Usporedba generatora statičnih stranica i CMS sustava

Kojić, Ivan

Undergraduate thesis / Završni rad

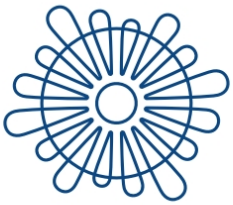
2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zadar / Sveučilište u Zadru**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:162:474007

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-26**



Sveučilište u Zadru
Universitas Studiorum
Jadertina | 1396 | 2002 |

Repository / Repozitorij:

University of Zadar Institutional Repository

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Zadru

Preddiplomski stručni studij Informacijske tehnologije

Ivan Kojić

Usporedba generatora statičnih stranica i CMS sustava

Završni rad

Zadar, 2023.

Sveučilište u Zadru

Preddiplomski stručni studij Informacijske tehnologije

Usporedba generatora statičnih stranica i CMS sustava

Završni rad

Student/ica:
Ivan Kojić

Mentor/ica:
Niko Vrdoljak

Zadar, 2023.

Izjava o akademskoj čestitosti

Ja, **Ivan Kojić,** ovime izjavljujem da je moj **završni** rad pod naslovom **Usporedba generatora statičnih stranica i CMS sustava** rezultat mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na izvore i radove navedene u bilješkama i popisu literature. Ni jedan dio mojega rada nije napisan na nedopušten način, odnosno nije prepisan iz necitiranih radova i ne krši bilo čija autorska prava.

Izjavljujem da ni jedan dio ovoga rada nije iskorišten u kojem drugom radu pri bilo kojoj drugoj visokoškolskoj, znanstvenoj, obrazovnoj ili inoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga i nakon obrane uređenoga rada.

Zadar, 24. rujna 2023.

SADRŽAJ

1 Uvod ... 1

2 Generatori statičnih stranica ... 2

2.1 Rani početci generatora statičnih stranica ... 4

2.2 Generatori statičnih stranica u suvremenom tehnološkom okruženju ... 5

3 CMS sustavi ... 7

3.1 Rani početci CMS sustava ... 9

3.2 CMS sustavi u suvremenom tehnološkom okruženju ... 10

4 Razvoj ... 16

4.1 Tehnologije korištene za razvoj ... 16

4.1.1 Python ... 16

4.1.2 Flask ... 18

4.1.3 Pelican ... 19

4.1.4 Jinja2 ... 20

4.1.5 MySQL ... 21

4.2 Razvoj web portala i pripadajućeg CMS sustava ... 22

4.2.1 Flask nacrti ... 22

4.2.2 Klase modela ... 24

4.2.3 Obrada grešaka ... 33

4.2.4 Flask forme ... 35

4.2.5 Prijava i autorizacija korisnika ... 43

4.2.6 Dizajn korisničkog sučelja ... 47

4.3 Razvoj web portala s Pelican generatorom statičnih stranica ... 49

4.3.1 Dizajn korisničkog sučelja ... 52

4.3.2 Dodatci ... 55

5 Usporedba ... 57

5.1 Prednosti i mane CMS sustava 57
5.2 Prednosti i mane generatora statičnih stranica 58
6 Zaključak 60
7 Popis literature 61
8 Prilozi 65

Sažetak

U današnjem digitalnom dobu uočljiva je rastuća potreba tvrtki, organizacija i pojedinaca za objavljivanjem i upravljanjem sadržajem. Ovaj izazov današnjice stvara potrebu za odgovorom na ključno pitanje, a to je koji je najoptimalniji i najpraktičniji način objavljivanja i upravljanja sadržajem. Istraživanjem načina i tehnologija dolazi se do dvije najzastupljenije opcije – CMS sustavi i generatori statičnih stranica. Ovaj završni rad analizira i uspoređuje ove dvije vodeće strategije upravljanja sadržajem detaljno razmatrajući njihove prednosti i ograničenja, a kako bi se osigurala precizna usporedba razvijena su dva identična portala čija izrada pruža dublji uvid u situacije u kojima je optimalnije koristiti CMS sustav ili generatore statičnih stranica.

Ključne riječi: generatori statičnih stranica, CMS sustavi, Python, Flask, Pelican

1 Uvod

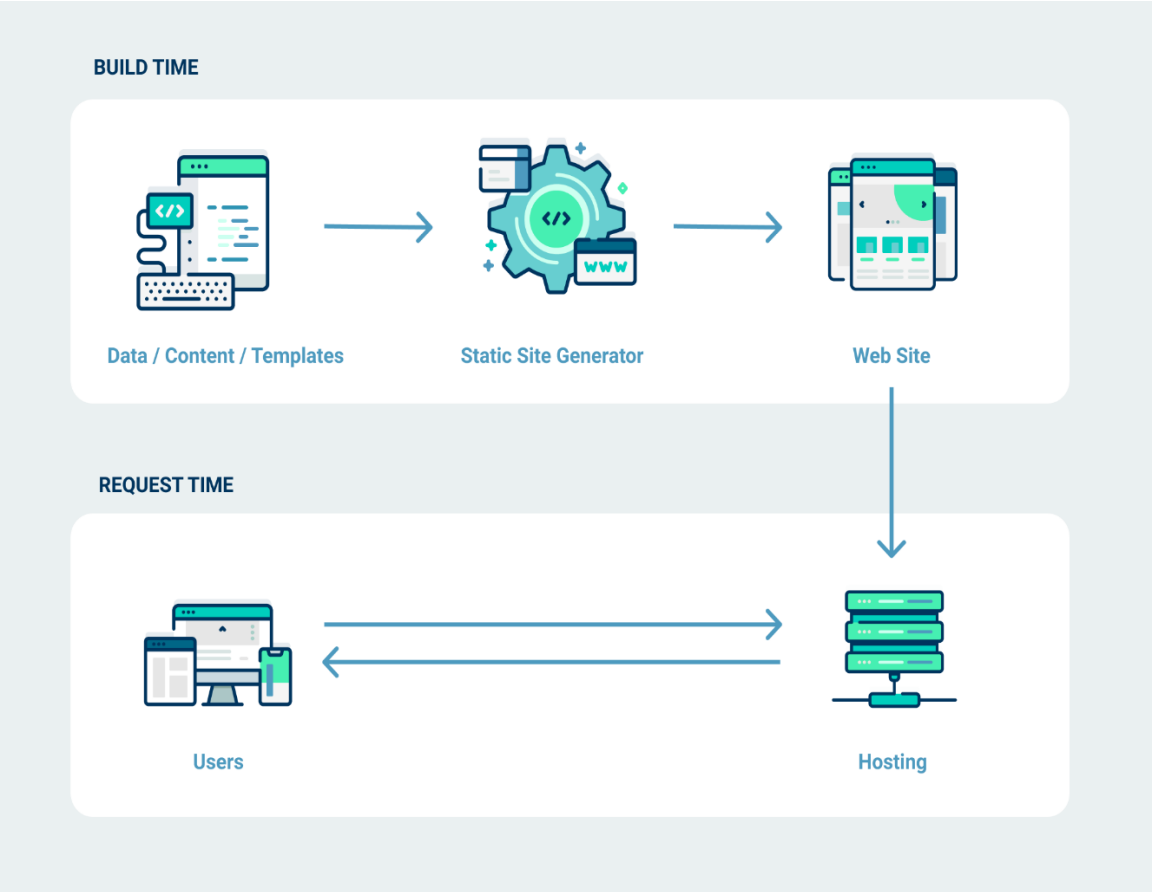
S obzirom na značajan napredak i sveprisutnost tehnologije, internet je evoluirao u nezaobilaznu i primarnu platformu za komunikaciju, informiranje i izražavanje. Iz tog razloga, posjedovanje vlastite web stranice ili web portala postalo je od ključne važnosti, kako u osobnom, tako i u poslovnom kontekstu. Užurbanost života dovela je ljude do točke kada sve žele i mogu raditi iz udobnosti vlastitog doma. Razgovor, kupnja, čitanje vijesti, sve su to stvari koje su nam dostupne u online obliku. U tom moru sadržaja javlja se želja i potreba za vlastitim web portalom, bila to online trgovina, portal s vijestima, osobni blog portal ili nešto sasvim drugačije. Objavljivanjem vlastitog sadržaja proširujemo svoja poznanstva i gradimo nove poslovne prilike. Međutim, kako bi novi sadržaj bio objavljen na vrijeme i na što bolji i pristupačniji način, potrebno je odabrati odgovarajuću tehnologiju, odnosno način objavljivanja i upravljanja sadržajem. Istraživanjem tehnologija i načina upravljanja i objavljivanja sadržaja može se zaključiti kako su generatori statičnih stranica te CMS sustavi najrasprostranjeniji u ovom području. Stoga je cilj ovog rada upoznavanje s navedenim alatima i tehnologijama, kao i analiza prednosti i mana oba pristupa. S željom da se najpreciznije ispuni cilj ovog rada, odnosno usporedba tih tehnologija i načina objavljivanja sadržaja, nisu odabrana gotova rješenja već su napravljena dva identična personalizirana web portala na kojima se može objavljivati vlastiti sadržaj.

U teorijskom dijelu ovog rada istražena je povijest i budućnost te načini na koji se koriste generatori statičnih stranica i CMS sustavi, dok praktični dio rada prati razvoj dva identična web portala na kojima se objavljuje sadržaj, jedan izradom vlastitog CMS sustava, a drugi izradom pomoću generatora statičnih stranica. Analiza same usporedbe detaljnije je obrađena u poglavlju Usporedba, gdje su sažete prednosti i nedostaci oba sustava za objavljivanje sadržaja i izradu web stranica. Zaključak istraživanja sintetizira ključne razlike između navedenih pristupa, uzimajući u obzir specifične potrebe i ciljeve projekta. Ova analiza također će biti od neprocjenjive važnosti za profesionalne web developere, ali i za korisnike s bazičnim tehničkim znanjem koji se suočavaju s izazovom odabira između generatora statičnih stranica i CMS sustava.

2 Generatori statičnih stranica

U kontekstu alternativa tradicionalnim sustavima za upravljanje sadržajem (CMS), tijekom posljednjih nekoliko godina uočava se nagli porast popularnosti generatora statičnih stranica. Generatori statičnih stranica predstavljaju alate koji se koriste za generiranje statičnih stranica temeljenih na kombinaciji seta podataka i seta predložaka[1]. Naime, statične web stranice sastavljene od čistog HTML-a, CSS-a i JavaScript-a generiraju se unaprijed te su odmah dostupne korisniku kada ih zatraži. Stranicama generiranima na takav način nije potrebna baza podataka kao ni server poslužitelj na kojem se vrti *backend* kod. Nadalje, s obzirom da su takve statične stranice unaprijed spremne, fokus s *request* vremena se prebacuje na *build* vrijeme[2], što ih čini atraktivnom opcijom u modernom okruženju web razvoja.

[1] Cloudflare, What is a static site generator?, Dostupno na: https://www.cloudflare.com/learning/performance/static-site-generator/ [Pristupljeno 15.07.2023.]
[2] Hawksworth P., 2020., Netlify, What is a Static Site Generator? And 3 ways to find the best one, Dostupno na: https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/ [Pristupljeno 16.07.2023.]



Slika 1: Prikaz toka izrade web stranice putem generatora statičnih stranica

Dostupno na: https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/

2.1 Rani početci generatora statičnih stranica

Iako su svoju popularnost stekli unazad nekoliko godina, generatori statičnih stranica postali su dostupni već početkom 2000-ih godina, a njihovi temelji i prvi koraci prema punim verzijama javili su se čak i krajem 1990-ih godina[3].

Jedan od najbližih prethodnika alatima za generiranje statičnih web stranica koji se prvi pojavljuje je HSC (*HTML Sucks Completely*), stvoren 1996. godine od strane autora Thomasa Aglassingera. Iako HSC zapravo nije bio generator statičnih stranica, imao je nekoliko značajki tipičnog generatora statičnih stranica. Glavna svrha HSC-a bila je rješavanje određenih problema s kojima se programeri susreću u radu s HTML-om, stoga je imao mogućnost uključivanja (*eng. includes*) i uvjetnih izjava. Također, sadržavao je čak i značajke koje su netipične za generatore statičnih stranica, kao što su provjera sintakse i automatsko dodavanje visine i širine slikovnih elemenata.

Nakon HSC-a pojavljuje se još jedan alat koji je prepoznao prednosti statičnih stranica. Naime, 2001. godine Ben i Men Trott predstavili su Movable Type, vlastitu open-source platformu za objavljivanje blogova temeljenu na Perl programskom jeziku. Važno je napomenuti da se spomenuta platforma nije koristila putem konzole ili naredbenog retka (engl. command line), već putem web grafičkog sučelja. Iako ovo nije uobičajeno za generatore statičnih stranica, rezultirajuća stranica bila je upravo statična. Movable Type se i dalje aktivno održava te je i danas pokretač brojnih blogova, web stranica i web portala[4].

