za potrebe primjera aplikacije određeno je kako postoje dva tipa korisnika, običan korisnik koji ima samo privilegije čitanja podataka (read-only) i admin korisnik, odnosno korisnik koji može čitati i uređivati podatke. Iz prethodnog primjera može se vidjeti kako upotreba JWT-a kod API-a započinje unosom korisničkih podataka u svrhu prijave korisnika[18]. Na pravom primjeru aplikacije to bi izgledalo poput primjera sa slike 11.

[18] Doglio F., How to secure a REST API using JWT, 2019., https://blog.logrocket.com/how-to-secure-a-rest-api-using-jwt-7efd83e71432/ (10.7.2019.)

```
POST /api/users-sessions
```

Payload:

```
{

“Username”: “fernando”

“Password”: “fernando123”

}
```

Slika 11. Zahtjev za prijavu korisnika

Izvor: https://blog.logrocket.com/how-to-secure-a-rest-api-using-jwt-7efd83e71432/ (10.7.2019.)

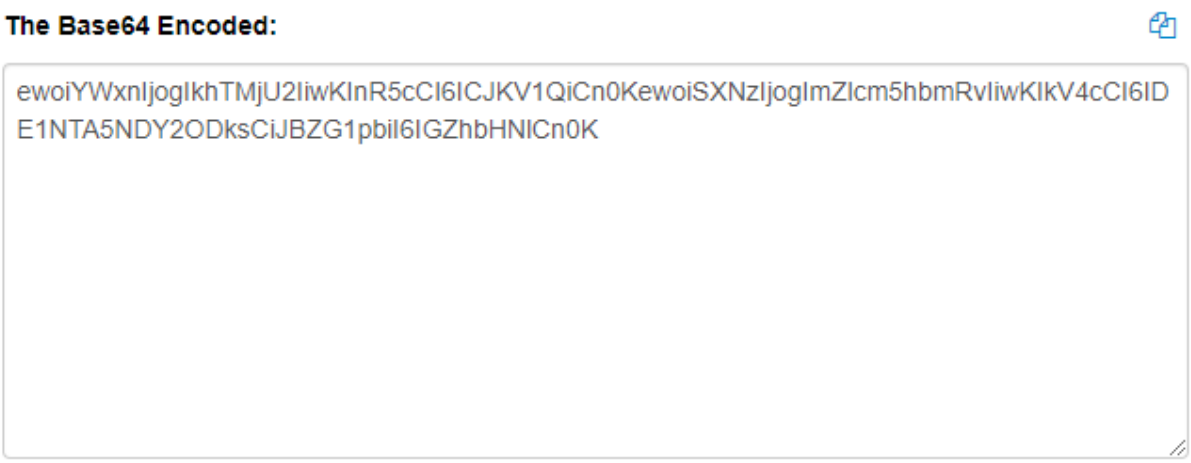
Slikom 11 prikazan je zahtjev nad API-em i payload koji klijentska strana aplikacije šalje koji sadrži korisničko ime i lozinku. Pretpostavkom da su akreditacijski podaci ispravni, sistem bi vratio novi JWT koji bi izgledao poput primjera prikazanog slikom 12.

```
Header:

{
“alg”: “HS256”,
“typ”: “JWT”
}

Payload:

{
“Iss”: “fernando”
“Exp”: 1550946689,
“Admin”: false
}
```

Slika 12. Zaglavlje i sadržaj JWT-a

Izvor: https://blog.logrocket.com/how-to-secure-a-rest-api-using-jwt-7efd83e71432/ (10.7.2019.)

Iz primjera aplikacije sa slike 12 prikazano je zaglavlje i sadržaj vraćenog JWT-a. U zaglavlju JWT-a koristi se HS256 algoritam odnosno Hash-based Message Authentication Code (HMAC) koji koristi SHA-256 funkciju hashiranja za šifriranje sadržaja. HS256 koristi istu tajnu za šifriranje i dešifriranje podataka. Sadržaj vraćenog JWT-a može izgledati i drugačije, odnosno ovisno o potrebama aplikacije sadržaj se može mijenjati. U zadanom primjeru sadržaj JWT-a sadrži tri zahtjeva: „Iss“, „Exp“ i „Admin“. Issuer zahtjev predstavlja korisničko ime korisnika koji se prijavio u sustav. Ovaj podatak može biti kasnije koristan za prikaz korisnikovog imena u korisničkom sučelju. Expiration Time zahtjev predstavlja vrijeme kada će vraćeni token prestati biti važeći, što u ovom slučaju predstavlja 8 sati. I naposljetku Admin zahtjev je zahtjev tipa boolean koji vraća 0 ili 1 odnosno true ili false ovisno o korisnikovom statusu prava. Admin zahtjev nam također može biti veoma koristan kasnije kod korisničkog sučelja za određivanje statusa prikaza i sakrivanja određenih elemenata sučelja. Sljedeći korak je šifriranje podataka i potpisivanje kako bismo dobili pravi token. Za zadani primjer tajna je „A secret API example“ i kada se šifrira zaglavlje i sadržaj dobiju se slijedeći kodirani stringovi prikazani slikom 13.[19]

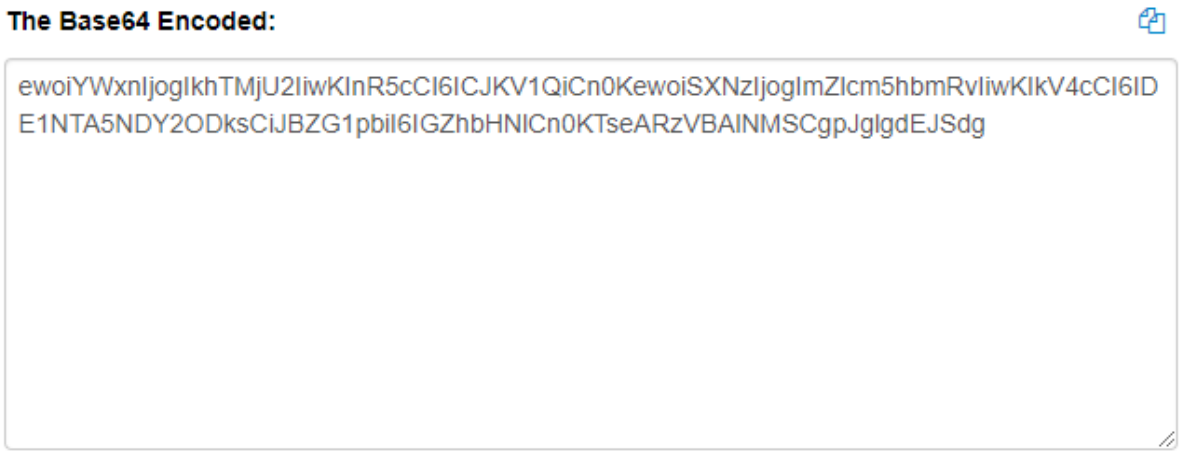


Slika 13. Base64 kodirani oblik zaglavlja i sadržaja JWT-a

Izvor: https://codebeautify.org/base64-encode (17.9.2019.)

[19] Doglio F, op.cit., (10.7.2019.)

Završna verzija JWT-a vraćenog od strane API-a iz zadanog primjera izgleda poput primjera slike 14.



Slika 14. Finalna verzija kodiranog JWT-a

Izvor: https://codebeautify.org/base64-encode (17.9.2019.)

Klijentska strana aplikacije koja zaprima taj JWT može isti i dešifrirati korištenjem istog tajnog ključa koji je bio korišten i kod šifriranja. Svaki daljnji zahtjev koji klijent šalje sadržavat će isti token koji će biti potvrđen od strane poslužitelja ponovnim potpisivanjem. Svaka promjena zahtjeva tokena rezultirala bi promjenom sadržaja, te se potpis ne bi poklapao s originalnim što dokazuje da je došlo do neovlaštene promjene. Ovaj primjer upotrebe JWT-a kod API-a prikazuje njegovu upotrebu kod procesa autentikacije i autorizacije korisnika. Primjerom prikazana aplikacija koristi proces prijave korisnika koji putem podataka kao što su korisničko ime i korisnička lozinka validira tvrdnju autentikacije, odnosno ispravnim podacima potvrđuje o kojem se korisniku radi. Vraćenim JWT-om to je prikazano tvrdnjom „Iss“ odnosno Issuer. Također primjer aplikacije prikazuje i proces autorizacije korisnika što je vidno iz zahtjeva „Admin“ koji predstavlja stupanj korisnikovih prava. Pomoću Admin zahtjeva aplikacija određuje elemente kojima korisnik može upravljati ili vidjeti ukoliko ima read-only prava.

5. KRIPTOGRAFSKE METODE JWT-A

JWT može biti korišten i kada nije kriptografski zaštićen tj. zahtjev zaglavlja „alg“ nije obavezan i može se postaviti na vrijednost „none“, no njegov puni potencijal iskoristivosti dolazi baš kod upotrebe kriptografskih algoritama za zaštitu podataka sadržanih JWT-om. Upotreba kriptografske metode odnosno algoritma za digitalni potpis JWT-a određena je zahtjevom zaglavlja „alg“. RFC-om 7518 naziva „JSON Web Algorithms (JWA)“ sadržan je registar vrijednosti odnosno parametara korištenih kod „alg“ zahtjeva zaglavlja JWS-a i JWE-a[20]. Tablicom 1 prikazana je tablica registriranih algoritama korištenih kod zahtjeva zaglavlja JWS-a, dok su tablicom 2 prikazani algoritami za upravljanje ključevima JWE-a.

Tablica 1. Kriptografski algoritmi korišteni s JWS-om

Vrijednost parametra „alg“	**Digitalni potpis ili MAC algoritam**	**Zahtjevi za provedbom**
HS256	HMAC using SHA-256	Potrebno
HS384	HMAC using SHA-384	Opcionalno
HS512	HMAC using SHA-512	Opcionalno
RS256	RSASSA_PCKS1-v1_5 using SHA-256	Preporučeno
RS384	RSASSA_PCKS1-v1_5 using SHA-384	Opcionalno
RS512	RSASSA_PCKS1-v1_5 using SHA-512	Opcionalno
ES256	ECDSA using P-256 and SHA-256	Preporučeno
ES384	ECDSA using P-384 and SHA-384	Opcionalno
ES512	ECDSA using P-512 and SHA-512	Opcionalno
PS256	RSASSA-PSS using SHA-256 and MGF1 With SHA-256	Opcionalno
PS384	RSASSA-PSS using SHA-384 and MGF1 With SHA-384	Opcionalno

[20] RFC 7518 - JSON Web Algorithms (JWA),2015., https://tools.ietf.org/html/rfc7518#section-3 (17.7.2019.)

PS512	RSASSA-PSS using SHA-512 and MGF1 With SHA-512	Opcionalno
none	No digital signature or MAC performed	Opcionalno

Izvor: https://tools.ietf.org/html/rfc7518#section-3 (17.7.2019.)

Tablica 2. Kriptografskih algoritmi korišteni s JWE-om

Vrijednost parametra „alg“	**Algoritam upravljanja ključem**	**Parametri zaglavlja**	**Zahtjevi za provedbom**
RSA1_5	RSA-PKCS1-v1_5	(bez)	Preporučeno-
RSA-OAEP	RSAES OAEP using default parameters	(bez)	Preporučeno +
RSA-OAEP-256	RSAES OAEP using SHA-256 and MGF1 with SHA-256	(bez)	Opcionalno
A128KW	AES Key wrap with default initial value using 128-bit key	(bez)	Preporučeno
A192KW	AES Key wrap with default initial value using 192-bit key	(bez)	Opcionalno
A265KW	AES Key wrap with default initial value using 256-bit key	(bez)	Preporučeno
dir	Direkt use of a shared symmetric key as the CEK	(bez)	Preporučeno
ECDH-ES	Elliptic Curve Diffie-Hellmab Ephemeral Static key agreement using Concat KDF	„epk“,“apu“,“apv“	Preporučeno +
ECDH-ES+A128KW	Elliptic Curve Diffie-Hellmab Ephemeral Static key agreement using Concat KDF and CEK wrapped with „A128KW“	„epk“,“apu“,“apv“	Preporučeno

ECDH-ES+A192KW	Elliptic Curve Diffie-Hellmab Ephemeral Static key agreement using Concat KDF and CEK wrapped with „A192KW“	„epk“,“apu“,“apv“	Opcionalno
ECDH-ES+A256KW	Elliptic Curve Diffie-Hellmab Ephemeral Static key agreement using Concat KDF and CEK wrapped with „A256KW“	„epk“,“apu“,“apv“	Preporučeno
A128GCMKW	Key wrapping with AES GCM using 128-bit key	„iv“, „tag“	Opcionalno
A192GCMKW	Key wrapping with AES GCM using 192-bit key	„iv“, „tag“	Opcionalno
A256GCMKW	Key wrapping with AES GCM using 256-bit key	„iv“, „tag“	Opcionalno
PBES2-HS256-A128KW	PBE with HMAC SHA-256 and „A128KW“ wrapping	„p2s“, „p2c“	Opcionalno
PBES2-HS384-A192KW	PBE with HMAC SHA-384 and „A192KW“ wrapping	„p2s“, „p2c“	Opcionalno
PBES2-HS512-A256KW	PBE with HMAC SHA-512 and „A256KW“ wrapping	„p2s“, „p2c“	Opcionalno

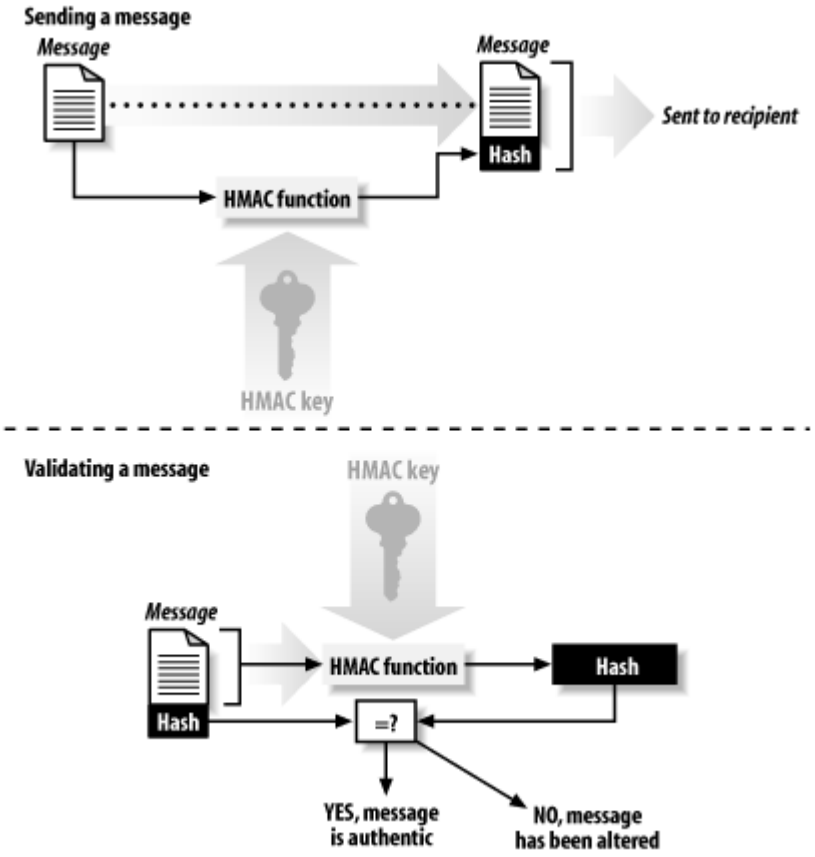
Izvor: https://tools.ietf.org/html/rfc7518#section-3 (17.7.2019.)

Prvi stupac tablice prikazuje set parametara odnosno vrijednosti koje zahtjev zaglavlja „alg“ može sadržati. Drugi stupac prikazuje puni naziv algoritma i kriptografske hash funkcije koju ta vrijednost predstavlja, dok treći stupac predstavlja potrebu implementacije algoritma.

5.1. HMAC algoritmi

HMAC ili Hash-Based Message Authentication Code je jedan od najčešće korištenih algoritama kod potpisa JWT-a. HMAC je simetrični algoritam čiji se rad bazira na korištenju jednog privatnog ključa kojeg znaju poslužitelj i klijent, te korištenju hash funkcije poput SHA-256. HMAC se razlikuje od MAC-a (Message Authentication Code) po tome što se ključ i poruka hashiraju u odvojenim koracima. To osigurava da proces nije podložan napadima proširenja kojima se dodaje poruka na kraju čime se može uzrokovati curenje elemenata ključa.[21] Također, HMAC-om se dokazuje da tko god je generirao MAC posjeduje MAC ključ. HMAC je algoritam učestalo korišten kod sigurnog transfera datoteka kod protokola poput FTPS, SFTP i HTTPS. Poznavanjem dijeljenog tajnog ključa obje stranke mogu potvrditi kako su zaprimljena poruka i MAC poslani od očekivane stranke. Primjer korištenja HMAC-a prikazan je slikom 13.

[21] Rouse M., Hash-based Message Authentication Code (HMAC), https://searchsecurity.techtarget.com/definition/Hash-based-Message-Authentication-Code-HMAC (18.7.2019.)



Slika 15. Prikaz rada HMAC algoritma kod sigurnog transfera datoteka

Izvor: https://docstore.mik.ua/orelly/other/puis3rd/0596003234_puis3-chp-7-sect-4.html

Ključevi se trebaju birati nasumično korištenjem kriptografski snažnog pseudo-nasumičnog generatora s nasumičnim sjemenom kako bi se osigurala sigurnost podataka.[22] Sigurnost HMAC-a bazirana je na funkciji hashiranja, odnosno na veličini bitova koja se koristi kod hashiranja. Temeljem sekcije 5.3.4 Nacionalnog Instituta Standarda i Tehnologija (engl. National Institute of Standards and Tehnology - NIST) pod nazivom „Recommendation for Applications Using Approved Hash Algorithms“ navodi se kako snaga sigurnosti je minimalna sigurnosna snaga ključa i dva puta veličine interne hash vrijednosti. Znači da ključ mora biti iste veličine ili veći od hash izlazne vrijednosti, što u primjeru zaglavlja JWT-a sa slike 12 gdje se koristi HS256 ključ mora biti minimalno 256 bitova ili veći. Još neki od popularnijih algoritama MAC-

[22] RFC 2104 - HMAC: Keyed-Hashing for Message Authentication, 1997., https://tools.ietf.org/html/rfc2104 (18.7.2019.)

a su HS384 (HMAC + SHA-384) i HS512 (HMAC + SHA-512). Najjači poznat napad na HMAC temelji se na učestalosti sudara hash funkcije tzv. „napadi rođendana“, no oni nemaju efekt na razumnoj hash funkciji. Za block veličine 64 bytova bilo bi potrebno otprilike 250.000 godina neprekidne konekcije od 1Gbps, uključujući uvjet da se ključ nije promijenio. [23]

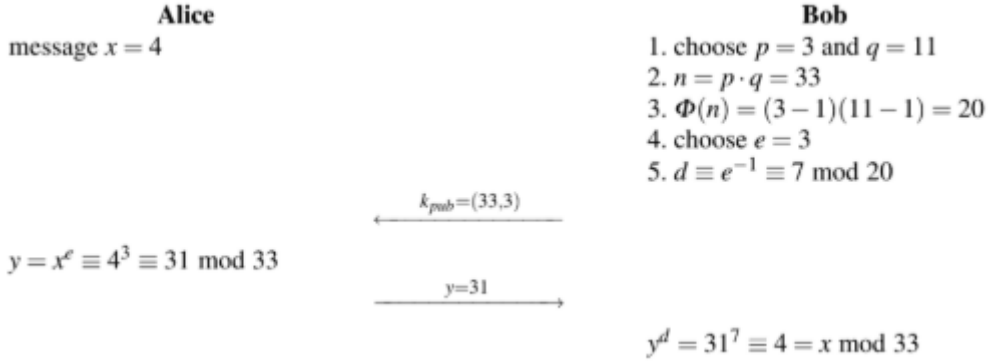
5.2. RSA digitalni potpis

RSA (Rivest-Shamir-Adleman) bazirani JWS osigurava integritet i autentičnost JWT-a. RSA potpis je jedan od prvih asimetričnih algoritama čiji rad se bazira na korištenju para privatnog i javnog ključa za siguran prijenos podataka. Poslužitelj posjeduje privatni ključ koji se koristi kod generiranja potpisa, dok primatelj JWT-a posjeduje javni ključ pomoću kojeg validira potpis. Ron Rivest, Adi Shamir i Len Adleman, izumitelji RSA kriptosistema, prvi puta su ga predstavili 1977. godine. Od njegove prve publikacije RSA kriptosistem bio je podložen mnogim istraživanjima u svrhu pronalaska slabosti, no najveća mana ovog sustava dolazi iz njegovog nepravilnog korištenja.[24] RSA-ov algoritam bazira se na problemu faktorizacije velikih brojeva, pa se za provalu zahtjeva veoma velika procesorska snaga.[25] Najmanja preporučena veličina ključa kod RSA sustava je 2048 bitova. RSA se u praksi učestalo koristi kod šifriranja malih količina podataka i kao digitalni potpis. Usto RSA se učestalo koristi zajedno sa simetričnim šifriranjem poput AES-a. Slikom 14 prikazan je postupak

[23] RFC 7518 - JSON Web Algorithms (JWA), op. cit.
[24] Boneh, D. (1999.), Twenty years of attacks on the RSA cryptosystem, *American Mathematical Society (AMS)*, 46 (2), str. 1.
[25] RSA Encryption, https://www.techopedia.com/definition/21852/rsa-encryption (19.7.2019.)

generiranja javnog i privatnog kod RSA kriptografskog sustava na popularnom primjeru „Alice & Bob“

Izvor: Paar, C., Pelzl, J. (2010.), *Understanding Cryptography*, Springer, str. 177.



Slika 16. Generiranje javnog i privatnog ključa

Prvim korakom slike 14 biraju se dva velika primarna broja p i q. Slijedećim koracima izračunavaju se privatni i javni ključ. Praktični primjeri RSA parametara su mnogo veći, odnosno ukoliko je n dužine 2048 bita parametri p i q su svaki 1024 bita.[26] Kriptografska specifikacija RSA-a sadržana je RFC-om 3447 naziva „Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1“ iz 2003. godine, no taj RFC je zamijenjen novijim RFC-om 8017 2016. godine naziva „PKCS #1: RSA Cryptography Specifications Version 2.2“. Taj RFC sadrži preporuke implementacije i upotrebe kriptografskog sustava javnog ključa baziranog na RSA algoritmu.[27] Tablicom 3 prikazani su najkorišteniji RSA algoritmi „alg“ zahtjeva zaglavlja JWT-a.

Tablica 3. Tablica najkorištenijih RSA algoritama kod JWT-a

Vrijednost parametra „alg“	**Algoritam digitalnog potpisa**
RS256	RSASSA-PKCS1-v1_5 using SHA-256

[26] Paar, C., Pelzl, J. (2010.), *Understanding Cryptography*, Springer, str. 175.
[27] RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2, 2016., https://tools.ietf.org/html/rfc8017 (20.7.2019.)

RS384	RSASSA-PKCS1-v1_5 using SHA-384
RS512	RSASSA-PKCS1-v1_5 using SHA-512

Izvor: https://tools.ietf.org/html/rfc7518#section-3 (24.7.2019.)