Još jedan od pionira generatora statičnih stranica, Denis Defreyne, prepoznao je problem brzine, odnosno nedostatka iste kod CMS- a baziranog na Ruby on Rails programskom jeziku na serverima s manje resursa. Stoga je Defreyne, samo godinu dana nakon što je lansiran Ruby on Rails programski jezik, predstavio Nanoc. On je posjedovao mnoge karakteristike današnjih generatora statičnih stranica, poput podrške za Markdown, rasporede (eng. *layouts*), metapodatke i predloške[5].

[3] Neumegen M., 2022., Cloudcannon, The 'Before Jekyll' era, Dostupno na: https://cloudcannon.com/blog/ssg-history-1-before-jekyll/ [Pristupljeno 16.07.2023.]
[4] MovableType, WHAT IS MOVABLE TYPE?, Dostupno na: https://www.movabletype.org/ [Pristupljeno 16.07.2023.]
[5] Nanoc, Why Nanoc?, Dostupno na: https://nanoc.app/about/ [Pristupljeno 17.07.2023.]

2.2 Generatori statičnih stranica u suvremenom tehnološkom okruženju

Sve do 2008. godine statične stranice su se i dalje smatrale zastarjelom tehnologijom, dok su CMS sustavi ostali preferirani izbor korisnika za objavljivanje sadržaja. Međutim, tada je Tom Preston-Werner, američki razvojni inženjer i poduzetnik, prvo lansirao platformu GitHub, a u prosincu iste godine predstavio Jekyll, jednostavan generator statičnih stranica usmjeren prema blogovima. Jekyll je također napisan u programskom jeziku Ruby on Rails te dijeli mnoge sličnosti s Nanoc-om. Međutim, unatoč tim sličnostima, Jekyll je uveo određene novine i značajke poput metapodataka u YAML isječku i pojednostavljeno objavljivanje sadržaja. Naime, ukoliko korisnik želi nešto objaviti, sve što je potrebno je postaviti datoteku napisanu u Markdown-u u odgovarajući direktorij te će ta datoteka postati objava na njegovom blogu. No, ono što je uvelike pomoglo pogurati Jekyll u svijetu statičnih stranica te generatora istih je lansiranje GitHub Pages. GitHub Pages besplatna je hosting platforma lansirana od strane GitHub-a koja služi upravo za hosting stranica napravljenih Jekyll-om. Jekyll je do danas promijenio nekoliko vodećih održavatelja te je trenutačno na verziji 4.3.2[6]. Jekyll je bio prekretnica u svijetu statičnih stranica te generatora istih. Statične stranice počele su dobivati sve više pažnje te su se sve manje smatrale ograničenom tehnologijom[7].

Nakon Jekyll-a pojavili su se mnogi generatori statičnih stranica koji su dodavali neke nove značajke na već postojeće ili uklanjali i poboljšavali one koje su predstavljale problem u modernom razvitku statičnih stranica. Jedan od njih je i Pelican generator statičnih stranica objavljen 2010. godine od strane Alexisa Metaireaua. Naime, Pelican dijeli mnoge iste i/ili slične značajke s Jekyll-om, ali u nekim aspektima ih čak i nadmašuje. Prijevodi, paginacija i podrška za Atom i RSS Feed jezgrene su funkcionalnosti Pelicana, dok navedeno može biti Jekyll-ov problem koji zahtjeva kompleksnije rješenje.

Budućnost generatora statičnih stranica nije ni jednostavna ni jednosmjerna. Povijest njihovog razvoja svodi se na dodavanje novih ideja i značajki, pa na njihovo pojednostavljenje te ponavljanje cijelog ovog ciklusa.

[6] Jekyll, Jekyll Simple, blog-aware, static sites, Dostupno na: https://jekyllrb.com/ [Pristupljeno 20.07.2023.]
[7] Neumegen M., 2022., Cloudcannon, The 'After Jekyll' era, Dostupno na: https://cloudcannon.com/blog/ssg-history-2-after-jekyll/ [Pristupljeno 21.07.2023.]

Danas je situacija takva da generatori statičnih stranica idu u jednom od četiri navedena smjera:

1. Čisti – Jekyll, Hugo, 11ty – s fokusom na generiranju čistog HTML-a i jednostavnosti
2. Djelomična hidracija – Elder.js, Astro, Slinkity – s fokusom na pronalazak ravnoteže između brzine i dinamičkih interakcija
3. SPA – Next.js, Nuxt, Sveltekit – s fokusom na približavanje granica statičnog i dinamičkog
4. Fullstack – Redwood.js, Blitz.js – sve potrebno za izgradnju modernih web aplikacija

Svaki od navedenih smjerova ima svoje prednosti i kompromise te je bolja opcija od drugog za različite primjene. Trenutačno se generatori statičnih stranica ne koriste samo za izradu blogova i web portala, već nude mogućnost razvoja od jednostavnih web stranica do složenih web aplikacija.

3 CMS sustavi

CMS sustavi (eng. *Content Managment System*), odnosno sustavi za upravljanje sadržajem softverski su alati i aplikacije koje korisnicima omogućuju olakšano i pojednostavljeno upravljanje sadržajem web stranice ili web portala. Također, omogućuju korisnicima kreiranje, uređivanje i upravljanje sadržajem, kao i upravljanje svim drugim aspektima njihove web stranice bez potrebe za tehničkim ili programerskim znanjem. Zahvaljujući svojim značajkama i mogućnostima koje omogućuju korisnicima veći fokus na sadržaj, a manji na tehničke aspekte, generatori statičnih stranica već godinama igraju ključnu ulogu u razvoju i održavanju web sadržaja.

CMS sustavi nemaju čvrsto određenu definiciju, već se mogu protumačiti na više načina ovisno o raznim scenarijima, ciljevima kreatora ili projekta. Organizacija pod nazivom AIIM (Udruga za upravljanje informacijama i slikama) (eng. *Association for Information and Image Management*), danas poznata kao AIIM (Udruga za inteligentno upravljanje informacijama) (eng. *Association for Intelligent Information Management*), prisvojila je akronime ECM(Upravljanje sadržajem tvrtke) (eng. *Enterprise Content Managment*) i WCM (Upravljanje web sadržajem) (eng. *Web Content Managment*) te odredila vlastite definicije[8].

Godine 2010., ECM bio je definiran kao skup strategija, metoda i alata korištenih za bilježenje, upravljanje, pohranu, čuvanje i isporuku sadržaja i dokumenata povezanih s organizacijskim procesima te su obuhvaćali upravljanje informacijama unutar cijelog poduzeća, bilo da se radi o papirnatim dokumentima, računalnim datotekama, bazama podataka ili e-pošti. Nedugo zatim, 2017. godine, AIIM organizacija predlaže da se pojam ECM zamijeni s pojmom IIM[9]. IIM je definiran kao skup strategija, metoda i alata namijenjenih za stvaranje, bilježenje, automatizaciju, isporuku, osiguranje i analizu sadržaja i dokumenata povezanih s organizacijskim procesima te se odnosi na upravljanje sadržajem i podacima, a ne samo sadržajem[10].

[8] Kohan B., Comentum, What is a Content Management System (CMS)?, Dostupno na: https://www.comentum.com/what-is-cms-content-management-system.html [Pristupljeno 15.08.2023.]
[9] Wikipedia, Enterprise content managment, Dostupno na: https://en.wikipedia.org/wiki/Enterprise_content_management [Pristupljeno 15.08.2023.]
[10] Mancini J., 2017., The next wave: Moving from ECM to Intelligent Information Management, Dostupno na: https://cdn2.hubspot.net/hubfs/332414/AIIM_Blog/Intel-info-Next-Wave-2017-updated.pdf [Pristupljeno 15.08.2023.]

WCM (Web Content Managment) definiran je kao proces kontroliranja sadržaja koji se koristi na jednom ili više online kanala uporabom komercijalnih, *open-source* ili personaliziranih alata, dok je WCMS (Web Content Managment System) softver za upravljanje isključivo web sadržaja[11].

U svijetu web developmenta te razvoja web stranica i aplikacija, najčešće se koristi izraz CMS, koji je u svojoj namjeni najsličniji definiciji WCM-a. S obzirom da u suštini svi dijele ključne značajke, CMS sisteme možemo definirati kao aplikacije, ne nužno web aplikacije, koje omogućuju korisnicima s različitim pravima upravljanje sadržajem na web stranici bez potrebe znanja HTML-a[12].

S tehničkog stajališta CMS sustavi se sastoje od dva ključna dijela:

- Aplikacija za upravljanje sadržajem (eng. *Content Managment Application*) – CMA – dio je CMS sistema koji korisnicima omogućuje dodavanje i upravljanje sadržajem. Ovaj dio možemo gledati kao *frontend* dio koji uključuje forme za unos teksta, galerije slika i slično.
- Aplikacija za dostavu sadržaja (eng. *Content Delivery Application*) – CDA – ovaj dio CMS sistema obuhvaća pozadinske procese koji obrađuju i pohranjuju uneseni sadržaj te ga čine vidljivim posjetiteljima. Ovaj dio možemo gledati kao *backend* dio.

Skupa navedeni dijelovi čine cjelinu koja korisnicima, paralelnim radom i postojanjem, čini održavanje web stranica jednostavnim[13].

Bez obzira kako nazvali, podijelili ili definirali CMS sustave, svi imaju iste ključne i jezgrene funkcionalnosti i značajke. Iako značajke mogu varirati između različitih CMS sustava, one jezgrene uključuju indeksiranje, pretraživanje i dohvaćanje sadržaja, upravljanje formatima te kontrolu revizijama (eng. *revision*).

[11] Gartner Glossary, Web Content Management (WCM), Dostupno na: https://www.gartner.com/en/information-technology/glossary/wcm-web-content-management [Pristupljeno 16.08.2023.]
[12] Kohan B., Comentum, What is a Content Management System (CMS)?, Dostupno na: https://www.comentum.com/what-is-cms-content-management-system.html [Pristupljeno 15.08.2023.]
[13] Kinsta, What is Content Management System (CMS)?, Dostupno na: https://kinsta.com/knowledgebase/content-management-system/ [Pristupljeno: 16.08.2023.]

Pored navedenih značajki postoje i mnoge druge koje su stekle svoju popularnost, kao što su:

- SEO-*friendly* URL-ovi
- Diskusijski forumi
- Sustav dozvola
- Podrška za predloške i prilagodljive predloške
- Admin panel
- Podrška za više jezika
- Integrirani upravitelji datotekama
- Integrirane evidencije revizija
- Sigurnosne mjere i autentifikacija[14]

3.1 Rani početci CMS sustava

Na samom početku weba i razvoja web stranica sve su web stranice bile statične. Proces promjene sadržaja bio je dugotrajan jer je zahtijevao ručno uređivanje, dok je dodavanje nove stranice značilo ponovno pisanje koda iz početka. Osim što je zahtijevalo znatne vremenske i ljudske resurse, također je zahtijevalo duboko razumijevanje tehnologije, a upravo zbog tih izazova pojavila se potreba za rješenjima kao što su CMS sustavi.

FileNet, osnovan 1985. godine, smatra se prvim sustavom koji je bio pravi sustav za upravljanje sadržajem, odnosno CMS sustav. Naime, s obzirom da je to razdoblje obilježeno i samim početkom razvoja weba, prva značajna promjena u razvoju CMS sustava događa se 1995. godine kada je FileNet predstavio skup programskih alata za obradu i upravljanje dokumentima i radnim procesima. Krajem 1995. godine pojavljuje se i Vignette kojem se često pripisuje da je izmislio pojam *Content Managment System*, odnosno sustav za upravljanje sadržajem (CMS sustav). Cilj Vignette-a bio je olakšati i pojednostaviti objavljivanje sadržaja te personaliziranje tog sadržaja. Godinu dana nakon, 1996. godine, Vignette predstavlja StoryBuilder. Tijekom tih godina pojavljuju se i prvi poslovno orijentirani CMS sustavi, odnosno CMS sustavi za poduzeća kao što su Interwoven iz 1995. godine, Documentum, FatWire, FutureTense i Inso iz 1996. godine te EPiServer iz 1997. godine.

[14] Amsler S., Churchville F., 2021., Techtarget, Definition – content management system(CMS), Dostupno na: https://www.techtarget.com/searchcontentmanagement/definition/content-management-system-CMS [Pristupljeno 17.08.2023.]

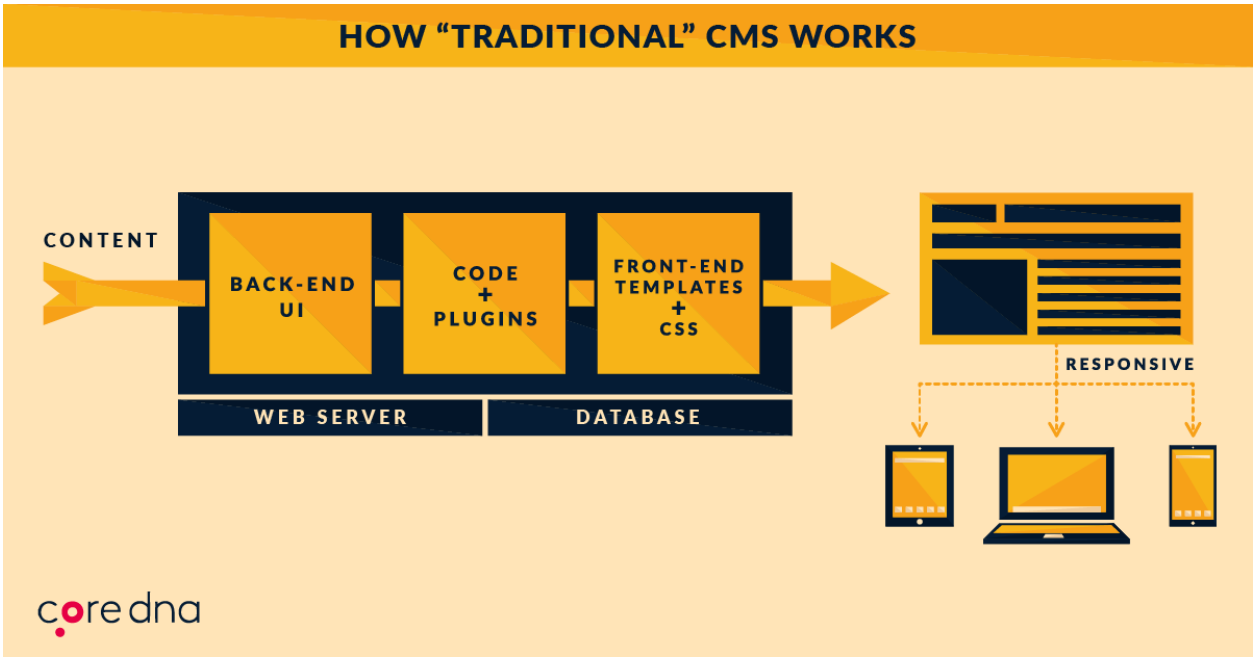
Početkom 2000.-ih godina CMS sustavi dominirali su svijetom web-a, a upravo tada se pojavljuju i open-source CMS sustavi koji su korisnicima nudili besplatne verzije CMS sustava. Tu su se isticali OpenCMS, PHP-Nuke, WordPress, Drupal, Joomla, a najviše pažnje privukao je WordPress, koji se fokusirao na blog sadržaj, ali je također omogućavao i personalizaciju te su developeri imali mogućnost stvaranja vlastitih dodataka i ekstenzija (eng. *extensions*).

Tijekom 2003. godine počinju se pojavljivati i takozvani *website builders,* CMS sustavi koji su nudili unaprijed izrađene predloške. Oni su korisnicima omogućavali izradu i održavanje vlastitih web stranica bez ikakvog znanja i poznavanja HTML-a, CSS-a ili nekog programskog jezika. Najistaknutiji iz tog perioda su WordPress i SquareSpace iz 2003. godine te Weebly i Wix iz 2006. godine[15].

3.2 CMS sustavi u suvremenom tehnološkom okruženju

Prvi CMS sustavi bili su monolitni sustavi (eng. *monolithic systems*), što znači da su na jednom mjestu sadržavali sve potrebno za upravljanje i održavanje sadržaja. Naime, to su aplikacije koje čine cjelinu te rade kao jedna aplikacija[16].

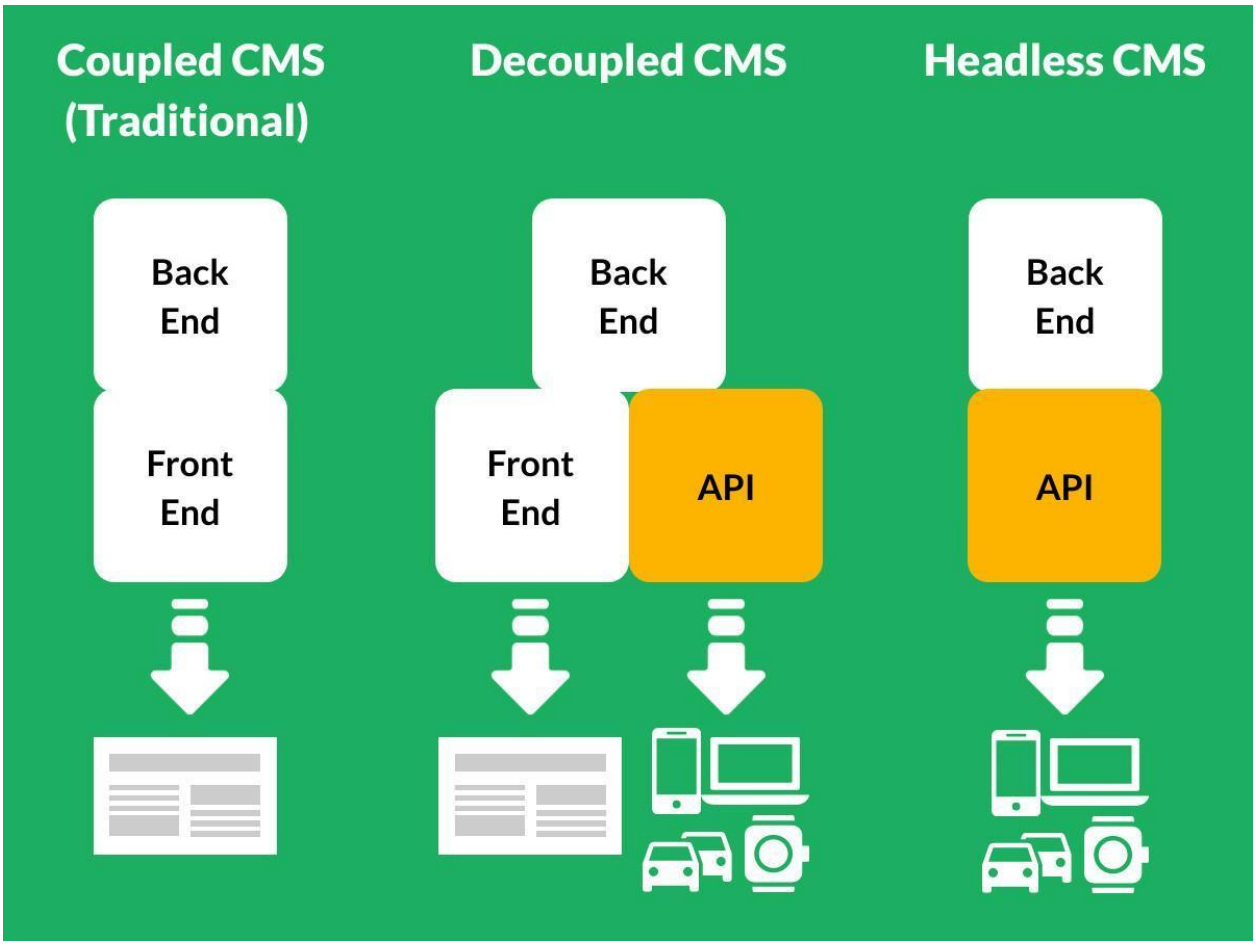
[15] Heslop B, 2018., Contentstack, History of Content Management Systems and Rise of Headless CMS, Dostupno na: https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms [Pristupljeno: 18.08.2023.]
[16] Wikipedia, Monolithic system, Dostupno na: https://en.wikipedia.org/wiki/Monolithic_system [Pristupljeno 18.08.2023.]



Slika 2 Ilustracija prikaza rada tradicionalnih CMS sustava, Dostupno na: https://www.linkedin.com/pulse/headless-cms-vs-decoupled-explained-5-minutes-sam-saltis/

Razvojem tehnologija na tržištu se pojavljuju brojni komunikacijski kanali koji zahtijevaju isti sadržaj, a to predstavlja problem monolitnoj arhitekturi tradicionalnih CMS sustava. No, kako bi se riješio ovaj izazov pojavljuju se Decoupled i Headless CMS rješenja[17].

[17] Heslop B, 2018., Contentstack, History of Content Management Systems and Rise of Headless CMS, Dostupno na: https://www.contentstack.com/blog/all-about-headless/content-management-systems-history-and-headless-cms



Slika 3 Ilustracija pojednostavljenih razlika između tradicionalnog, Decoupled i Headless CMS sustava, Dostupno na: https://www.justrelate.com/headless-cms-and-decoupled-cms-explained-52faddc2d9e51b07

Decoupled CMS sustavi sastoje se od dva ili više sustava koji međusobno komuniciraju bez međusobne povezanosti. Takva arhitektura CMS sustava developerima omogućuje mijenjanje prezentacijskog i logičkog sloja bez utjecaja na sami sadržaj.

Glavne komponente Decoupled CMS sustava su:

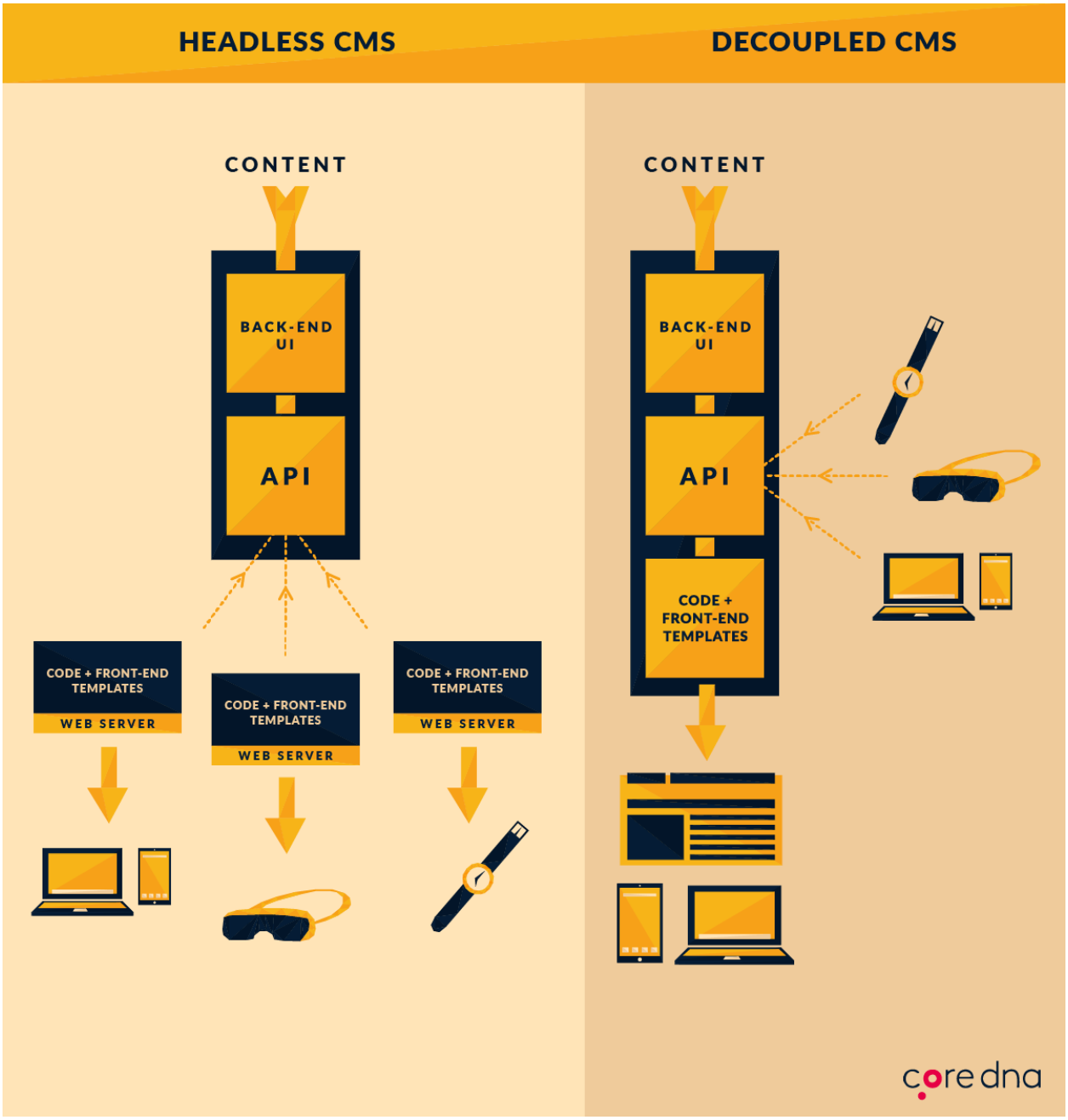
- Baza podataka
- *Backend* aplikacija
- API – poveznica između *backend* i *frontend* aplikacija
- *Frontend* aplikacija

Razdvajanjem CMS sustava na ovakav način developeri imaju potpunu slobodu nad razvitkom *backend* i *frontend* aplikacija[18].