5.3. ECDSA digitalni potpis

ECDSA (Elliptic Curve Digital Signature Algorithm) je algoritam baziran na ECC (Elliptic Curve Cryptography) kriptosustavu koji radi na principu korištenja privatnog i javnog ključa poput RSA kriptosustava.[28] Princip rada ECDSA algoritma bazira se na korištenju matematičkih jednadžbi putem kojih se dobiva krivulja na grafu. Matematička jednadžba ECDSA formirana je poput **y^2 = (x^3 + a * x + b) mod p**. Iz te krivulje odabire nasumično se odabire jedna točka koja predstavlja točku podrijetla. Nakon generiranja nasumičnog broja koji prestavlja privatni ključ, korištenjem matematičke jednadžbe dobiva se druga točka podrijetla odnosno dobiva se javni ključ. Kod digitalnog potpisa korištenjem ECDSA sustava potpis se sastoji od dva broja R i S dobivenog izračunom privatnog ključa. Validacija se vrši korištenjem javnog ključa i broja S u metamatičkoj jednadžbi, te ukoliko se za rezultat dobije broj R potpis je validan.[29] Za razbijanje ECDSA ključa potrebno je riješiti ECDLP (Elliptic Curve Discrete Logarithm Problem) za čije rješavanje nije bilo velikih algoritamskih napredaka od njenog predstavljanja 1985. godine od strane Neala Koblitza i Victora Millera. Time ECDSA sustav postiže istu razinu sigurnosti kao i RSA sustav uz korištenje manjih veličina ključeva čime se dobiva značajno na brzini obrade kod mnogih operacija. Korištenjem manjih ključeva također doprinosi brzini algoritama zbog potrebe kalkulacije manjih brojeva, te doprinosi bržim konekcijama i učitavanjima web stranica zbog manje količine podataka slane TLS (Transport Layer Security) konekcijom. Eliptični ključ veličine 256 bita pruža jednaku zaštitu poput 3248 bitnog asimetričnog ključa. Kada se usporedi TLS handshake brzina 256 bitnog ECDSA ključa i 2048 bitnog RSA ključa rezultat prikazuje 9.5 puta veću uštedu operacija privatnog ključa, odnosno 256 bitnim ECDSA ključem postiže se 9516.8 potpisa u sekundi, dok 2048 bitnim RSA ključem samo 1001.8 potpisa u sekundi.[30] Neki od

[28] RFC 7518 - JSON Web Algorithms (JWA), op. cit.
[29] Understanding how ECDSA protects your data, https://www.instructables.com/id/Understanding-how-ECDSA-protects-your-data/ (16.9.2019.)
[30] Sullivan N., ECDSA: The digital signature algorithm of a better internet, 2014., https://new.blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/ (24.7.2019.)

poznatijih korisnika ECDSA sustava su Bitcoin i Apple. Tablicom 4 prikazani su učestalo korišteni ECDSA sustavi kod digitalnog potpisa JTW-a uz korištenje SHA algoritma.

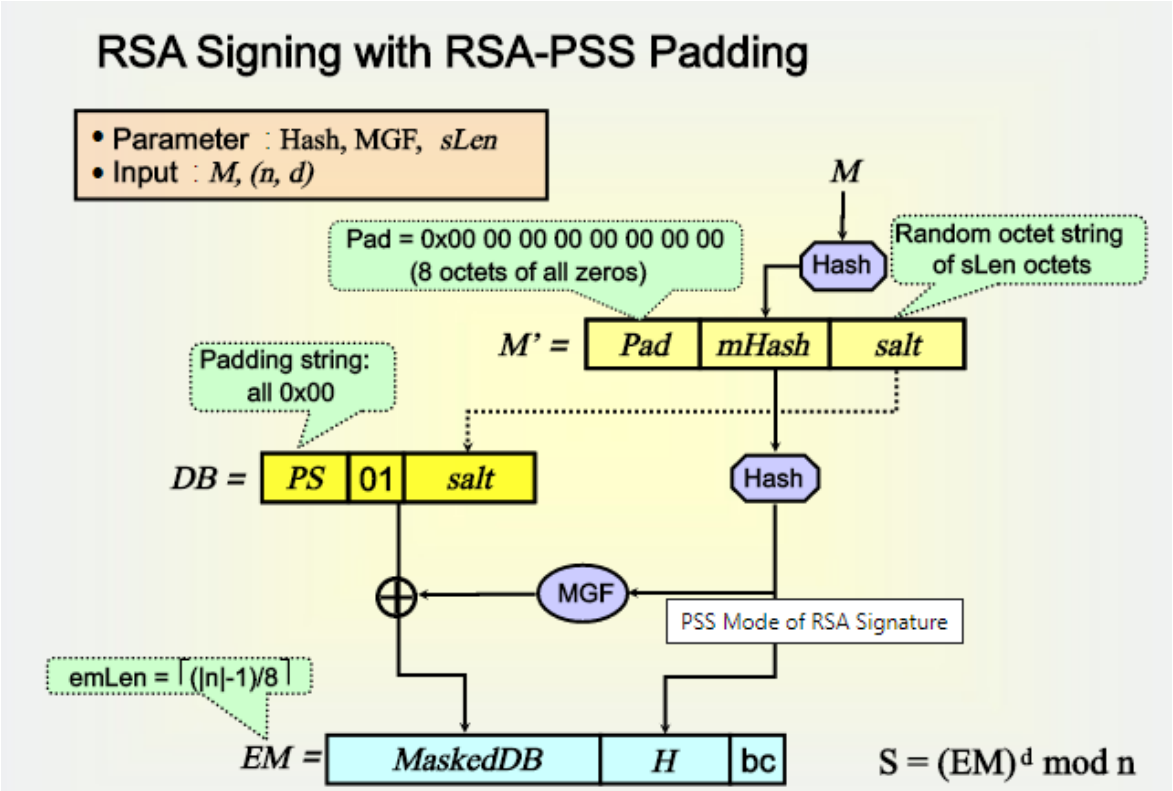
Tablica 4. Najkorišteniji ECDSA algoritmi kod JWT-a

Vrijednost parametra „alg“	**Algoritam digitalnog potpisa**
ES256	ECDSA using P-256 and SHA-256
ES384	ECDSA using P-384 and SHA-384
ES512	ECDSA using P-512 and SHA-512

Izvor: https://tools.ietf.org/html/rfc7518#section-3 (24.7.2019.)

5.4. RSASSA-PSS digitalni potpis

RSA-PSS (Probabilistic Signature Scheme) je novija shema digitalnog potpisa kreirana od strane Mihira Bellarea i Phillipa Rogawaya, a bazira se na RSA kriptografskom sustavu. PSS shema zahtijeva da sama poruka provjeri potpis tj. poruka se ne može više vratiti iz potpisa. Slikom 15 prikazano je korištenje PSS moda RSA potpisa.



Slika 17. Prikaz korištenja PSS moda RSA potpisa

Izvor: PSS mode of RSA signature, https://developpaper.com/pss-mode-of-rsa-signature/ (16.9.2019.)

Postoji nekoliko značajnih razlika između RSASSA-PKCS-v1_5 i RSASSA-PSS shema potpisa. PKCS shema je deterministična odnosno ista poruka i isti ključ će uvijek proizvesti identičnu vrijednost potpisa, dok je PPS nasumičan i svaku put će proizvesti različitu vrijednost osim ako se koristi „sol“ (eng. Salt) nulte duljine. Salt u kriptografiji predstavlja nasumičnu vrijednost korištenu kao dodatnu ulaznu vrijednost kod hashiranja podataka. Također, kod PSS sheme se ne može izdvojiti probavna vrijednost kao kod PKCS sheme, no može se provjeriti autentičnost prema poznatoj probavnoj vrijednosti. PSS shema sadrži sigurnosni dokaz i robusnija je od PKCSV1_5 sheme, no PKCSV1_5 je i dalje šire primjenjivana. Kod RSASSA-RSS sheme postoji set poznatih parametara koji uključuju:

1. Hash algritam/funkciju (zadana funkcija je SHA-1),
2. MGF (Mask Generation Function) odnosno funkcija generiranja maske,
3. Duljinu soli (Salt) – zadano vrijednost je 20, no konvencija je koristiti hLen (duljinu hash funkcije u bajtovima),
4. Polje prikolice (eng. Trailer field) – koristi se kod postupka kodiranja.[31]

Takav set parametara vidljiv je slikom 16.

```
hashAlgorithm       sha1,
maskGenAlgorithm    mgf1SHA1 (the function MGF1 with SHA-1)
saltLength          20,
trailerField        trailerFieldBC (the byte 0xbc)
```

Slika 18. Set zadanih parametara RSASSA-PSS sheme

Izvor: https://www.cryptosys.net/pki/manpki/pki_rsaschemes.html (24.7.2019.)

[31] D.I. Management Services Pty Ltd., 2004-19., RSA signature and encryption schemes: RSA-PSS and RSA-OAEP https://www.cryptosys.net/pki/manpki/pki_rsaschemes.html (24.7.2019.)

5.5. RSAES-OAEP šifriranje ključa

RSAES-OAEP je shema enkripcije korištenjem javnog ključa s kombinacijom OAEP-a (Optimal Asymmetric Encryption Padding) koji koristi integracijsku shemu oblaganja (eng. Padding) učestalo zajedno uz RSA šifriranje. OAEP je kreiran od strane Mihira Bellarea i Phillipa Rogawaya te standardiziran RFC-om 2437 naslova „PKCS #1: RSA Cryptography Specifications Version 2.0“ iz 1998. godine. OAEP algoritam je jedna forma Feistelove mreže koja koristi par slučajnih proroka (eng. random oracles) G i H.[32] Slučajni prorok je alat kojeg su također uveli Bellare i Rogaway kako bi se omogućilo strogo dokazivanje sigurnosti za neke kriptografske protokole poput OAEP-a. Hash funkcija H može se razmatrati kao crna kutija koja za svaki upit bitstringa M daje potpuno slučajnu vrijednost. Za svaki upit prorok radi neovisan slučajni odabir te čuva njegov zapis H(M) i ponavlja isti odgovor ako se radi isti upit M.[33] Slučajni proroci kao matematička abstrakcija su prvi puta bili spominjani 1993. godine u publikaciji Mihara Bellarea i Phillipa Rogawaya pod nazivom „Random Oracles are Practical: A Paradigm for Designing Ecient Protocols“. Kroz godine korištene se mnoge metode oblaganja uz RSA enkripciju, no OAEP ima prednost dobrih sigurnosnih jamstava i dobrih performansi. Također korištenjem OAEP-a uz RSA pruža se sigurnost protiv napada adaptivno odabranih šifriranih tekstova (eng. adaptive chosen ciphertext attacks) u kojima napadač ima priliku slanja upita proroku time simulirajući dešifrirajuće primitive. Ova vrsta napada je usko povezana s konceptom sigurnosti predstavljenim 1994. godine od strane Bellarea i Rogawaya pod nazivom „Plaintext awareness“ kojim su dokazali da ukoliko se primitiv šifriranja teško invertira bez privatnog ključa, tada je odgovarajuća OAEP bazirana shema šifriranja svjesna otvorenog teksta (eng. Plaintext) i time napadač ne može proizvesti valjani šifrirani tekst bez da zna otvoreni tekst.[34] Tablicom 5 prikazane su vrijednosti parametara RSAES-OAEP šifriranja ključa korištene kod JWT-a.

[32] Optimal asymmetric encryption padding, https://asecuritysite.com/encryption/rsa_oaep (26.7.2019.)
[33] Koblitz, N., Menezes, A. J. (2015.), The random oracle model: a twenty-year retrospective, *Journal Designs, Codes and Cryptography*, 77 (2-3), str. 1.
[34] RSAES-OAEP Encryption Scheme, 2000., https://www.inf.pucrs.br/~calazans/graduate/TPVLSI_I/RSA-oaep_spec.pdf (29.7.2019.)

Tablica 5. RSAES-OAEP algoritmi kod JWT-a

Vrijednost parametra „alg“	Algoritam upravljanja ključem
RSA-OAEP	RSAES OAEP using default parameters
RSA-OAEP-256	RSAES OAEP using SHA-256 and MGF1 with SHA-256

Izvor: https://tools.ietf.org/html/rfc7518#section-3 (29.7.2019.)

5.6. Omotavanje ključa AES algoritmom

Omotavanje ključa (eng. Key wrapping) pomoću AES algoritma (Advanced Encryption Standard) definirano je RFC-om 3394 naziva „Advanced Encryption Standard (AES) Key Wrap Algorithm“ iz 2002. godine. Takvi algoritmi se koriste za zaštitu ključeva u mirovanju ili prijenosi preko nesigurnih mreža. AES algoritam dizajniran je za omotavanje i šifriranje podataka ključa. Prije omotavanja podaci ključa se lome na blokove od 64 bita.[35] Omotavanje ključa je kriptografski konstrukt koji koristi simetrično šifriranje za enkapsulaciju podataka ključa. Ključ AES enkripcijskog ključa (eng. KEK – key encryption key) mora biti veličine 128, 192 ili 256 bita. Tablicom 6 prikazani su vrijednosti parametara korištenih kod AES algoritma omotavanja ključa iz koje se može vidjeti kako se nazivi parametara poklapaju sa specifikacijama veličine KEK ključa.

[35] RFC 3394 - Advanced Encryption Standard (AES) Key Wrap Algorithm https://tools.ietf.org/html/rfc3394 (29.7.2019.)