[18] Optimizely, Decoupled CMS, Dostupno na: https://www.optimizely.com/optimization-glossary/decoupled-cms/#:~:text=What%20is%20decoupled%20CMS%3F,website%20or%20app%20users%20see. [Pristupljeno 18.08.2023.]

Headless CMS sustavi koriste *API-first* princip, što znači da nemaju prezentacijski sloj, a možemo ih gledati kao samo *backend* dio klasičnog CMS sustava. Također, Headless CMS sustavi isporučuju sadržaj putem API-ja (sučelja za programiranje aplikacija) (eng. *Application Programming Interface*). Prezentacijski sloj, odnosno *frontend* aplikacija, sadržaju pristupa putem HTTP zahtjeva što omogućuje veliku fleksibilnost, skalabilnost i prilagodbu.

Iako mnogi zamjenjuju značenje Decoupled i Headless CMS sustava, oni se razlikuju u jednoj osnovnoj stavki, a to je da Decoupled CMS sustavi imaju prezentacijski sloj, dok Headless CMS sustavi nemaju.[19]



Slika 4 Ilustracija razlika između Headless i Decoupled CMS sistema, Dostupno na: https://www.linkedin.com/pulse/headless-cms-vs-decoupled-explained-5-minutes-sam-saltis/

[19] Butti R, Storyblok, Headless CMS Explained: What is it? Why does it matter?, Dostupno na: https://www.storyblok.com/tp/headless-cms-explained#headless-cms-explained-what-is-it-why-does-it-matter [Pristupljeno 18.08.2023.]

Svakodnevno svjedočimo porastu broja komunikacijskih kanala i medija te sve većoj potrebi za objavljivanjem sadržaja na raznim platformama, kao što su web stranice i portali, igraće konzole, pametni satovi, glasom-aktivirani uređaji i slično. No, kao odgovor na ove izazove, budućnost CMS sustava jasno ukazuje na usmjerenost prema Headless CMS sustavima, dok će Decoupled CMS sustavi također imati svoju ulogu.

4 Razvoj

Kako bi se postigla što preciznija usporedba, razvijene su dvije web aplikacije s istim primarnim ciljem - objavljivanjem web sadržaja na vlastitom portalu. Nadalje, kako bi se simulirali stvarni zahtjevi korisnika, nisu odabrana gotova rješenja, teme ni predlošci. Također, odabir tehnologija za izradu personaliziranog web portala, bilo putem personaliziranog CMS sustava ili generatora statičnih stranica, ovisi o krajnjim zahtjevima i postavljenim ciljevima.

4.1 Tehnologije korištene za razvoj

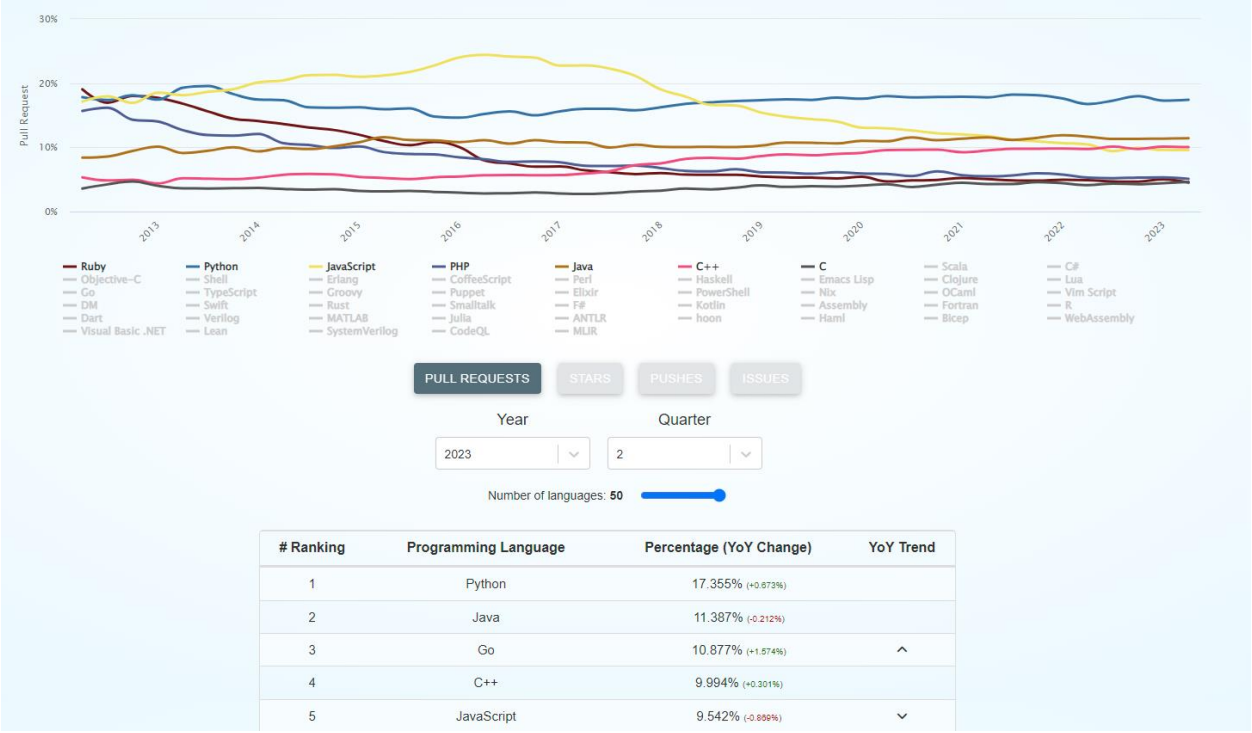
4.1.1 Python

Za izradu obje web aplikacije u okviru ovog rada odabran je Python programski jezik. Python, koji je stvorio Guido van Rossum 1991. godine[20], jest interpretirani i visokorazinski programski jezik s objektno orijentiranom paradigmom i dinamičkom sintaksom. Njegove ugrađene visokorazinske strukture podataka, zajedno s dinamičkim tipiziranjem i dinamičkim vezivanjem, čine ga iznimno privlačnim za brzu izradu aplikacija te za korištenje kao skriptni jezik za povezivanje postojećih komponenti. Nadalje, ovaj programski jezik ističe se po jednostavnoj i lako čitljivoj sintaksi koja potiče čitljivost koda i smanjuje troškove održavanja programa. Dodatno, podržava module i pakete, potičući modularnost i ponovnu upotrebu koda[21]. Python se često koristi za izgradnju web stranica i softvera, automatizaciju zadataka te provođenje analize podataka. TIOBE Programming Community Indeks, pokazatelj popularnosti programskih jezika, stavlja ga na prvo mjesto[22], čemu se pridružuje i statistika GitHut-a, odnosno statistika programskih jezika na GitHub-u, također s prvim mjestom[23].

[20] Wikipedia, Python (programming language), Dostupno na: https://en.wikipedia.org/wiki/Python_(programming_language) [Pristupljeno 02.09.2023.]
[21] Python, What is Python? Executive Summary, Dostupno na: https://www.python.org/doc/essays/blurb/ [Pristupljeno 02.09.2023.]
[22] TIOBE, TIOBE Index for September 2023, Dostupno na: https://www.tiobe.com/tiobe-index/ [Pristupljeno 02.09.2023.]
[23] GitHut 2.0, Dostupno na: https://madnight.github.io/githut/#/pull_requests/2023/2 [Pristupljeno 02.09.2023]

Sep 2023	Sep 2022	Change	Programming Language	Ratings	Change
1	1		Python	14.16%	-1.58%
2	2		C	11.27%	-2.70%
3	4	▲	C++	10.65%	+0.90%
4	3	▼	Java	9.49%	-2.23%
5	5		C#	7.31%	+2.42%
6	7	▲	JavaScript	3.30%	+0.48%
7	6	▼	Visual Basic	2.22%	-2.18%
8	10	▲	PHP	1.55%	-0.13%
9	8	▼	Assembly language	1.53%	-0.96%
10	9	▼	SQL	1.44%	-0.57%

Slika 5 TIOBE Programming Community Index, Dostupno na: https://www.tiobe.com/tiobe-index/



Slika 6 GitHut 2.0 Dostupno na: https://madnight.github.io/githut/#/pull_requests/2023/2

4.1.2 Flask

Iako je nastao kao prvotravanjska šala Armina Ronachera 2010. godine, Flask je postao jedan od ključnih alata za razvoj web aplikacija[24]. Flask predstavlja Python okvir (eng. *framework*) koji se zbog svoje jednostavnosti i skalabilnosti često naziva i mikro okvirom (eng. *microframework*). Ono što ga čini posebnim i uspješnim je upravo njegova prilagodljivost. Naime, on ne nameće određenu strukturu projekta kao većina okvira za izradu web aplikacija, pa developeri imaju potpunu slobodu u odabiru alata i biblioteka koje će koristiti u svojim projektima. Nadalje, Flask nema integriran ORM (eng. *Object Relational Mapper*) ni slične značajke, već nudi one osnovne poput usmjeravanja URL-ova (eng. *URL Routing*) i sustava za predloške (eng. *Template engine*)[25]. Također, podržava i oslanja se na mnoštvo ekstenzija potrebnih za razne zahtjeve u razvoju web aplikacija. Ekstenzije dodaju funkcionalnost aplikaciji kao da su implementirane u samom Flask-u, pa se tako mogu pronaći ORM ekstenzije, ekstenzije za provjeru obrazaca (eng. *Forms*), obradu prijenosa datoteka, razne ekstenzije za autentifikaciju i slično. Samu popularnost Flaska potvrđuje i činjenica da ga koriste aplikacije poput Pinterest-a i LinekedIn-a[26].

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Slika 7 Primjer početne "Hello World" Flask web aplikacije

```
$ flask --app hello run
 * Serving Flask app 'hello'
 * Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

Slika 8 Konzolni izlaz kod pokretanja Flask web aplikacije

[24] Full Stack Python, Flask, Dostupno na: https://www.fullstackpython.com/flask.html [Pristupljeno 03.09.2023.]
[25] Python Tutorial, What is Flask Python, Dostupno na: https://pythonbasics.org/what-is-flask-python/ [Pristupljeno 03.09.2023.]
[26] Wikipedia, Flask (web framework), Dostupno na: https://pythonbasics.org/what-is-flask-python/ [Pristupljeno 03.09.2023.]

4.1.3 Pelican

Pelican je jedan od generatora statičnih stranica napisan u Python programskom jeziku, koji je svoje ime dobio po anagramu riječi *calepin*, koja u francuskom jeziku znači bilježnica[27]. On predstavlja generator statičnih stranica koji kombinira sadržaj napisan u Markdown-u ili reStructuredText-u kako bi stvorio statične web stranice, a s obzirom da je generator statičnih stranica ne treba mu baza podataka ili dinamički server. Uočljiva je njegova česta upotreba kod izrade blogova, osobnih web stranica i sličnih web projekata[28].

Pelican trenutačno podržava mnogo značajki kao što su:

- Blog članci i stranice
- Komentari putem vanjskih usluga
- Teme stvorene pomoću Jinja2 predložaka
- Generiranje PDF dokumenata
- Uvoz iz WordPress-a, Dotclear-a i slično
- Integracija s vanjskim alatima kao što su Google Analytics[29].

```
python -m pip install "pelican[markdown]"
```

Slika 9 Instalacija Pelican-a u Python virtualnom okruženju

```
pelican-quickstart
```

Slika 10 Inicijalno pokretanje Pelican-a

```
pelican content
```

Slika 11 Naredba za generiranje sadržaja

```
pelican --listen
```

Slika 12 Pokretanje Pelican web servera

[27] Pelican, Pelican 4.8.0, Dostupno na: https://docs.getpelican.com/en/latest/ [Pristupljeno 03.09.2023.]
[28] Full Stack Python, Pelican, Dostupno na: https://www.fullstackpython.com/pelican.html [Pristupljeno 03.09.2023.]
[29] Jamstack, Pelican, Dostupno na: https://jamstack.org/generators/pelican/ [Pristupljeno 03.09.2023.]

4.1.4 Jinja2

Jinja predstavlja web sustav za predloške (eng. *Template engine*) za Python programski jezik. Kreirana je od strane Armina Ronachera te je licencirana pod BSD licencom. Jinja je bazirana na tekstu pa se može koristiti za generiranje bilo kojeg markup-a, kao i izvornog koda. Također, omogućava prilagodbu oznaka (eng. *tags*), filtera, testova, globalnih varijabli kao i pozive funkcija s argumentima na objektima. Jinja je zadani sustav predložaka za Flask, kao i za izradu tema za Pelican generator statičnih stranica.

```
{% extends "layout.html" %}
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

Slika 13 Primjer Jinja koda

4.1.5 MySQL

MySQL predstavlja sustav za upravljanje relacijskim bazama podataka[30] u vlasništvu tvrtke Oracle. Trenutačno je na drugom mjestu po popularnosti sustava za upravljanje bazama podataka i na prvom mjestu kao besplatan softver otvorenog koda (eng. *Open-source*)[31]. MySQL se koristi u mnogim web stranicama i aplikacijama kao što su Facebook, Flickr, MediaWiki, Twitter, Youtube i slično[32].

Rank			DBMS	Database Model	Score		
Sep 2023	Aug 2023	Sep 2022			Sep 2023	Aug 2023	Sep 2022
1.	1.	1.	Oracle	Relational, Multi-model	1240.88	-1.22	+2.62
2.	2.	2.	MySQL	Relational, Multi-model	1111.49	-18.97	-100.98
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	902.22	-18.60	-24.08
4.	4.	4.	PostgreSQL	Relational, Multi-model	620.75	+0.37	+0.29
5.	5.	5.	IBM Db2	Relational, Multi-model	136.72	-2.52	-14.67
6.	↑ 7.	↑ 7.	SQLite	Relational	129.20	-0.72	-9.62
7.	↓ 6.	↓ 6.	Microsoft Access	Relational	128.56	-1.78	-11.47
8.	8.	↑ 9.	Snowflake	Relational	120.89	+0.27	+17.39
9.	9.	↓ 8.	MariaDB	Relational, Multi-model	100.45	+1.80	-9.70
10.	10.	10.	Microsoft Azure SQL Database	Relational, Multi-model	82.73	+3.22	-1.69

Slika 14 Rangiranje MySql baze podataka

[30] MySQL, What is MySQL?, Dostupno na: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html [Pristupljeno 04.09.2023.]
[31] DB-Engines, 2023., DB-Engines Ranking of Relational DBMS, Dostupno na: https://db-engines.com/en/ranking/relational+dbms [Pristupljeno 04.09.2023.]
[32] Wikipedia, MySQL, Dostupno na: https://en.wikipedia.org/wiki/MySQL [Pristupljeno 04.09.2023.]

4.2 Razvoj web portala i pripadajućeg CMS sustava

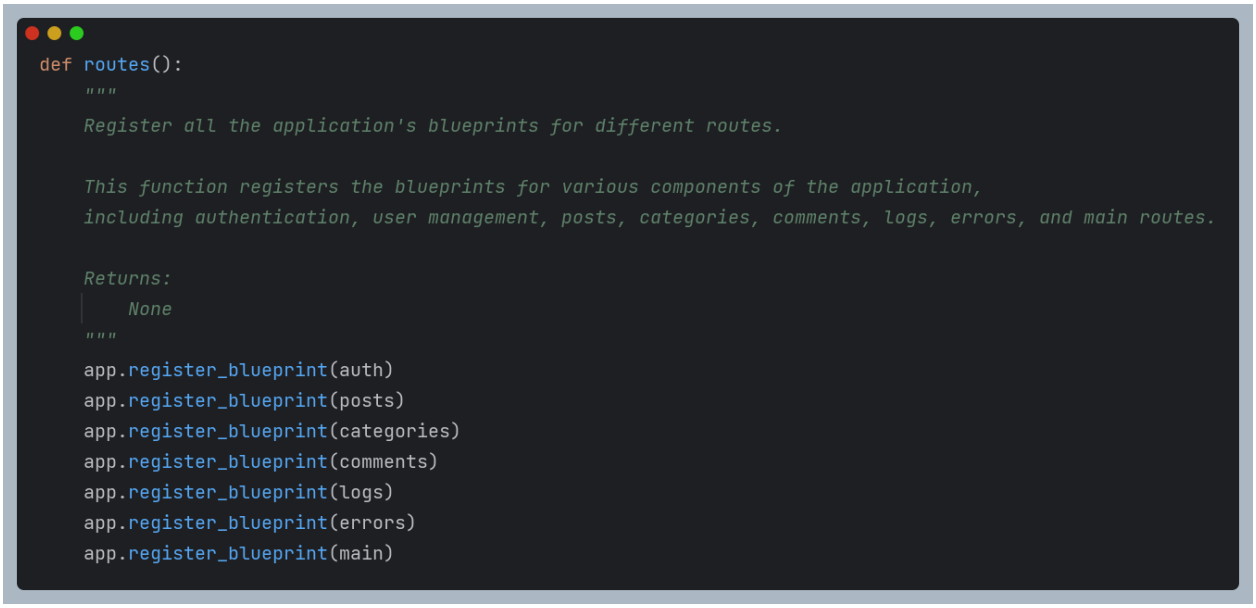
Za razvoj web portala i pripadajućeg CMS sustava odabran je Flask *framework*. Najprije, da bi razvoj uopće započeo bilo je potrebno odrediti sve zahtjeve, odnosno sve značajke koje će navedeni CMS sustav imati. Naime, jezgra svakog CMS sustava klasična je CRUD (*Create, Read, Update, Delete*) aplikacija pa je i tijek izrade krenuo u ovom smjeru. Također, da bi CMS sustav i pripadajući portal imali sve potrebno za modernu web aplikaciju, bilo je nužno omogućiti korisnicima rad s korisničkim računima, a kako bi administratori imali uvid u sve što se događa na web aplikaciji, bilo je potrebno imati odgovarajući način za obradu grešaka (eng. *Error handling*), kao i za bilježenje zahtijeva (eng. *Logging Requests*).

4.2.1 Flask nacrti

Kako bi aplikacija bila modularna te kako bi razvoj i održavanje bili što pristupačniji, u izradi web aplikacije korištena je biblioteka Flask-Blueprints. Ova biblioteka razdvaja strukturu aplikacije po izboru developera te omogućuje bolju preglednost strukture projekta.

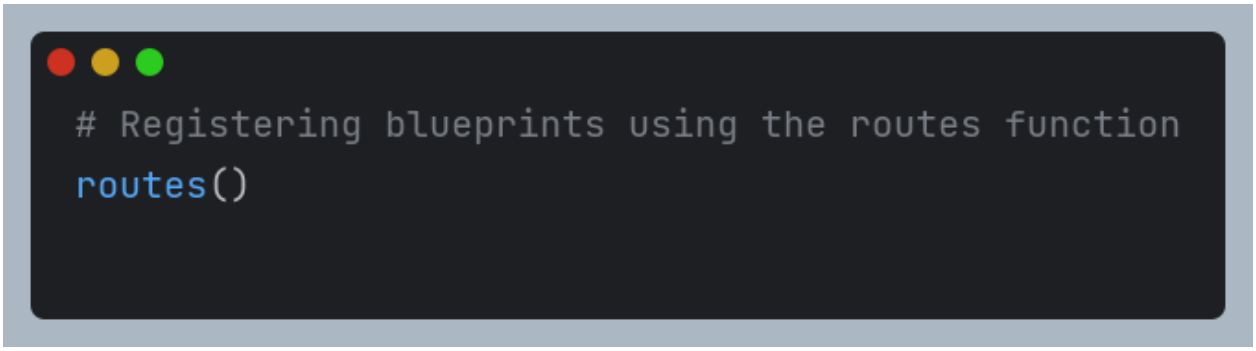
Slika 15 Primjer koda za kreiranje nacrta (eng. *Blueprint*)

Svaki nacrt (eng. *Blueprint*) potrebno je odvojiti u pripadajuću mapu te ga registrirati u inicijalnoj *__init__.py* datoteci. Kako bi projekt bio još pregledniji, registracija nacrta smještena je u funkciju *routes* koja se nalazi u datoteci *blueprint_routes.py* te se poziva u navedenoj inicijalnoj datoteci.



```python
def routes():
    """
    Register all the application's blueprints for different routes.

    This function registers the blueprints for various components of the application,
    including authentication, user management, posts, categories, comments, logs, errors, and main routes.

    Returns:
        None
    """
    app.register_blueprint(auth)
    app.register_blueprint(posts)
    app.register_blueprint(categories)
    app.register_blueprint(comments)
    app.register_blueprint(logs)
    app.register_blueprint(errors)
    app.register_blueprint(main)
```

Slika 16 Funkcija registracije nacrta u aplikaciji



```python
# Registering blueprints using the routes function
routes()
```

Slika 17 Pozivanje funkcije *routes()* u inicijalnoj datoteci koja registrira nacrte

Svi nacrti u aplikaciji imaju pripadajuće datoteke s potrebnim metodama:

- *__init__.py* – datoteka koja u Python-u označuje paket
- *routes.py* – datoteka u kojoj su definirane API točke (eng. *API Points*) i pripadajuće funkcije
- *forms.py* – datoteka u kojoj su definirane forme za unos podataka
- *database_manager.py* – datoteka u kojoj su definirane funkcije korištene za rad s bazom podataka
- *utils.py* – datoteka s pomoćnim funkcijama

Aplikacija je podijeljena u šest nacrta:

- *Auth* – nacrt za autorizaciju i autentifikaciju
- *Posts* – nacrt za rad s *Post* klasom modela, odnosno za rad s člancima/postovima
- *Categories* – nacrt za rad s *Categoy* klasom modela, odnosno za rad s kategorijama
- *Comments* – nacrt za rad s *Comment* klasom modela, odnosno za rad s komentarima
- *Logs* – nacrt za rad s bilježenjem zahtjeva i s *RequestLog* klasom modela
- *Errors* – nacrt za rad s greškama i *ErrorLog* klasom modela. U njemu su definirane funkcije rukovanja greškama
- *Main* – nacrt za rad s početnom stranicom, stranicama za prikaz članaka/postova, kontakt formom, i ostalim info stranicama
- *Users* – nacrt za rad s *User* klasom modela, odnosno za rad s korisnicima

4.2.2 Klase modela

Flask ne dolazi s pripadajućim ORM-om, što ostavlja slobodu u odabiru načina rada s bazom podataka. Flask-SQLAlchemy najpopularniji je ORM u Flask okruženju te je u ovom slučaju bio najbolja opcija. Klase modela, kao i njihovi atributi sastavljeni su kako bi zadovoljili sve zahtjeve aplikacije, stoga su sačinjeni od klase *User* (korisnici), *Role* (prava), *UserRoles* (prava po korisniku), *Post* (članci/objave), *Category* (kategorije članaka), *RequestLog* (zapis poziva), *ErrorLog* (zapis grešaka).