Tablica 6. Tablica AES key wrap algoritama kod JWT-a

Vrijednost parametra „alg“	Algoritam upravljanja ključem
A128KW	AES key wrap with default initial value using 128-bit key
A192KW	AES key wrap with default initial value using 192-bit key
A256KW	AES key wrap with default initial value using 256-bit key

Izvor: https://tools.ietf.org/html/rfc7518#section-3 (29.7.2019.)

5.7. Šifriranje ključa pomoću AES GCM

AES-GCM (eng. Advanced Encryption Standard with Galois Counter Mode) je mod operacije blokovske šifre koji omogućuje velike brzine operacija ovjerenog šifriranja i integriteta podataka predstavljen od strane Američkog nacionalnog instituta za standarde i tehnologije (NIST). AES-GCM ima dvije glavne funkcije, a to su šifriranje bloka šifre i množenje preko polja. Algoritam pruža šifriranje i dešifriranje korištenjem 128, 192 ili 256 bitnog ključa šifre. Broj izvršenih transformacija AES-a ovisi o duljini šifriranog ključa, čime se povećava vrijeme potrebno za stvaranje izlaza koje nadalje utječe na same performanse AES-GCM-a. Tablicom 7 prikazano je potreban broj rundi za određene dužine ključa.[36]

Tablica 7. Tablica potrebnih broja rundi ovisno o dužini ključa kod AES-GCM-a

	Dužina ključa (32-bitna riječ)	Veličina bloka (32-bitna riječ)	Broj rundi
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

[36] Ahmad, N., Wei, L. M., Jabbar, M. H. (2018.), Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array, *IOP Conf. Series: Journal of Physics: Conf. Series 1019*, IOP Publishing, str. 2.

Izvor: https://iopscience.iop.org/article/10.1088/1742-6596/1019/1/012008/pdf (3.8.2019.)

Vrijednosti parametara AES-GCM-a kod zahtjeva zaglavlja „alg“ prikazane su tablicom 8 iz koje je vidno kako se nazivi zahtjeva poklapaju s pravilima korištenja 128, 192 ili 256 bitnog ključa.

Tablica 8. Tablica AES-GCM parametara zahtjeva zaglavlja

Vrijednost parametra „alg“	**Algoritmi upravljanja ključevima**
A128GCMKW	Key wrapping with AES GCM using 128-bit key
A192GCMKW	Key wrapping with AES GCM using 192-bit key
A256GCMKW	Key wrapping with AES GCM using 256-bit key

Izvor: https://tools.ietf.org/html/rfc7518#section-3 (3.8.2019.)

6. SIGURNOSNE PREDNOSTI I NEDOSTACI JWT-A

Pravilnim korištenjem JWT-a može se osigurati ispravnost kao i sigurnost podataka. JTW se učestalo koristi kod autentikacije i autorizacije korisnika, server-to-server autorizacije, sesijskih tokena i mnogih drugih. JWT je tehnologija koja je jako popularna i o kojoj se mnogo govori, što pokazuje i statistika sa slike 17 koja prikazuje popularnost JWT-a u prve dvije godine od njegova nastanka.

- 972 GitHub repos related to JWT.
- 2600+ StackOverflow threads.
- 400K page views on jwt.io.
- 50K Google results.

Slika 19. Popularnost JWT-a u prve dvije godine od nastanka

Izvor: https://auth0.com/blog/jwt-json-webtoken-logo/ (4.8.2019.)

No kao i svaka druga tehnologija JWT ima svoje prednosti i nedostatke. Ovim poglavljem bit će obrađene neke od točaka prednosti kao i nedostaci, te će se detaljno objasniti njihova pojava kod JWT tehnologije.

Neke od prednosti korištenja JWT-a su:

- Jednostavnost korištenja,
- Dobra učinkovitost,
- Kompaktnost,
- Mogućnost korištenja na više servisa,
- Prenosnost,
- Decentralizirana arhitektura,
- Smanjuje potrebu korištenja baze podataka.

Kao što je bilo predstavljeno prethodnim poglavljima tehnologija JWT-a je vrlo jednostavna za korištenje i ukoliko se koristi u skladu sa specifikacijama smatra se vrlo sigurnom za korištenje. Zahvaljujući mnogobrojnim gotovim bibliotekama i paketima za implementaciju JWT-a i njegovo korištenje, njegovo vrijeme i kompleksnost implementacije se uvelike smanjuje te se zbog tih razloga učestalo koristi. JWT je također popularan zbog svoje kompaktnosti što pridonosi boljim performansama. Mala veličina znači da je prijenos brz te se može poslati putem URL-a, POST parametra ili unutar HTTP zaglavlja. Ograničenje veličine JWT-a obično je 7 do 8 kilobita. Performansama također pridonosi činjenica da sadržaj (eng. payload) JWT-a sadrži sve potrebne informacije o korisniku što negira potrebu za korištenje baze podataka više od jednom. Time se značajno smanjuje vrijeme odaziva te ukoliko se koriste baze podataka koje naplaćuju uslugu korištenja po broju provedenih upita može se marginalno uštedjeti na troškovima zbog manje potrebe za istim. Jedna od prednosti JWT tokena je ta da se jedan token može koristiti s više backend servisa. Također se može koristiti decentralizirani sustav jer token može biti generiran bilo gdje, odnosno autentikacija može biti provedena na resursnom serveru ili može biti odvojena na svoj zasebni server. Potreba za upravljanje sesijama također je riješena zbog informacija sadržanih JWT-ovim sadržajem, te se jedna token može koristiti kroz više sesija. Token također sadrži podatke poput vremena isteka važenja tokena što pridonosi tome da je JWT van sesijska tehnologija.[37] No uz sve navedene prednosti postoje i mnogobrojni nedostaci koje povezujemo s korištenje JWT tehnologije.

Neki od tih nedostataka su:

- Kompromizirani tajni ključ,
- Zastarjeli kriptografski algoritam,
- Nedostatak znanja o kriptografiji,
- Lako je prekoračiti veličinu JWT-a.

JWT tokeni se smatraju veoma sigurnim za korištenjem ukoliko se koriste pravilno, no ova tehnologija ima jednu veliku prednost koja se isto tako može smatrati velikim

[37] Raghuwanshi R., JWT (JSON Web Tokens) Are Better Than Session Cookies, 2017., https://dzone.com/articles/jwtjson-web-tokens-are-better-than-session-cookies?fbclid=IwAR0J2qCBPFgv5q35lqHseXKvn5VF6UqQs3WH-ggkgCKm9f0_ZIf0J9lybk8 (1.9.2019.)

nedostatkom ovog sustava, a to je korištenje istog ključa. Ukoliko je ključ pohranjen na sigurnom mjestu JWT je siguran za korištenje. Ukoliko dođe do curenja ključa npr. od strane nemarnog ili nepažljivog developera ili admina, cijeli sustav je ugrožen i sklon provali. Napadač koji ima pristup tajnom ključu ima pristup svim podacima. Ukoliko dođe do ovog scenarija jedini način oporavka je generiranje novog tajnog ključa, no s time svi klijentovi tokeni su nevažeći i svi korisnici se trebaju ponovno prijaviti u sustav. Većina proboja dolazi kroz korištenje socijalnog inženjerstva, a ne kroz komplicirana tehnička hakiranja. Sljedeći veliki nedostatak JWT-a može biti zastarjeli kriptografski algoritam. JWT je tehnologija koja svoju sigurnost kompletno bazira na kriptografskom algoritmu potpisa. Iako ovo nije učestalo u prošlosti su zabilježeni mnogobrojni primjeri gdje se kriptografski algoritam smatrao zastarjelim zbog pronalaska načina provale istog. Postoji primjer WEP enkripcije koja je bila najkorištenija enkripcija za zaštitu wifi lozinke te se pronašao nedostatak i bila je provaljena i time se smatrala zastarjelom tehnologijom. Ukoliko dođe do toga da se algoritam smatra zastarjelim, a koristi se kod JWT-a za digitalni potpis, svi korisnici bi se trebali ponovno prijaviti u sustav i prošlo bi neko vrijeme dok se postojeće biblioteke JWT-a ne bi ažurirale novim kriptografskim algoritmom. Nadalje, iako se korištenje JWT-a smatra relativno jednostavnim i njegovu implementaciju podupiru mnogobrojne biblioteke i paketi, razumijevanje samog kriptografskog algoritma digitalnog potpisa JWT-a zahtijeva osnovno, ako ne i naprednije poznavanje kriptografije, koju mnogi developeri možda ne posjeduju. Ukoliko developer nema adekvatnog znanja o kriptografiji, on može nenamjerno implementirati loš kriptografski sustav ili implementirati ozbiljne sigurnosne propuste (eng. loophole) u sustav. Jedan od primjera takvog propusta mogao bi biti pohrana cijelog korisnikovog objekta u token.[38] Zadnji nedostatak JWT-a protivi se jednoj od prednosti koja je bila navedena u ovom poglavlju, a to je veličina JWT-a. Iako JWT token može biti male veličine, učestalo njegova veličina premašuje veličinu keksa (eng. cookie). Kod JWT-a sve informacije sadržane su u sadržaju tokena. To u kombinaciji s lošim developerom može dovesti do velikih količina podataka sadržanih tokenom. Svaki zahtjev nad sustavom zahtijeva slanje tokena i ukoliko je njegova veličina velika može doći do loših performansa, odnosno do velikih vremena potrebnih za slanje JWT-a.

[38] Golwalkar R., Pros and cons in using JWT (JSON WEB TOKENS), 2016., https://medium.com/@rahulgolwalkar/pros-and-cons-in-using-jwt-json-web-tokens-196ac6d41fb4 (1.9.2019.)

7. PRIMJER APLIKACIJE KORIŠTENJA JWT-A KOD AUTENTIKACIJE

Ovim dijelom diplomskog rada fokus je bio na izradi jednostavne aplikacije koja prikazuje proces autentikacije korisnika uz pomoć JWT-a. Prije same izrade aplikacije, odnosno prije početka pisanja koda aplikacije potrebno je instalirati nekoliko paketa i programa potrebnih za provedbu zadatka programiranja. Paketi su bili instalirani prema potrebi i tako će biti opisani u diplomskom radu. Za programiranje korišten je Visual Studio Code text editor. Kod instalacije paketa kao i kod inicijalizacije projekta korišten je terminal Visual Studio Code-a. Naredbom „npm init –y“ kreirana je datoteka „package.json“ koja sadrži sve korištene pakete i time je inicijaliziran projekt. Prvi paket koji je bio instaliran prije početka pisanja koda bio je Express naredbom „npm install express“. Express je node.js modul korišten za upravljanje tkz. rutama (eng. routes). On utvrđuje kako će aplikacija reagirati na zahtjev klijenta prema određenoj krajnjoj točki koja je URI (eng. Uniform Resource Identifier) i specifičnom HTTP zahtjevom metode (GET,POST...).[39] Još jedan paket koji je bio potreban prije samog kodiranja bio je Nodemon. Nodemon je alat korišten kod izrade node.js baziranih aplikacija te se koristi za automatsko ponovno pokretanje aplikacije kod promjene datoteke. Za njegovu instalaciju korištena je naredba „npm install nodemon“.[40] Za pokretanje servera korištena je funkcija „app.listen“ prikazana kodom s parametrom porta poslužitelja koji će se kasnije koristiti kod slanja podataka serveru. Fukcija je također izvršava jednostavan ispis u konzoli zbog provjere rada servera.

```
app.listen(3000, () => console.log('Server running'))
```

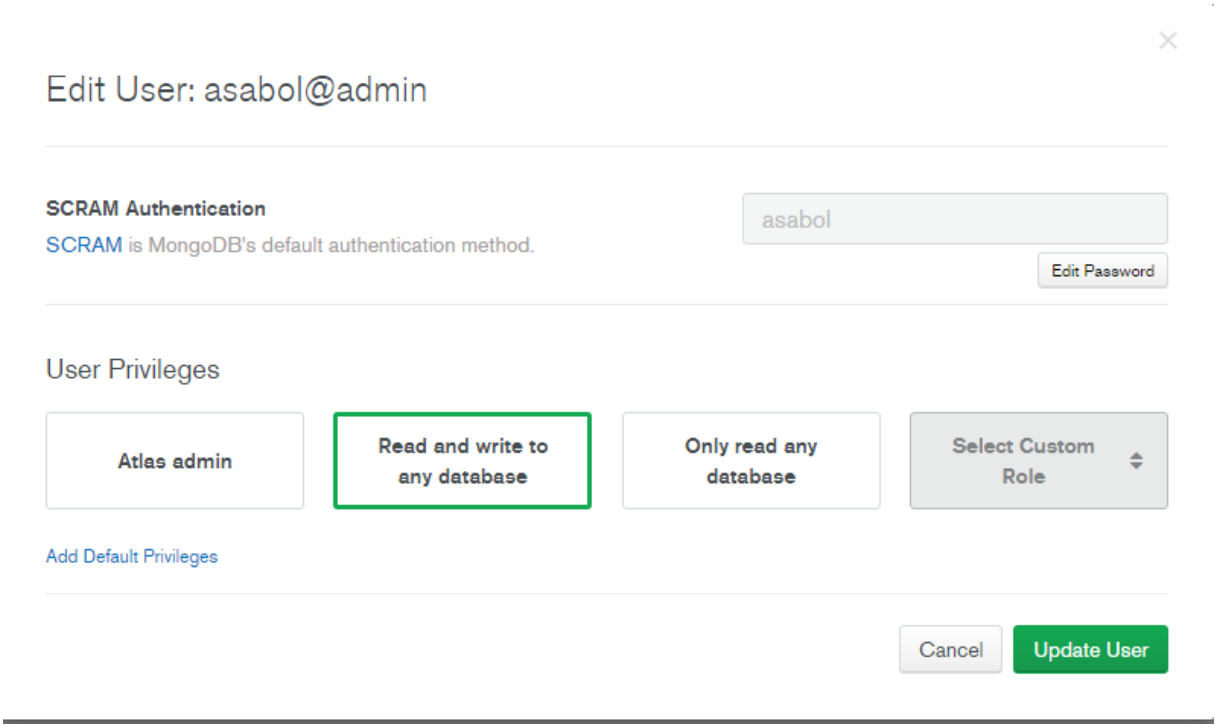
Kod 1. Funkcija postavke servera na port 3000

Izvor: Autor

Slijedeći korak projekta bio je postavljanje baze podataka. Za bazu je odabrana MongoDB Atlas baza koja radi na principu potpuno upravljane baze u oblaku. Korišteni su tzv. klasteri (eng. clusters) koji su implementacije MongoDB baze kojima upravlja Atlas. Prva stvar koju je bilo potrebno postaviti kod MongoDB baze su prava pristupa.

[39] Express, https://www.npmjs.com/package/express (17.8.2019.)
[40] Nodemon, https://www.npmjs.com/package/nodemon (17.8.2019.)

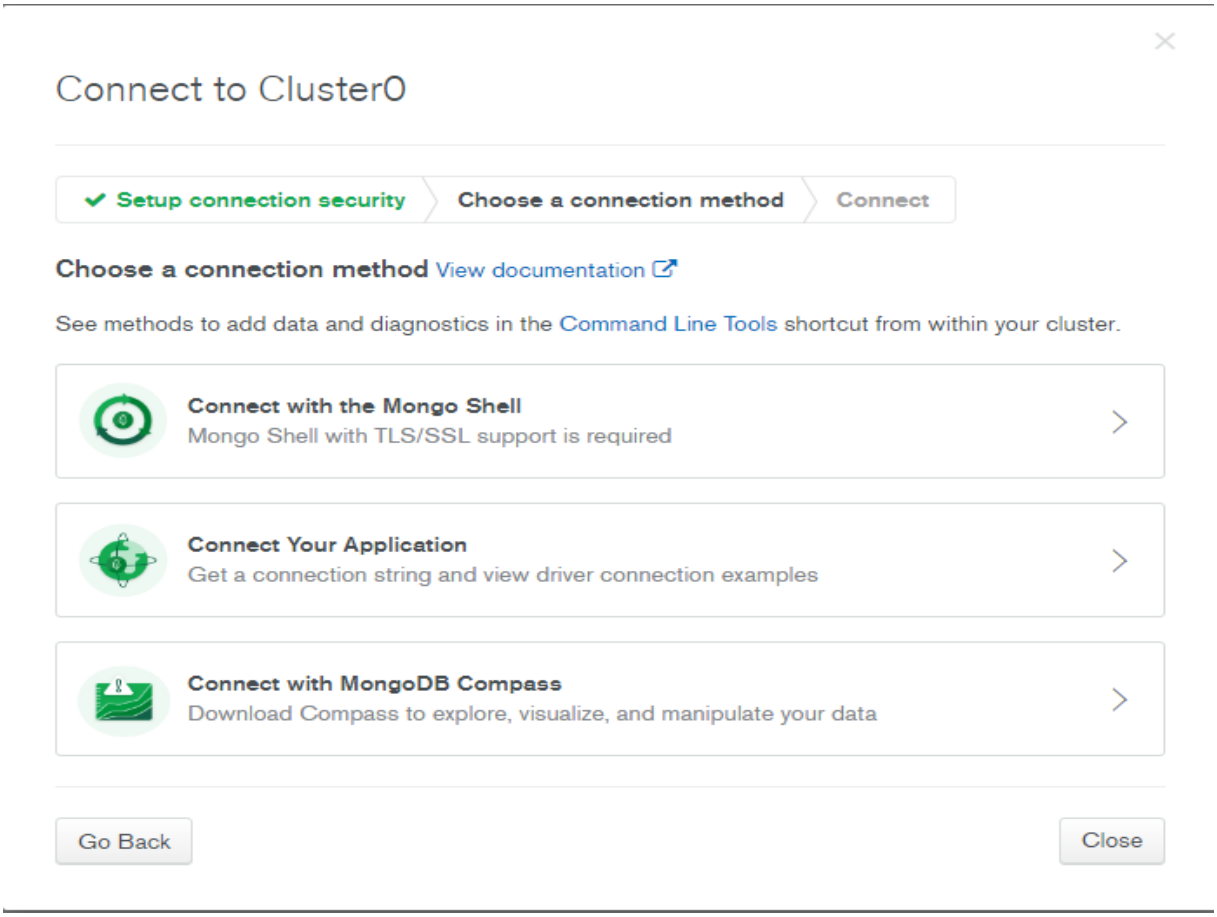
Slikom 18 prikazan je postupak izrade korisnika i dodjeljivanje korisničkih prava putem sučelja baze.



Slika 20. Izrada korisnika u MongoDB bazi podataka

Izvor: https://cloud.mongodb.com (17.8.2019.)

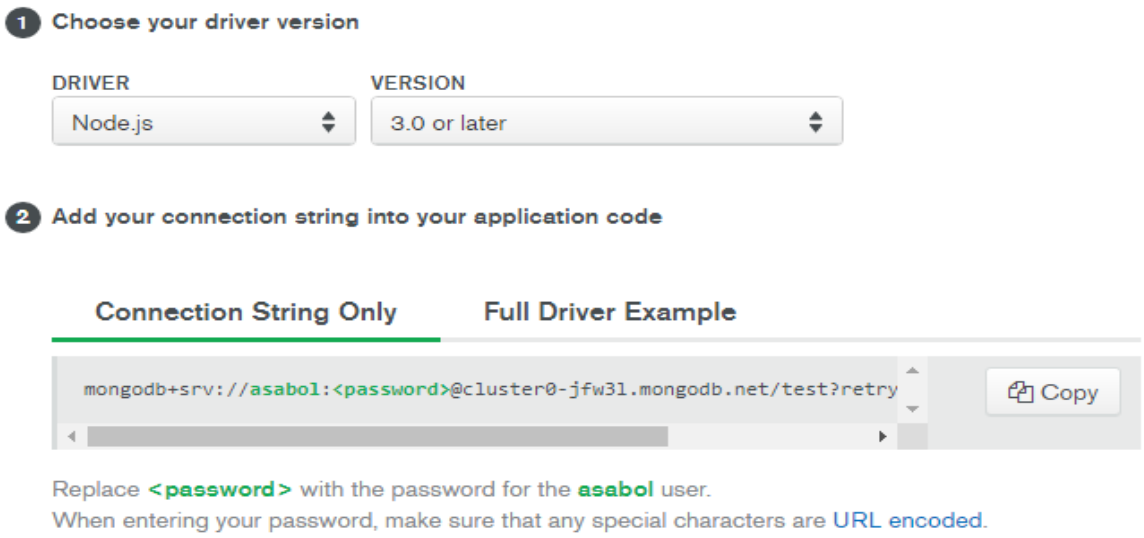
Također kod postavke MongoDB baze bilo je potrebno postaviti novu IP adresu na listu dopuštenih adresa. Ovaj proces je bio vrlo jednostavan putem korisničkog sučelja baze gdje se klikom miša na gumb naziva „Add current IP address“ postavila adresa trenutnog računala s kojeg je korisnik povezan na MongoDB. Za potrebe lakšeg upravljanja MongoDB bazom i izrade modela korišten je alat Mongoose koji je korišten za asinkroni rad s okruženjem. Instalacija Mongoose paketa provedena je putem terminala naredbom „npm install mongoose“. Zadnji korak povezivanja baze podataka s aplikacijom zahtjevao je navigiranje kroz korisničko sučelje MongoDB-a do izbornika sa slike 19 gdje je odabrana opcija „Connect Your Application“.



Slika 21. Sučelje za povezivanje aplikacije s bazom

Izvor: https://cloud.mongodb.com (17.8.2019.)

Sljedećim korakom sučelja dobiva se mogućnost odabira verzije drivera koji je u slučaju ove aplikacije bio node.js 3.0 ili starije verzije i MongoDB baza prikazuje liniju koda koju je potrebno ubaciti u kod aplikacije za uspješno povezivanje. Slikom 20 prikazan je kod potreban za povezivanje aplikacije s bazom.



Slika 22. Sučelje MongoDB-a za povezivanje aplikacije

Izvor: https://cloud.mongodb.com (17.8.2019.)

Iz prikazane linije koda sa slike 20 može se vidjeti kako su sadržani podaci za prijavu korisnika u bazu, točnije linija koda sadrži korisničku oznaku i lozinku. Dobra praksa programiranja je zaštita takvih podataka te se za rješavanje tog problema koristio paket dotenv koji je zapravo modul nulte ovisnosti koji učitava varijable iz .env datoteke i temelji se na metodologiji zvanoj 12 faktora aplikacije. Ova metodologija se može koristiti kao set pravila i smjernica kod procesa izrade aplikacija u programskim jezicima. Za instalaciju dotenv paketa korištena je naredba „npm install dotenv“.[41] Nakon uspješne instalacije paketa linija koda sa slike 20 prebačena je u odvojenu datoteku nastavka .env te se u glavnoj datoteci naziva index.js dodaje linija koda „const dotenv = require('dotenv');“ kojom se specificira kako je potrebno koristiti dotenv paket.

Datoteka .env također sadrži tajni ključ o kojem će biti riječ kasnije u ovom poglavlju. Sljedeći korak izrade aplikacije je bio izrada datoteke naziva User.js u kojem je sadržana shema za registraciju korisnika u bazi podataka. Shema korisnika nazvana „userSchema“ je objekt koji predstavlja šablonu tj. model koji je korišten kod izrade novog korisnika u bazi podataka. Kod prikazuje datoteku User.js koja sadrži shemu korisnika.

[41] Dotenv, https://www.npmjs.com/package/dotenv (27.8.2019.)