Klasa *User* predstavlja tablicu *users* u bazi podataka te se sastoji od atributa kao što su *id*, *username* (korisničko ime), *email*, *email_confirmed_at* (vrijeme potvrde email adrese), *password* (lozinka), *name* (ime i/ili prezime) i *active* (aktivnost korisnika). Također, klasa modela *User* ima poveznicu na klase, odnosno na tablice *user_roles*, *comments*, *request_logs* i *error_logs*.

```python
class User(db.Model, UserMixin):
    __tablename__ = "users"

    id = db.Column( *args: db.Integer, primary_key=True)
    username = db.Column( *args: db.String(25), unique=True, nullable=False)
    email = db.Column( *args: db.String(100), unique=True, nullable=False)
    email_confirmed_at = db.Column( *args: db.DateTime(), default=datetime.utcnow)
    password = db.Column( *args: db.String(100), nullable=False)
    roles = db.relationship("Role", secondary="user_roles")
    name = db.Column( *args: db.String(50), nullable=True)
    comments = db.relationship(
        "Comment", backref="user", lazy="dynamic", cascade="all, delete-orphan"
    )
    active = db.Column( *args: db.Boolean(), default=True)
    request_logs = db.relationship("RequestLog", backref="user", lazy="dynamic")
    error_logs = db.relationship("ErrorLog", backref="user", lazy="dynamic")


    Ivan Kojic
    def __repr__(self):
        return f"User('{self.username}', '{self.email}')"
```

Slika 18 Klasa modela *User*

Klasa *Role* predstavlja tablicu *roles* u bazi podataka te se sastoji od atributa kao što *id* i *name* (naziv uloge).

```
class Role(db.Model):
    __tablename__ = "roles"

    id = db.Column( *args: db.Integer(), primary_key=True)
    name = db.Column( *args: db.String(50), unique=True)
```

Slika 19 Klasa modela *Role*

Klasa *UserRoles* predstavlja pomoćnu *user_roles* tablicu u bazi podataka. Svrha ove tablice je poveznica između *User* i *Role* tablica, stoga ima strane ključeve (eng. *Foreign Key*) na *id* atribute *users* i *roles* tablica.

```
class UserRoles(db.Model):
    __tablename__ = "user_roles"

    id = db.Column( *args: db.Integer(), primary_key=True)
    user_id = db.Column( *args: db.Integer(), db.ForeignKey( column: "users.id", ondelete="CASCADE"))
    role_id = db.Column( *args: db.Integer(), db.ForeignKey( column: "roles.id", ondelete="CASCADE"))
```

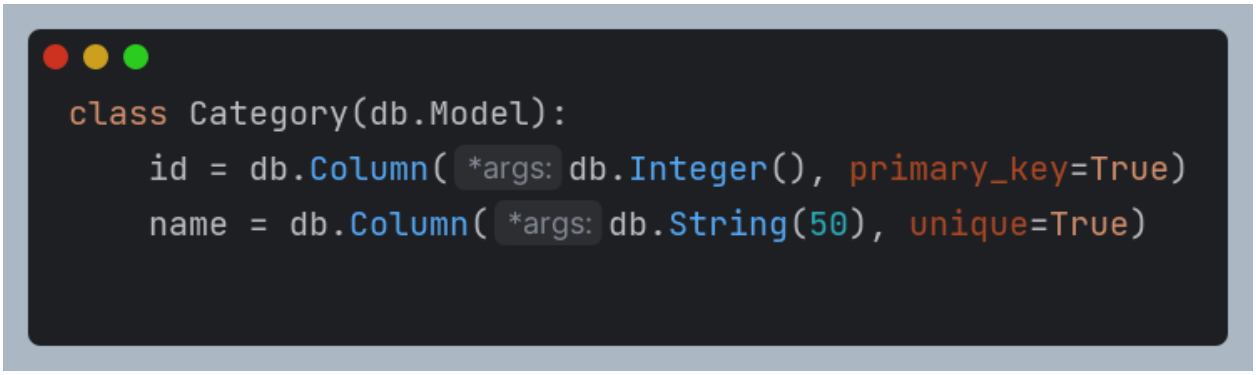
Slika 20 Klasa pomoćne tablice *UserRoles*

Klasa *Post* predstavlja istoimenu tablicu u bazi podataka, a sadrži podatke o objavama, odnosno člancima, stoga ima *id*, *title* (naslov objave/članka), *subtitle* (podnaslov članka/objave), *description* (opis članka/objave), *slug* (URL - slug), *headImg* (putanja do naslovne slike), *category_id* (ID kategorije), *language* (jezik), *author* (autor članka/objave), *date_posted* (datum kad je članak/objava spremljena u bazu), *content* (sadržaj članka/objave), i *isPublished* (objavljen li je članak/objava) atribute. Također, klasa modela ima poveznicu na klase, odnosno tablice *category* i *comments*.

```python
class Post(db.Model):
    id = db.Column( *args: db.Integer, primary_key=True)
    title = db.Column( *args: db.String(100), nullable=False)
    subtitle = db.Column( *args: db.String(100), nullable=False)
    description = db.Column( *args: db.Text, nullable=False)
    slug = db.Column( *args: db.String(100), nullable=False)
    headImg = db.Column( *args: db.String(254), nullable=False, default="default.jpg")
    category_id = db.Column( *args: db.Integer, db.ForeignKey("category.id"), nullable=False)
    category = db.relationship("Category", backref="posts")
    language = db.Column( *args: db.String(30), nullable=False)
    author = db.Column( *args: db.String(50), nullable=False)
    date_posted = db.Column( *args: db.DateTime, nullable=False, default=datetime.utcnow)
    content = db.Column( *args: db.Text, nullable=False)
    isPublished = db.Column( *args: db.Boolean(), nullable=False, default=False)
    comments = db.relationship(
        "Comment", backref="post", lazy="dynamic", cascade="all, delete-orphan"
    )


    Ivan Kojić +1
    def __repr__(self):
        formatted_date_posted = self.date_posted.strftime("%d-%m-%Y %H:%M:%S")
        return f"Post('{self.title}', '{self.language}', '{self.slug}', '{formatted_date_posted}')"
```

Slika 21 Klasa modela *Post*

Klasa *Category* predstavlja istoimenu tablicu u bazi podataka te sadrži atribute za *id* i *name* (naziv) kategorije.



Slika 22 Klasa modela *Category*

Klasa *Comment* predstavlja istoimenu tablicu u bazi podatka koja služi za pohranjivanje korisničkih komentara. Sadrži *id*, *content* (sadržaj komentara) i *date_posted* (datum objavljenog komentara) atribute te veze na *user* i post klase modela, odnosno tablice putem stranih ključeva *user_id* i *post_id*.

Slika 23 Klasa modela Comment

Klasa *RequestLog* predstavlja istoimenu tablicu u bazi podataka te služi za bilježenje zahtjeva. Sastoji se od *id*, *timestamp* (vrijeme kad se zahtjev dogodio), *user_id* (ID korisnika koji je poslao zahtjev), *endpoint* (krajnja točka s zatraženog zahtjeva) te *methodType* (tip zatražene metode – *GET / POST / PUT / DELETE*) atributa.

```
class RequestLog(db.Model):
    id = db.Column( *args: db.Integer, primary_key=True)
    endpoint = db.Column(db.String(255))
    methodType = db.Column(db.String(10))
    user_id = db.Column( *args: db.Integer, db.ForeignKey("users.id"))
    timestamp = db.Column( *args: db.DateTime, default=datetime.utcnow)
```

Slika 24 Klasa modela *RequestLog*

Klasa *ErrorLog* predstavlja istoimenu tablicu u bazi podatka te se koristi za zapisivanje greški ukoliko se dogode. Sastoji se od *id*, *timestamp* (vrijeme kad se greška dogodila), *user_id* (ID korisnika koji je poslao zahtjev kada se greška dogodila), *endpoint* (krajnja točka s zatraženog zahtjeva kada se dogodila greška) te *methodType* (tip zatražene metode – *GET / POST / PUT / DELETE*), *status_code* (kod statusa greške) te *error_message* (poruka greške) atributa.

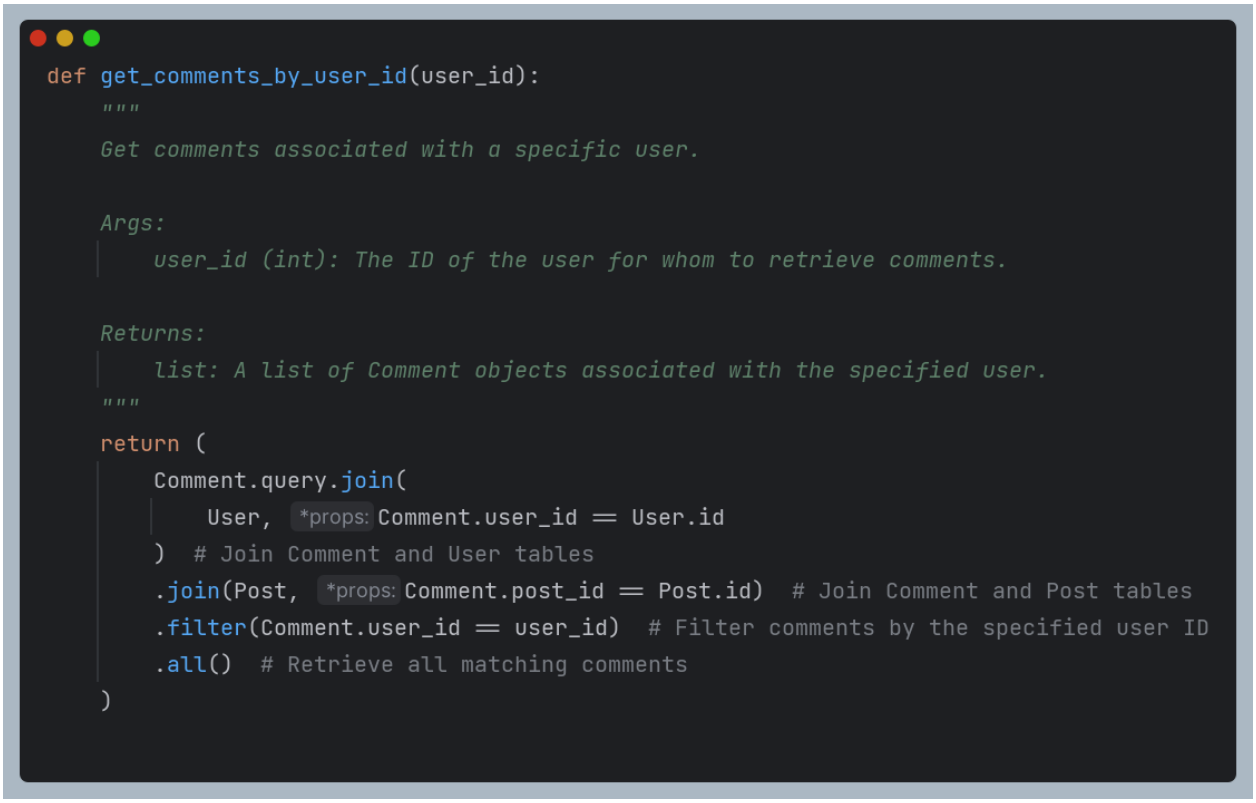
```
class ErrorLog(db.Model):
    id = db.Column( *args: db.Integer, primary_key=True)
    timestamp = db.Column( *args: db.DateTime, nullable=False, default=datetime.utcnow)
    user_id = db.Column( *args: db.Integer, db.ForeignKey("users.id"))
    endpoint = db.Column(db.String(255))
    methodType = db.Column(db.String(10))
    status_code = db.Column(db.Integer)
    error_message = db.Column(db.Text)
```

Slika 25 Klasa modela *ErrorLog*

Sve navedene klase nalaze se u datoteci *models.py*.

Logički dio rada s bazom podataka, odnosno čitanje, zapisivanje, ažuriranje i brisanje podataka, bilo je potrebno odvojiti od funkcija s rutama, stoga su kreirane *database_manager.py* datoteke unutar pripadajućih nacrta (eng. *Blueprints*).

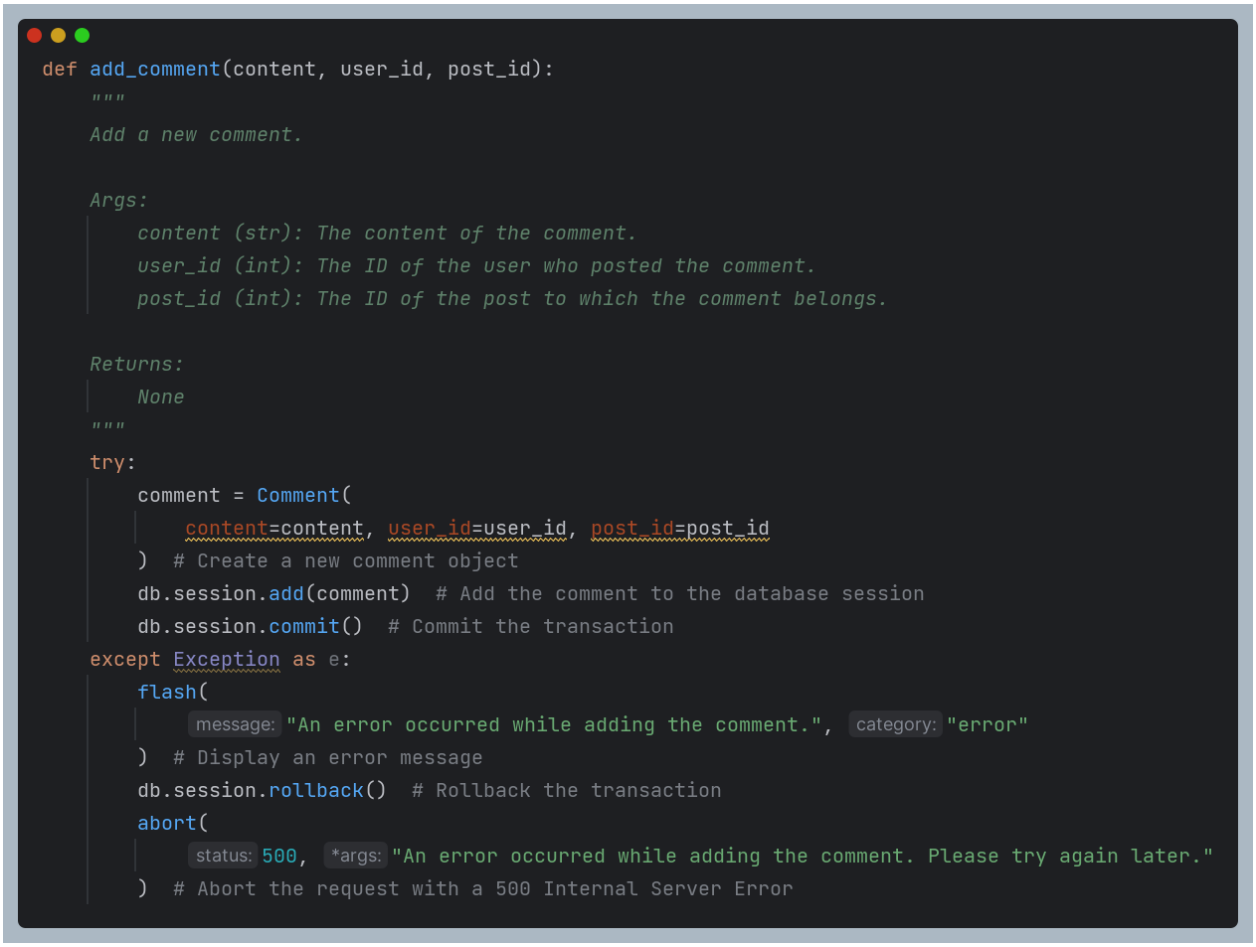
Funkcije za zapisivanje, ažuriranje ili brisanje podataka omotane su u *try-except* blokove kako bi se osigurala odgovarajuća obrada i zapisivanje grešaka te kako bi korisnik dobio odgovarajuću poruku.



```
def get_comments_by_user_id(user_id):
    """
    Get comments associated with a specific user.

    Args:
        user_id (int): The ID of the user for whom to retrieve comments.

    Returns:
        list: A list of Comment objects associated with the specified user.
    """
    return (
        Comment.query.join(
            User, *props: Comment.user_id == User.id
        )  # Join Comment and User tables
        .join(Post, *props: Comment.post_id == Post.id)  # Join Comment and Post tables
        .filter(Comment.user_id == user_id)  # Filter comments by the specified user ID
        .all()  # Retrieve all matching comments
    )
```

Slika 26 Primjer dohvata iz baze podataka

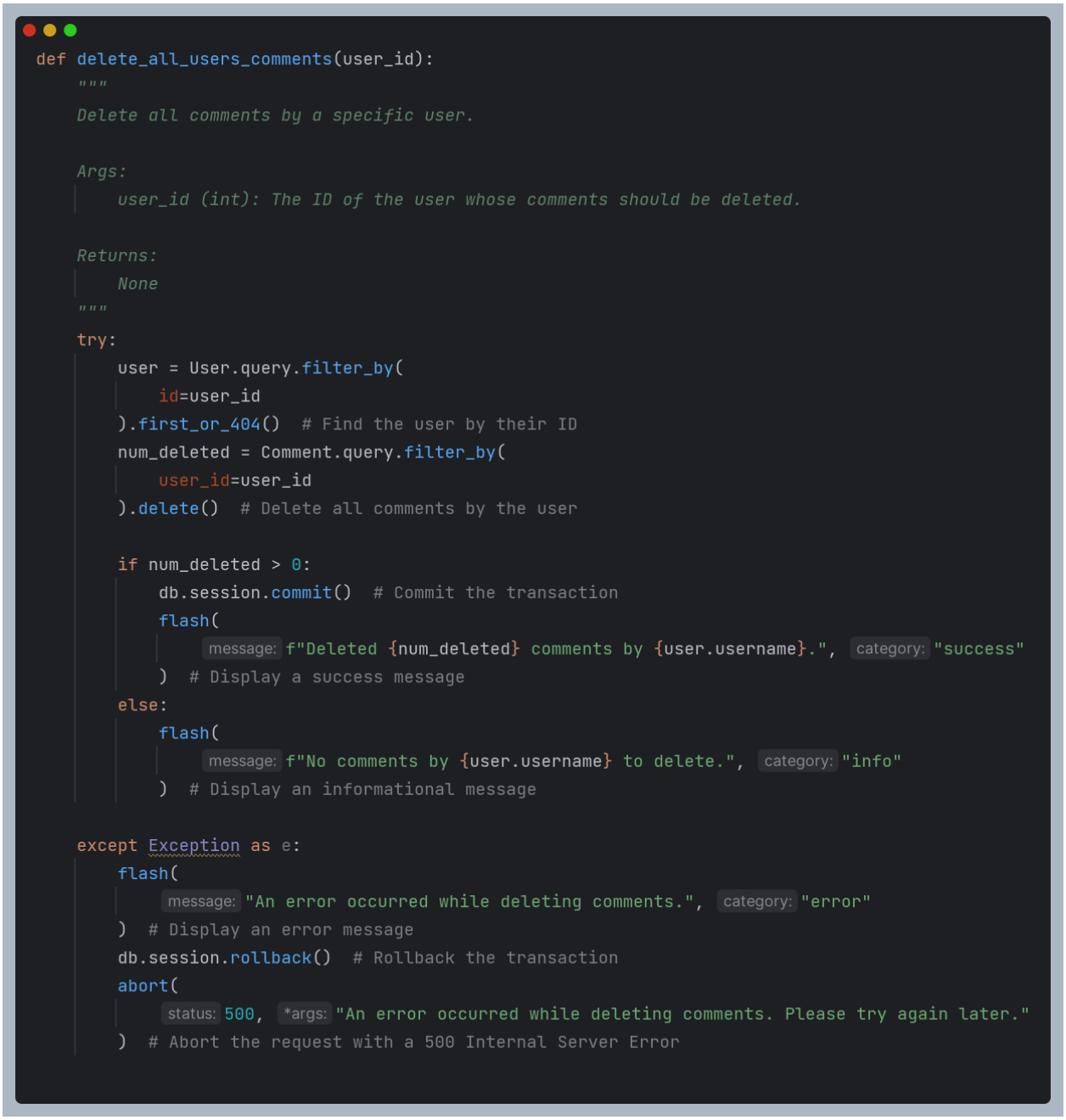
Na slici 26 prikazan je primjer dohvata iz baze, odnosno primjer dohvata svih komentara ostavljenih od strane nekog korisnika. Funkcija prima ID korisnika kao parametar te radi upit na bazu na retke čija vrijednost *user_id* odgovara proslijeđenom parametru. Navedeni upit spaja podatke *Comment* i *Post* tablica te ih sve dohvaća.



```python
def add_comment(content, user_id, post_id):
    """
    Add a new comment.

    Args:
        content (str): The content of the comment.
        user_id (int): The ID of the user who posted the comment.
        post_id (int): The ID of the post to which the comment belongs.

    Returns:
        None
    """
    try:
        comment = Comment(
            content=content, user_id=user_id, post_id=post_id
        )  # Create a new comment object
        db.session.add(comment)  # Add the comment to the database session
        db.session.commit()  # Commit the transaction
    except Exception as e:
        flash(
            message: "An error occurred while adding the comment.", category: "error"
        )  # Display an error message
        db.session.rollback()  # Rollback the transaction
        abort(
            status: 500, *args: "An error occurred while adding the comment. Please try again later."
        )  # Abort the request with a 500 Internal Server Error
```

Slika 27 Primjer zapisa u bazu podataka

Slika 27 prikazuje primjer zapisa u bazu podataka, odnosno dodavanja novog komentara u tablicu *Comment*. Primljeni parametri funkcije vezuju se kao atributi na novonastali objekt klase *Comment*, odnosno entitet koji se zapisuje u bazu podataka.



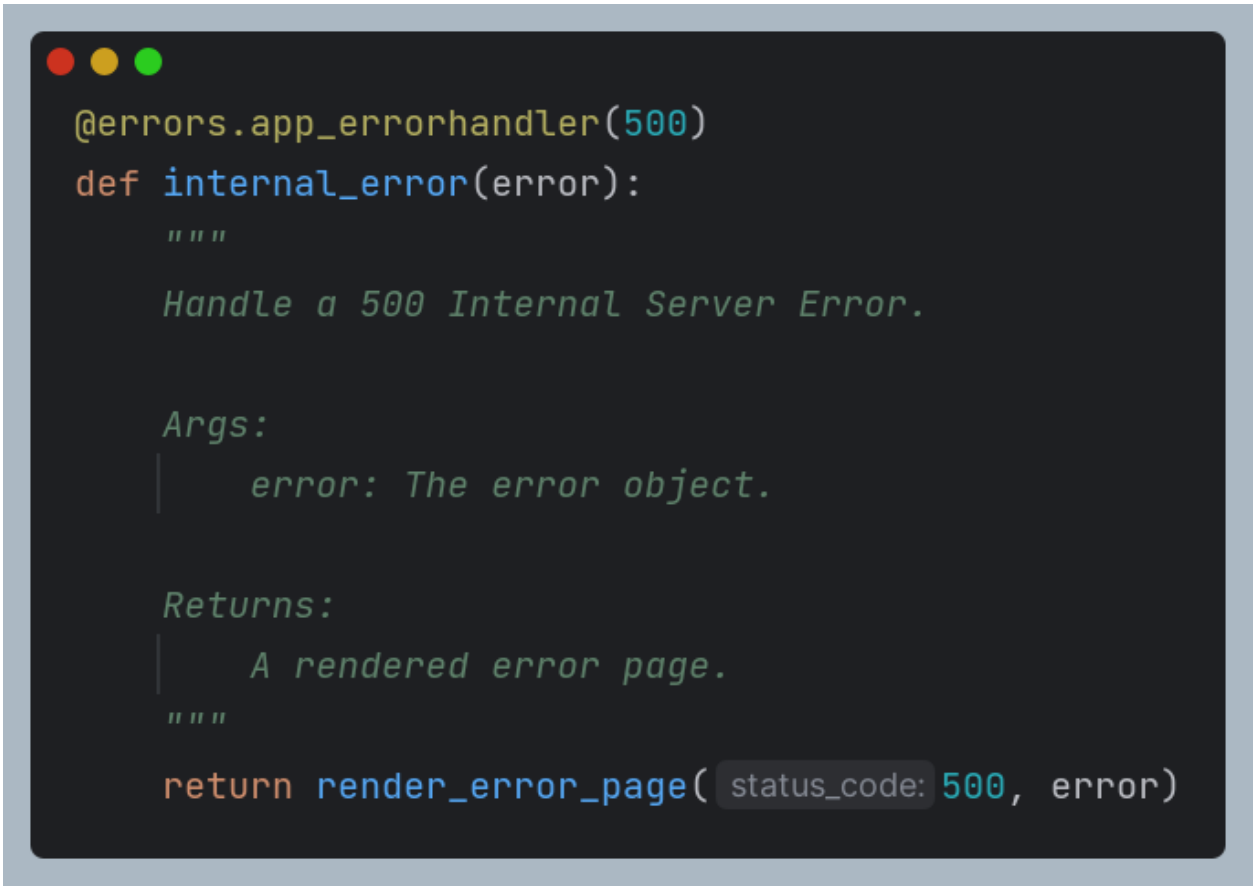
```python
def delete_all_users_comments(user_id):
    """
    Delete all comments by a specific user.

    Args:
        user_id (int): The ID of the user whose comments should be deleted.

    Returns:
        None
    """
    try:
        user = User.query.filter_by(
            id=user_id
        ).first_or_404()  # Find the user by their ID
        num_deleted = Comment.query.filter_by(
            user_id=user_id
        ).delete()  # Delete all comments by the user

        if num_deleted > 0:
            db.session.commit()  # Commit the transaction
            flash(
                message: f"Deleted {num_deleted} comments by {user.username}.", category: "success"
            )  # Display a success message
        else:
            flash(
                message: f"No comments by {user.username} to delete.", category: "info"
            )  # Display an informational message

    except Exception as e:
        flash(
            message: "An error occurred while deleting comments.", category: "error"
        )  # Display an error message
        db.session.rollback()  # Rollback the transaction
        abort(
            status: 500, *args: "An error occurred while deleting comments. Please try again later."
        )  # Abort the request with a 500 Internal Server Error
```

Slika 28 Primjer brisanja iz baze podataka

Na slici 28 prikazan je primjer brisanja podataka iz baze, točnije brisanja svih komentara istog autora. ID korisnika zaprimljen kao parametar funkcije se najprije koristi u upitu u kojem se provjerava postojanje navedenog korisnika, a zatim funkcija briše sve retke iz baze čiji stupac *user_id* odgovara zadanom ID-ju korisnika te vraća broj obrisanih redaka.