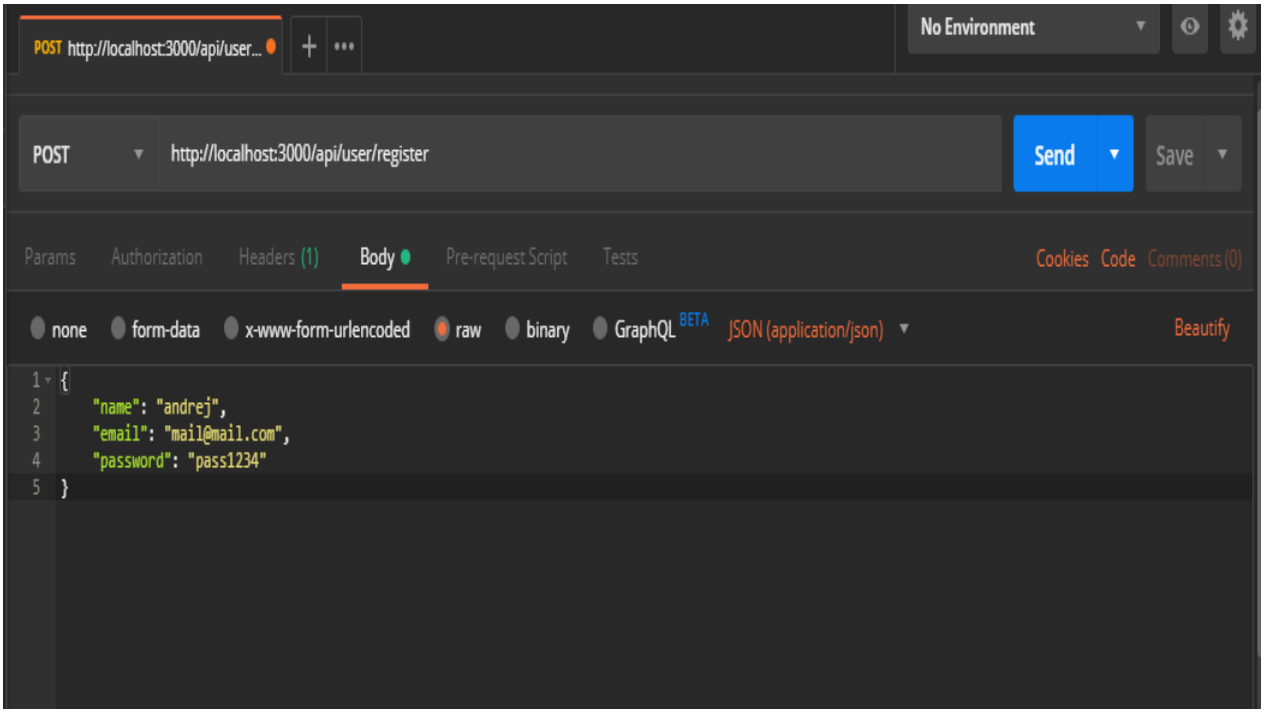
```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema ({
    name: {
        type: String,
        required: true,
        min: 6,
        max: 255
    },
    email: {
        type: String,
        required: true,
        min: 6,
        max: 255
    },
    password: {
        type: String,
        required: true,
        min: 6,
        max: 1024
    },
    date: {
        type: Date,
        default: Date.now
    }
});

module.exports = mongoose.model('User', userSchema);
```

Kod 2. Kod sadržan datotekom User.js

Izvor: Autor (27.8.2019.)

Kodom 2 prikazuje se shema korištena kod izrade novog korisnika i vidljivo je kako je za izradu korisnika potrebno odrediti ime, email, lozinku i datum. Svojstva ime, email i lozinka imaju definirano svojstvo naziva required kojim je definirano kako je potrebno to polje obavezno ispuniti kod izrade korisnika, dok svojstvo date ima unaprijed zadanu vrijednost svojstvom naziva default ukoliko se polje ostavi neispunjeno. Također prilikom izrade svojstva sheme korisnika postavljeni su neki zahtjevi ispunjavanja polja poput minimalnog i maksimalnog broja znakova. Sljedeći korak provedbe programiranja aplikacije je bila instalacija API razvojnog okruženja Postman. Postman je aplikacija koja se koristi kod izrade, upravljanja i održavanja API-a. Nakon uspješne instalacije aplikacije sljedeći korak bio je testiranje funkcije kreiranja novog korisnika.

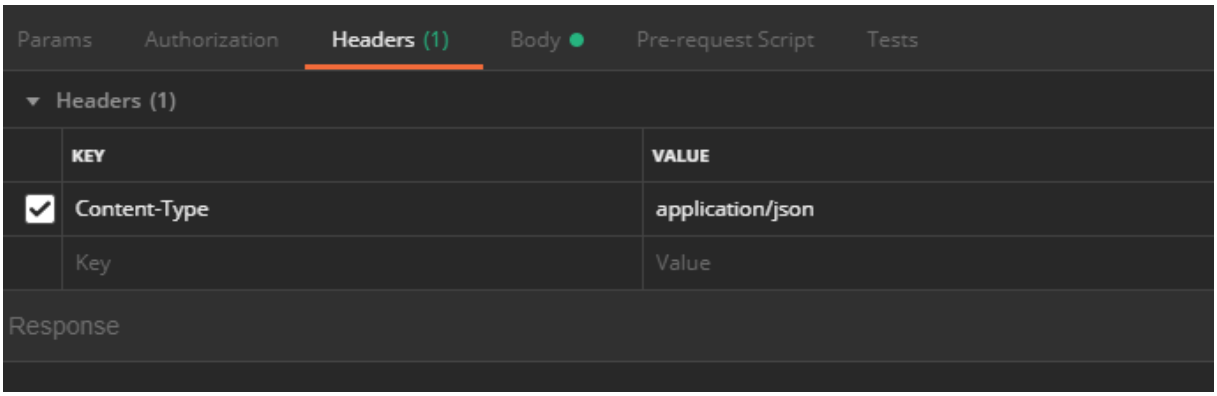
U korisničkom sučelju aplikacije Postman sa slike 21 potrebno je korektno ispuniti nekoliko polja kako bi se dobili željeni rezultati.



Slika 23. Sučelje Postman aplikacije

Izvor: Autor (29.8.2019.)

Prvi korak je bio odabir metode slanja koja je u mojem slučaju bila „post“ što je bilo određeno linijom koda „router.post('/register', async (req, res)“ sadržane u datoteci auth.js gdje je bila definirana ruta „register“ za registraciju novih korisnika. Drugi korak je unos ispravnog URL-a za slanje koji se sastoji od nekoliko predodređenih djelova poput porta poslužitelja koji je postavljen na 3000 linijom koda „app.listen(3000, () => console.log('Server started'));“. Zadnji korak je upisivanje ispravnih podataka kojima se kreira novi korisnik poput imena, emaila i lozinke. Važno je napomenuti kako su ti podaci upisani u JSON obliku te je potrebno postaviti vrstu sadržaja na vrijednost „application/json“ što je prikazano slikom 22.



Slika 24. Postavljanje vrijednosti zaglavlja

Izvor: Autor (29.8.2019.)

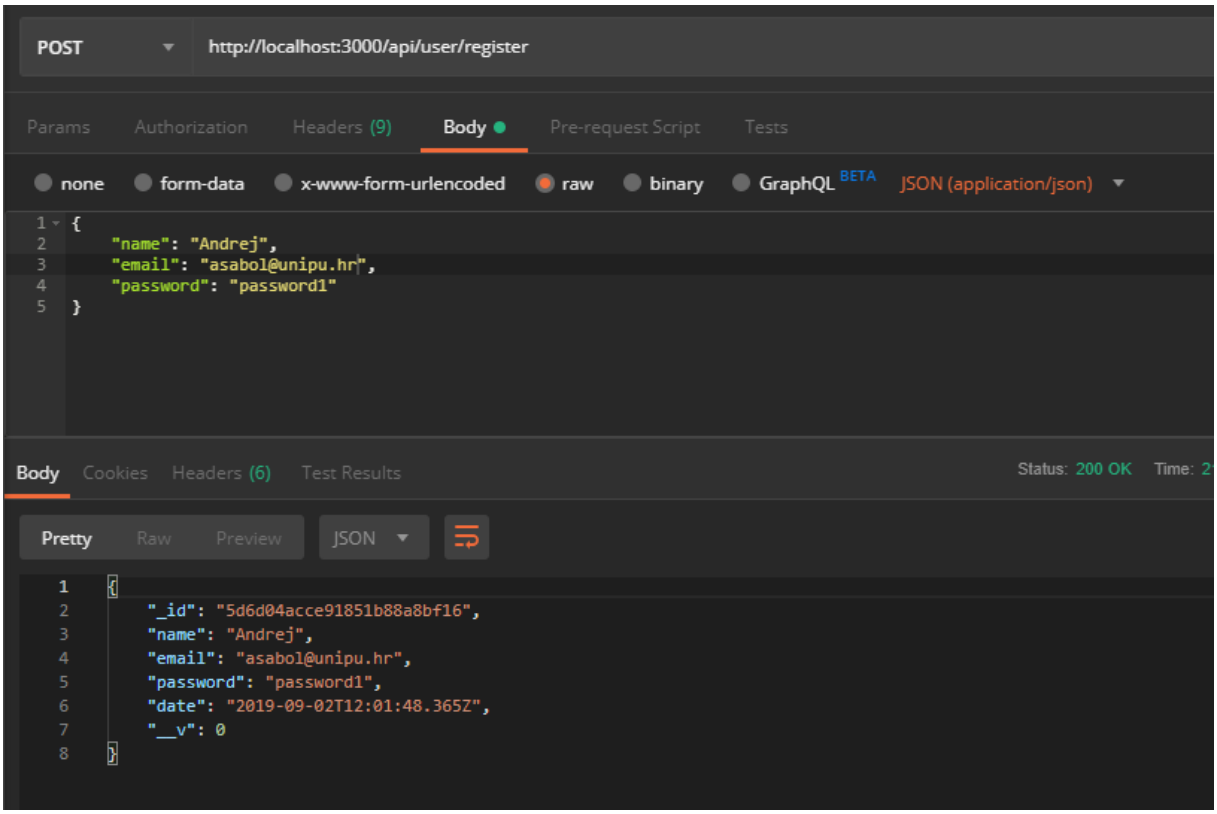
Nakon uspješnog slanja podataka u JSON obliku u Postman aplikaciji dobiva se povratna informacija uspješnog slanja i u MongoDB bazi vidljivi je novokreirani korisnik što možemo vidjeti slikom 23.


```
_id: ObjectId("5d6d04acce91851b88a8bf16")
name: "Andrej"
email: "asabol@unipu.hr"
password: "password1"
date: 2019-09-02T12:01:48.365+00:00
__v: 0
```


Slika 25. Novokreirani korisnik u bazi podataka

Izvor: Autor (29.8.2019.)

Također, povratnu informaciju prilikom kreiranja novog korisnika dobiva se i u programu Postman što prikazuje slika 24.



Slika 26. Novokreirani korisnik u programu Postman

Izvor: Autor (29.8.2019.)

Zbog potrebe validacije podataka kod izrade novog korisnika sljedeći korak izrade aplikacije bio je instalacija paketa naziva „@hapi/joi“. Instalacija paketa je provedena naredbom terminala npm install @hapi/joi. Ovo je paket korišten kod validacije javascript objekata što je u slučaju ove aplikacije bilo potrebno kod izrade novog korisnika gdje su postavljena ograničenja poput minimalnog broja karaktera kod odabira imena, emaila i lozinke. Sve funkcije potrebne kod autentikacije su postavljene u novoj datoteci nazvanoj validation.js koja izgleda poput primjera koda 3.

```
const Joi = require('@hapi/joi');

const registerValidation = (data) => {
    const schema = {
        name: Joi.string().min(6).required(),
        email: Joi.string().min(6).required().email(),
        password: Joi.string().min(6).required()
    };
    return Joi.validate(data, schema);
}
```

```
const loginValidation = (data) => {
    const schema = {
        email: Joi.string().min(6).required().email(),
        password: Joi.string().min(6).required()
    };
    return Joi.validate(data, schema);
}

module.exports.registerValidation = registerValidation;
module.exports.loginValidation = loginValidation;
```

Kod 3. Kod datoteke validation.js

Izvor: Autor (29.8.2019.)

Iz primjera koda 3 vidljivo je kako postoje dvije funkcije validacije, validacija registracije novog korisnika i validacija prijave postojećeg korisnika. Funkcija validacije novog korisnika postavlja dva zahtjeva odnosno ograničenja minimalnog broja karaktera i potrebu obaveznog ispunjavanja polja na polje korisničkog imena i lozinke i dodatni zahtjev na polje emaila koji osigurava unos pravilne forme emaila. Validacija kod prijave postojećeg korisnika provjerava pravilan unos polja emaila i lozinke. Nadalje funkcije registerValidation i loginValidation koriste se u datoteci auth.js gdje se još vrši dodatna validacija vidljiva kodom 4 za registraciju i prijavu korisnika.

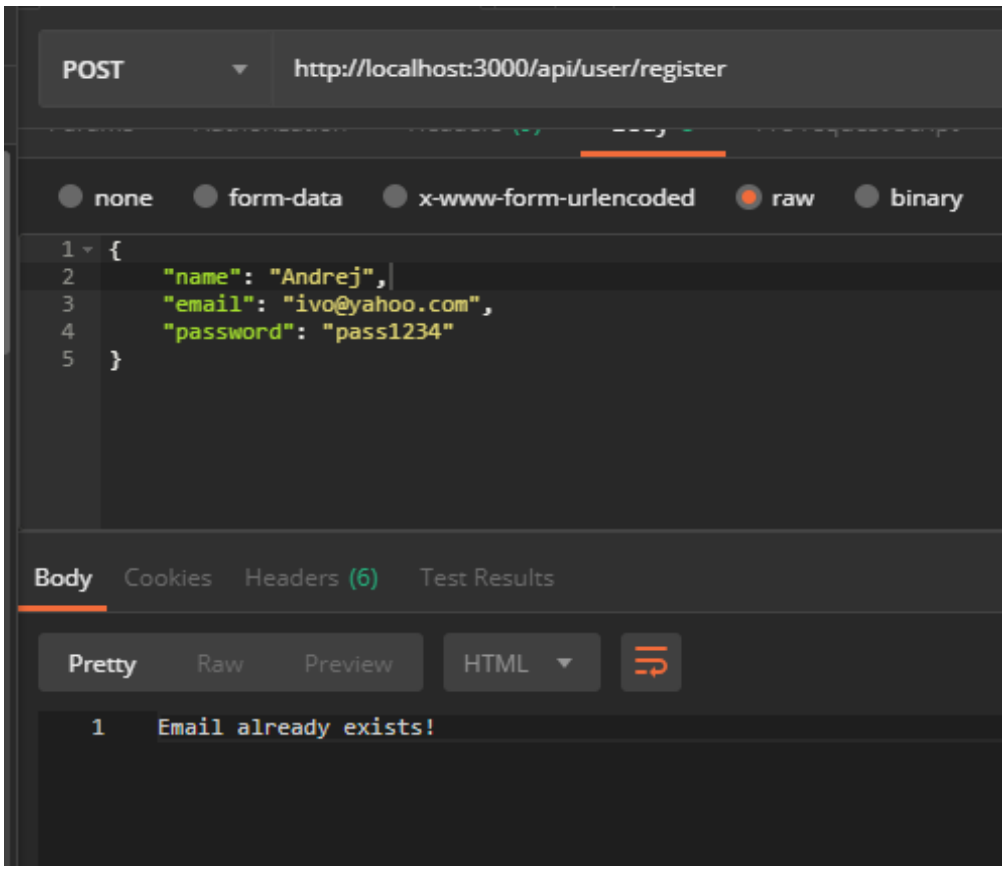
```
router.post('/register', async (req,res) =>{

    const { error } = registerValidation(req.body);
    if(error) return
res.status(400).send(error.details[0].message);

    const emailExists = await User.findOne({email:
req.body.email});
    if(emailExists) return res.status(400).send('Email already
exists!');

    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await
bcrypt.hash(req.body.password, salt);

router.post('/login', async (req, res) => {
    const { error } = loginValidation(req.body);
    if(error) return
res.status(400).send(error.details[0].message);
```

```
    const user = await User.findOne({email: req.body.email});
    if(!user) return res.status(400).send('Invalid email');

    const validPass = await bcrypt.compare(req.body.password,
user.password);
    if(!validPass) return res.status(400).send('Invalid
password');
```

Kod 4. Kod validacije korisnika

Izvor: Autor (29.8.2019.)

Kod registracije korisnika dodatno se provjerava polje emaila tako da se uspoređuje s postojećim emailovima pohranjenim u bazi podataka kako ne bi bilo moguće prijaviti postojeći mail više puta. Kod prijave korisnika provjerava se da li unešeni email i lozinka odgovaraju emailu i lozinki pohranjenoj u bazi. Primjer neuspješne prijave korisnika prikazan je slikom 25 gdje je uzrok pogrešne prijave korisnika pogrešna lozinka.



Slika 27. Primjer neuspjele prijave korisnika

Izvor: Autor (1.9.2019.)

Sljedeći korak aplikacije bio je implementacija hash funkcije korištene za hashiranje lozinke zbog dodatne zaštite lozinke jer pohranjivanje lozinke u obliku običnog teksta (eng. plain text) nije dobra praksa. Za to se koristila biblioteka bcrypt čija je instalacija također provedena naredbom npm install bcryptjs. Bcrypt je funkcija hashiranja lozinke dizajnirana od strane Nielsa Provosa i Davida Mazieresa 1999. godine. Ova funkcija bazirana je na simetričnoj blok šifri zvanoj Blowfish te koristi tzv. sol za dodatni input kod hashiranja.[42] Blowfish šifra je simetrična blok šifra nastala 1993. godine od strane Brucea Schneiera. Učestalo se koristi kod sigurnosti emaila, softvera za sigurnosne kopije, enkripcijskih alata i tiVo uređaja. Blowfish se smatra veoma brzom i efikasnom blokovskom šifrom koja generira veliki ključ što pridonosi sigurnosti. Također kod Blowfish blokovske šifre blokovi se dijele na duljine od 64 bita.[43] Primjerom aplikacije rađene ovim projektnim dijelom koristila se „sol“ duljine 10 bita korištena kod hashiranja lozinke što je vidljivo kodom 5.

```
const salt = await bcrypt.genSalt(10);
    const hashedPassword = await
bcrypt.hash(req.body.password, salt);

    const user = new User({
        name: req.body.name,
        email: req.body.email,
        password: hashedPassword
    });
    try {
        const savedUser = await user.save();
        res.send({ user: user_id });
    } catch (err) {
        res.status(400).send(err);
    }
});
```

Kod 5. Kod korišten za hash funkciju

Izvor: Autor (1.9.2019.)

Iz koda slike vidljivo je kako se u varijablu nazvanu salt pohranjuje sol duljine 10 bitova, te se varijabla „salt“ koristi u funkciji bcrypt.hash kao drugi parametar gdje je prvi

[42] Bcrypt, https://www.npmjs.com/package/bcrypt (1.9.2019.)

[43] Gatliff B., Encrypting data with the Blowfish algorithm, 2003., https://www.design-reuse.com/articles/5922/encrypting-data-with-the-blowfish-algorithm.html (1.9.2019.)

parametar lozinka koju je korisnik odabrao prilikom registracije novog korisnika. Hashirani oblik lozinke pohranjuje se u varijablu naziva hashedPassword koja se nadalje koristi kod kreiranja novog korisnika što je vidljivo linijom 23 primjera koda sa slike. Slikom 26 prikazana je uspješna registracija novog korisnika u bazi koja ima hashirani oblik korisničke lozinke.

Slika 28. Hashirana lozinka u bazi podataka

Izvor: Autor (1.9.2019.)

Zadnji dio projekta zahtijevao je korištenje json web tokena te za potrebu korištenja JWT-a instaliran je paket „jsonwebtoken“ naredbom npm install jsonwebtoken. Kreirana je nova datoteka naziva „verifyToken.js“ koja sadrži funkciju provjere tokena te je prikazana kodom 6.

```
const jwt = require('jsonwebtoken');

module.exports = function (req,res,next){
    const token = req.header('auth-token');
    if(!token) return res.status(401).send('Access Denied!');

    try {
        const verified = jwt.verify(token,
process.env.TOKEN_SECRET);
        req.user = verified;
        next();
    } catch (err) {
        res.status(400).send('Invalid token');
    }
}
```

Kod 6. Kod datoteke verifyToken.js

Izvor: Autor (1.9.2019.)

Ovom aplikacijom izrada tokena je izvršena na dva načina, jedan je bio korištenje biblioteke „jsonwebtoken“, a drugi način izrade bio je manualna izrada putem dva objekta nazvana „h“ i „p“ koji su predstavljali zaglavlje i sadržaj tokena, te upotrebom biblioteke „cryptoJS“ za base64url šifriranje zaglavlja i sadržaja tokena. Ručna izrada tokena prikazana je kodom 7.

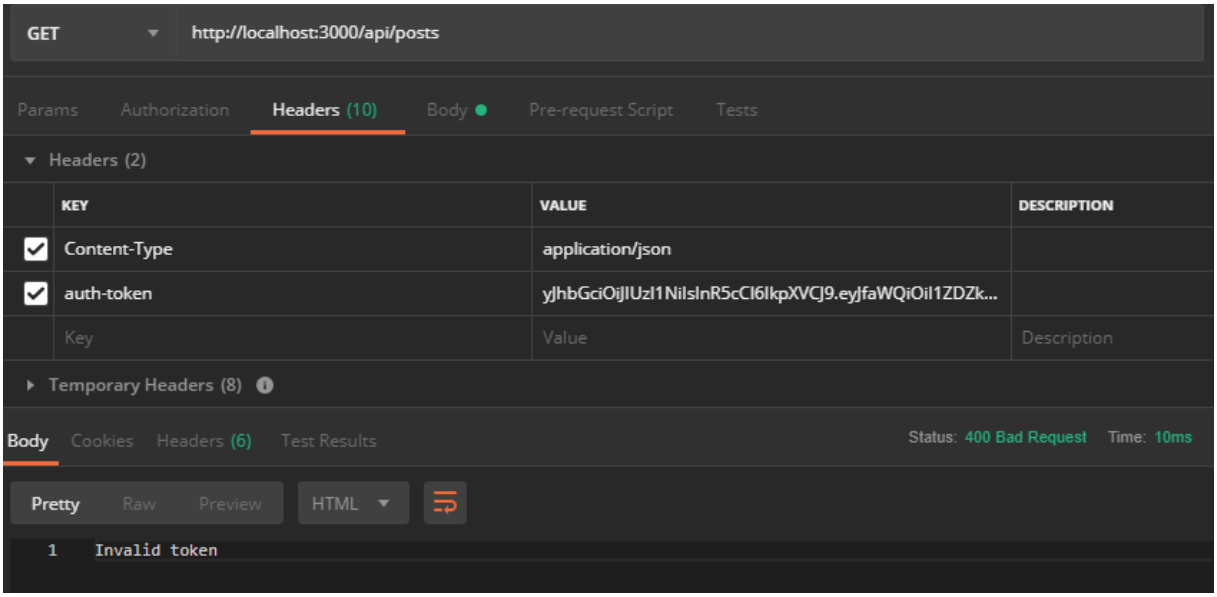
```
//token
    var h = {
        alg: "HS256",
        typ: "JWT"
    };

    var p = {
        _id: user._id,
        iat: Date.now()
    };

    function base64url(source) {
        encodedSource = CryptoJS.enc.Base64.stringify(source);

        // Remove padding equal characters
        encodedSource = encodedSource.replace(/=+$/, '');

        // Replace characters according to base64url
specifications
        encodedSource = encodedSource.replace(/\+/g, '-');
        encodedSource = encodedSource.replace(/\//g, '_');

        return encodedSource;
      }

      var stringifiedHeader =
CryptoJS.enc.Utf8.parse(JSON.stringify(h));
      var encodedHeader = base64url(stringifiedHeader);

      var stringifiedData =
CryptoJS.enc.Utf8.parse(JSON.stringify(p));
      var encodedData = base64url(stringifiedData);

      var signature = encodedHeader + "." + encodedData;
      signature = CryptoJS.HmacSHA256(signature,
process.env.TOKEN_SECRET);
```

```
        signature = base64url(signature);
        var token = encodedHeader + "." + encodedData + "." +
signature;
```

Kod 7. Ručna izrada JWT tokena

Izvor: Autor (16.9.2019.)

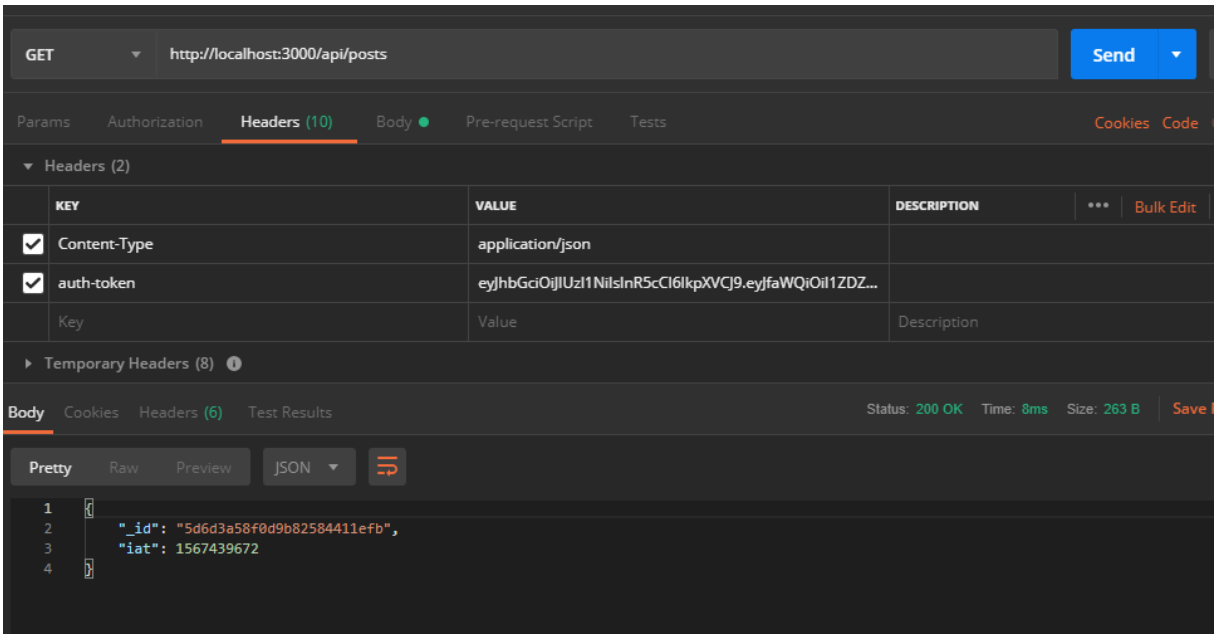
Funkcija „auth“ provjerava ispravnost tokena te ukoliko nema zaprimljen token funkcija šalje status 401 odnosno zabranjuje pristup korisniku. Ukoliko postoji token funkcija također provjerava njegovu ispravnost kako ne bi došlo do izmjena na tokenu i ukoliko token nije ispravan šalje status 400 s porukom „invalid token“ i također zabranjuje pristup korisniku. Na kraju samog projekta napravljena je privatna ruta za čiji pristup korisnik mora imati ispravan JWT token. Ukoliko korisnik ne sadrži token ili se token ne poklapa s dobivenim tokenom aplikacija će izbaciti jednu od grešaka ovisno o slučaju uzroka greške kao na primjeru prikazanom slikom 27 gdje se token ne poklapa s dobivenim tokenom.



Slika 29. Greška uzrokovana neispravnim tokenom

Izvor: Autor (1.9.2019.)

Ukoliko se priloži ispravan token i korisnik unese ispravne podatke za prijavu poput email i korisničke lozinke sustav daje povratnu informaciju koja sadrži korisnički id i vrijeme inicijalizacije tokena (iat) što je prikazano slikom 28.



Slika 30. Primjer odobrenog pristupa korisnika

Izvor: Autor (1.9.2019.)

Time je završen projektni zadatak koji je bio izrađen s ciljem prikaza praktične primjene JWT-a kod autentikacije i autorizacije korisnika.

8. ZAKLJUČAK

Iz navedenog se može izvući nekoliko važnih zaključaka. JWT je jedna od tehnologija koja iako se koristi gotovo jedno desetljeće mnogi developeri ne razumiju njene osnovne koncepte rada. To proizlazi od samog razumijevanja procesa autentikacije i autorizacije te njihovih razlika. JWT je još uvijek izvrsna tehnologija koja se koristi u mnogobrojnim sustavima, bilo u radu društvenih mreža kod prijava korisnika, zaštiti korisničkih podataka kod email usluga ili mnogobrojnim poslovnim sustavima. Njegovoj popularnosti pridonose i mnogobrojne biblioteke i paketi koji pomažu kod implementacije JWT-a u sustav, te lakšem korištenju istog. Također JWT se smatra izvrsnim za korištenje kada je pitanje sigurnosti nekog sustava. Mnogobrojni kriptografski algoritmi mogu se koristiti kod digitalnog potpisa ključa JWT tokena, što može pridonijeti tome da se osigurava kriptografska sigurnost tokena. Iako JWT dolazi s nekoliko značajnih nedostataka, od kojih se najznačajniji odnose na neznanje developera kod korištenja ove tehnologije, njegove sigurnosne prednosti kod korištenja dovoljan su razlog za njegovu popularnost što pokazuje i činjenica da u prvih dvije godine postojanja JWT-a njegova službena stranica jwt.io je imala preko četiristo tisuća pregleda. Veliki problem kod korištenja JWT-a proizlazi od samog nepoimanja osnova kriptografije i time dolazi do velikih propusta u sigurnosti prilikom lošeg kodiranja korištenjem JWT-a. Iz izrade aplikativnog dijela rada može se zaključiti kako korištenje JWT tehnologije je poduprijeto mnogobrojnim lekcijama putem besplatnih videozapisa na raznim platformama kao i mnogobrojnim dokumentacijama s više izvora poput službenih stranica JWT-a. Kriptografski dio zaštite podataka JWT-a olakšan je korištenjem biblioteka i paketa poput bcryptjs paketa za hashiranje lozinke i joi paketa za validaciju korisničkih podataka prilikom prijave u sustav. Zaključuje se kako je token bazirana autentikacija trenutno jako popularna tehnologija i toj popularnosti pridonosi developerska zajednica kreiranjem brojnih paketa za jednostavnije korištenje JWT-a. Tehnologija token bazirane autentikacije je definitivno ovdje da ostane, a u budućnosti će njena popularnost rasti čemu će pridonositi razvoj paketa za implementaciju JWT-a.

LITERATURA

Knjiga:

1. Paar, C., Pelzl, J. (2010.), *Understanding Cryptography*, Springer.

Članci u znanstvenim časopisima:

1. Ahmad, N., Wei, L. M., Jabbar, M. H. (2018.), Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array, *IOP Conf. Series: Journal of Physics: Conf. Series 1019*, IOP Publishing, str. 1-7.
2. Boneh, D. (1999.), Twenty years of attacks on the RSA cryptosystem, *American Mathematical Society (AMS),* 46 (2), str. 1-16.
3. Koblitz, N., Menezes, A. J. (2015.), The random oracle model: a twenty-year retrospective, *Journal Designs, Codes and Cryptography*, 77 (2-3), str. 1-30.

Internet izvori:

1. D.I. Management Services Pty Ltd., 2004-19., RSA signature and encryption schemes: RSA-PSS and RSA-OAEP https://www.cryptosys.net/pki/manpki/pki_rsaschemes.html (24.7.2019.)
2. Dias R., „The 5 factors of authentication“, 2017, https://medium.com/@renansdias/the-5-factors-of-authentication-bcb79d354c13 (23.4.2019.)
3. Doglio F., How to secure a REST API using JWT,2019., https://blog.logrocket.com/how-to-secure-a-rest-api-using-jwt-7efd83e71432/ (10.7.2019.)
4. Gatliff B., Encrypting data with the Blowfish algorithm, 2003., https://www.design-reuse.com/articles/5922/encrypting-data-with-the-blowfish-algorithm.html (1.9.2019.)
5. Golwalkar R., Pros and cons in using JWT (JSON WEB TOKENS), 2016., https://medium.com/@rahulgolwalkar/pros-and-cons-in-using-jwt-json-web-tokens-196ac6d41fb4 (1.9.2019.)
6. Introduction to JSON Web Tokens https://jwt.io/introduction/ (22.6.2019)

7. Khillar S., „Difference between Authentication and Authorization“, 2017, http://www.differencebetween.net/technology/difference-between-authentication-and-authorization/ (23.4.2019.)
8. Libraries for Token Signing/Verification, https://jwt.io/ (10.7.2019.)
9. Optimal asymmetric encryption padding, https://asecuritysite.com/encryption/rsa_oaep (26.7.2019.)
10. Raghuwanshi R., JWT (JSON Web Tokens) Are Better Than Session Cookies, 2017., https://dzone.com/articles/jwtjson-web-tokens-are-better-than-session-cookies?fbclid=IwAR0J2qCBPFgv5q35lqHseXKvn5VF6UqQs3WH-ggkgCKm9f0_ZIf0J9lybk8 (1.9.2019.)
11. Request for Comment (RFC) https://www.techopedia.com/definition/27929/request-for-comments-rfc (20.5.2019.)
12. Rouse M., Hash-based Message Authentication Code (HMAC), https://searchsecurity.techtarget.com/definition/Hash-based-Message-Authentication-Code-HMAC (18.7.2019.)
13. RSA Encryption, https://www.techopedia.com/definition/21852/rsa-encryption (19.7.2019.)
14. RSAES-OAEP Encryption Scheme, 2000., https://www.inf.pucrs.br/~calazans/graduate/TPVLSI_I/RSA-oaep_spec.pdf (29.7.2019.)
15. Ryck De P., The hard parts of JWT security nobody talks about, 2019. https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nobody-talks-about.html (24.6.2019)
16. Siddiqui A., „Authentication vs Authorization“, 2018, https://medium.com/datadriveninvestor/authentication-vs-authorization-716fea914d55 (23.4.2019.)
17. Singh A., JSON Web Token (JWT) with Web API, 2017., http://blogs.quovantis.com/json-web-token-jwt-with-web-api/ (09.7.2019.)
18. Siriwardena P., JWT,JWS and JWE for Not So Dummies! (part I), 2016. https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3 (22.6.2019.)

19. Stecky-Efantis M., 5 Easy Steps to Understanding JSON Web Tokens (JWT), 2016. https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec (22.5.2019.)
20. Stubbs R., Cryptographic Key Management – the Risks and Mitigation, 2018. https://www.cryptomathic.com/news-events/blog/cryptographic-key-management-the-risks-and-mitigations (24.6.2019)
21. Sullivan N., ECDSA: The digital signature algorithm of a better internet, 2014., https://new.blog.cloudflare.com/ecdsa-the-digital-signature-algorithm-of-a-better-internet/ (24.7.2019.)
22. What is an RFC?, 2018. https://flaviocopes.com/rfc/ (20.5.2019.)

Dokumentacija:

1. Bcrypt, https://www.npmjs.com/package/bcrypt (1.9.2019.)
2. Dotenv, https://www.npmjs.com/package/dotenv (27.8.2019.)
3. Express, https://www.npmjs.com/package/express (17.8.2019.)
4. JSON Web Token (JWT) RFC 7519 https://tools.ietf.org/html/rfc7519 (22.5.2019.)
5. Nodemon, https://www.npmjs.com/package/nodemon (17.8.2019.)
6. RFC 2104 - HMAC: Keyed-Hashing for Message Authentication, 1997., https://tools.ietf.org/html/rfc2104 (18.7.2019.)
7. RFC 3394 - Advanced Encryption Standard (AES) Key Wrap Algorithm https://tools.ietf.org/html/rfc3394 (29.7.2019.)
8. RFC 7518 - JSON Web Algorithms (JWA), 2015., https://tools.ietf.org/html/rfc7518#section-3 (17.7.2019.)
9. RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2, 2016., https://tools.ietf.org/html/rfc8017 (20.7.2019.)

POPIS SLIKA

POPIS TABLICA

POPIS KODOVA

SAŽETAK

Ovim radom istraživala se tehnologija json web tokena i njegova upotreba. Također definirani su pojmovi autentikacije i autorizacije koji su osnova kod korištenja JWT-a. Nadalje detaljno su opisane kriptografske metode korištene kod digitalnog potpisa i šifriranja ključa JWT-a. Razrađene su prednosti i nedostaci korištenja tehnologije JWT-a koje su detaljno poduprijete primjerima i načinu zaobilaska istih. Zadnji dio rada fokusirao se na izradu jednostavne aplikacije koja prikazuje upotrebu JWT-a kod autentikacije i autorizacije. Za izradu aplikacije korišteno je NodeJs okruženje, te mnoge biblioteke i paketi koji pomažu lakšoj implementaciji JWT-a poput bcrypt, nodemon, joi i drugi.

Ključne riječi: JWT, Json Web Token, autentikacija , autorizacija, nodejs, kriptografija

ABSTRACT

This work explores the technology of json web tokens and its use. Authentication and authorization concepts that are the basis for using JWT are also defined. The cryptographic methods used for digitally signing and encrypting the JWT key are further described. The advantages and disadvantages of using JWT technology have been elaborated, which are backed up in detail by examples and how to avoid them. The last part of the paper focused on creating a simple application that demonstrates the use of JWT for authentication and authorization. NodeJs environment was used to build the application, as well as many libraries and packages to help facilitate JWT implementation such as bcrypt, nodemon, joi and others.

Key words: JWT, Json Web Token, authentication, authorization, nodejs, cryptography