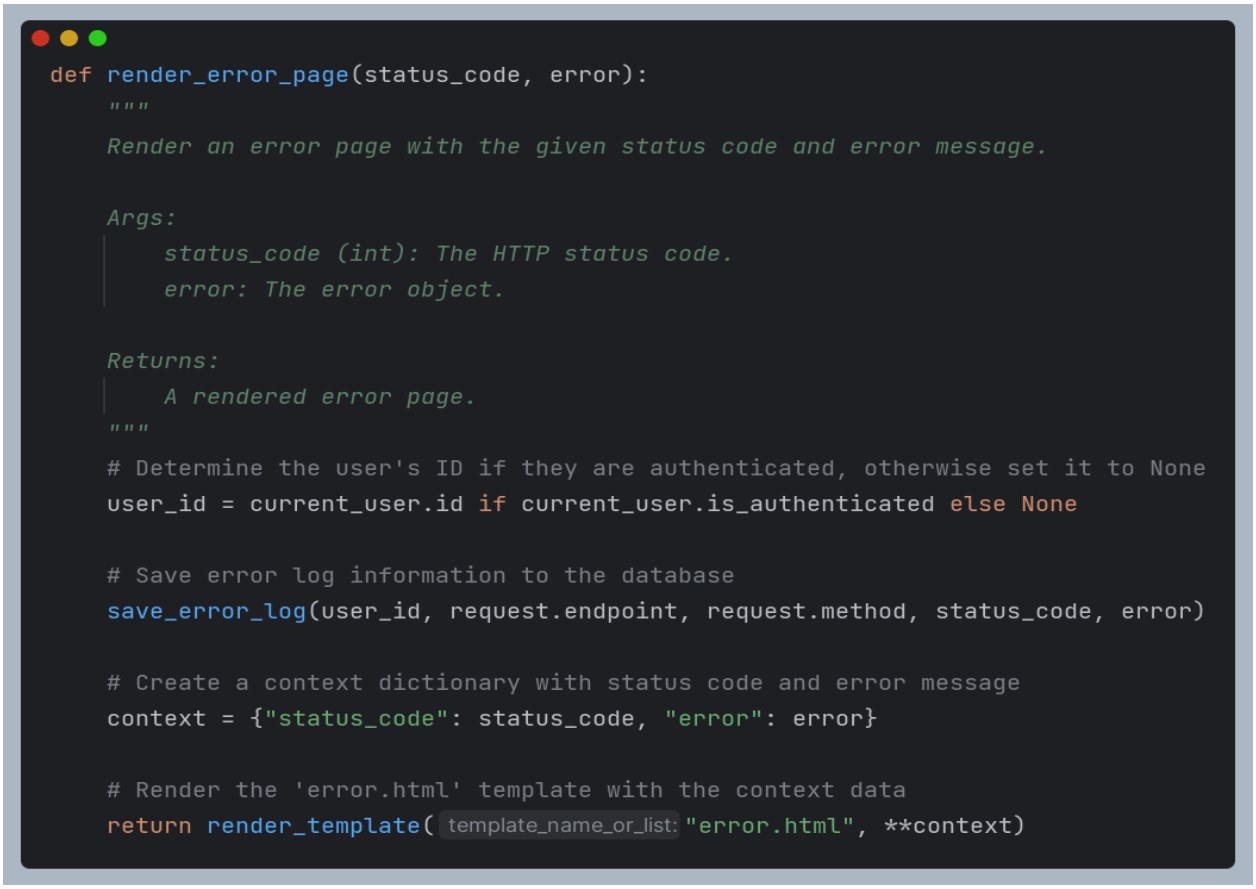
4.2.3 Obrada grešaka

Za vrijeme rada aplikacije otvara se mogućnost pojave greške, stoga je potrebno osigurati valjanu obradu i zapis grešaka. *Errors* nacrt osigurava odgovarajuću poruku korisnicima te javljanje administratorima sustava da je došlo od određene greške. Također, u njemu su definirane greške koje aplikacija može vratiti korisniku te je omogućen prikaz odgovarajuće poruke te spremanje tih grešaka u bazu podataka. Nadalje, osigurana je obrada greški kao što su greške s kodom 400, 401, 403, 404 i 500.



Slika 29 Primjer funkcije za obradu greške s kodom 500

Na slici 29 prikazan je primjer jedne takve funkcije za obradu greške, preciznije greške s kodom 500 *Internal Server Error*. Navedena funkcija je u aplikaciji registrirana kao *Error Handler* upravo za tu grešku, dodjeljivanjem tog parametra u dekoratoru funkcije. Pojavom navedene greške poziva se funkcija kojoj je svrha obrada te greške.



```python
def render_error_page(status_code, error):
    """
    Render an error page with the given status code and error message.

    Args:
        status_code (int): The HTTP status code.
        error: The error object.

    Returns:
        A rendered error page.
    """
    # Determine the user's ID if they are authenticated, otherwise set it to None
    user_id = current_user.id if current_user.is_authenticated else None

    # Save error log information to the database
    save_error_log(user_id, request.endpoint, request.method, status_code, error)

    # Create a context dictionary with status code and error message
    context = {"status_code": status_code, "error": error}

    # Render the 'error.html' template with the context data
    return render_template( template_name_or_list: "error.html", **context)
```

Slika 30 Funkcija koja prikazuje odgovarajući ekran korisniku te poziva funkciju za spremanje podataka o grešci u bazu podataka

Na slici 30 prikazana je univerzalna funkcija za prikaz odgovarajuće poruke, odnosno ekrana, korisniku kao i za spremanje podataka o grešci u bazu podataka. Kao parametre prima statusni kod greške kao i samu grešku. Navedeni podaci spremaju se u bazu podataka skupa s ID-jem korisnika ukoliko je prijavljen, a naposljetku korisniku se prikazuje ekran s porukom o grešci prikazan na slici 31.

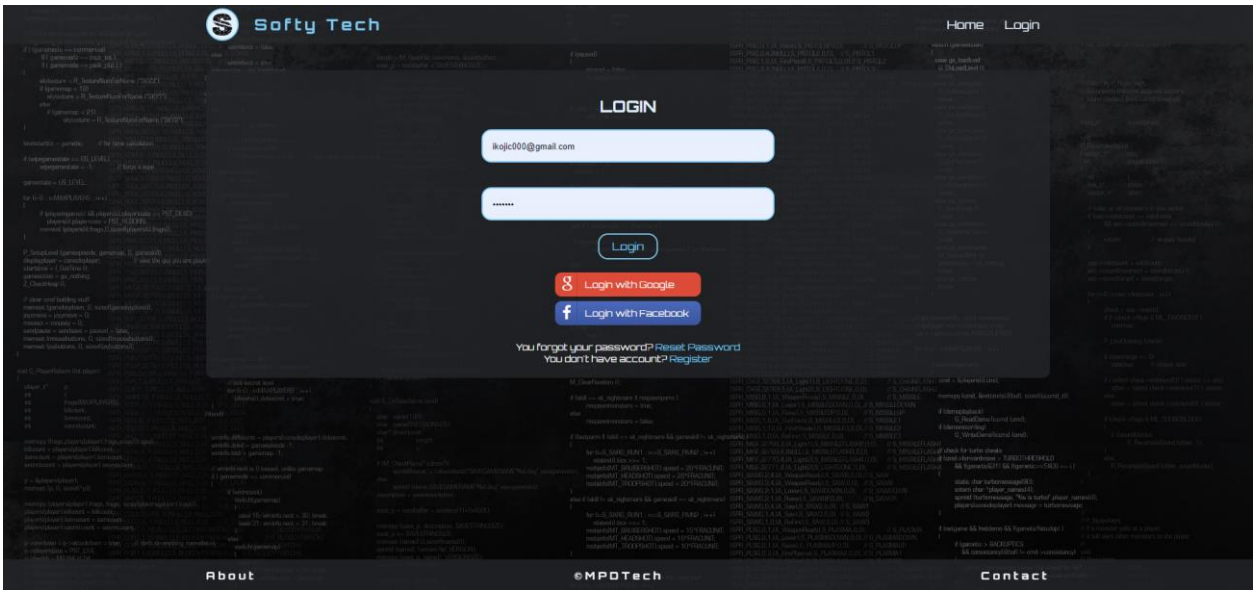
Slika 31 Primjer prikaza poruke na ekranu ukoliko se dogodi greška

4.2.4 Flask forme

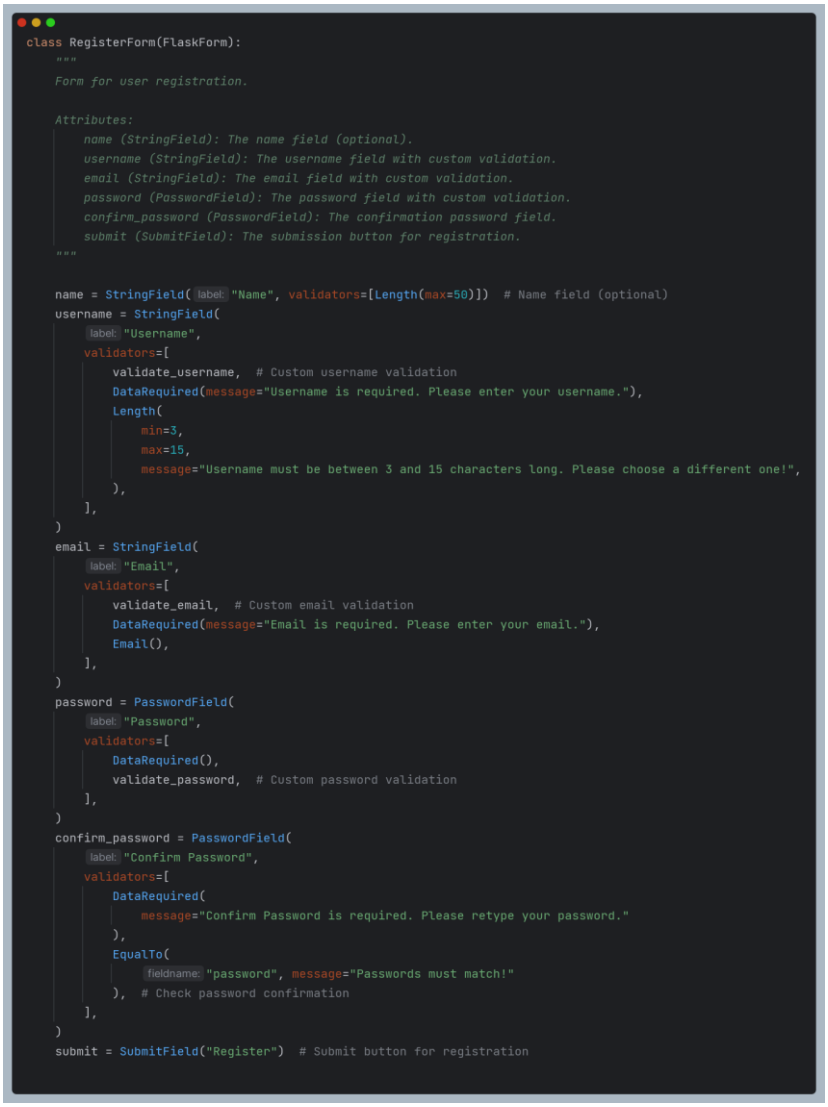
Za rad s formama odabrana je biblioteka Flask-WTF koja omogućuje rad s WTForms bibliotekom. Navedena biblioteka služi za kreiranje i prikaz formi i validaciju istih. Forme je najprije potrebno definirati kao klase s valjanim atributima. Svaki atribut odgovara jednom polju definirane forme. Forme korištene u projektu definirane su u datotekama *forms.py* u pripadajućim mapama nacrta.

```python
class LoginForm(FlaskForm):
    """
    Form for user login.

    Attributes:
        email (StringField): The email field for username or email input.
        password (PasswordField): The password field.
        submit (SubmitField): The submission button for login.
    """

    email = StringField(
        label: "Email",
        validators=[
            DataRequired(
                message="Username/Email is required. Please enter your username or email."
            ),
        ],
    )
    password = PasswordField( label: "Password", validators=[DataRequired()])
    submit = SubmitField("Login")  # Submit button
```

Slika 32 Definiranje forme za prijavu - *LoginForm* klase

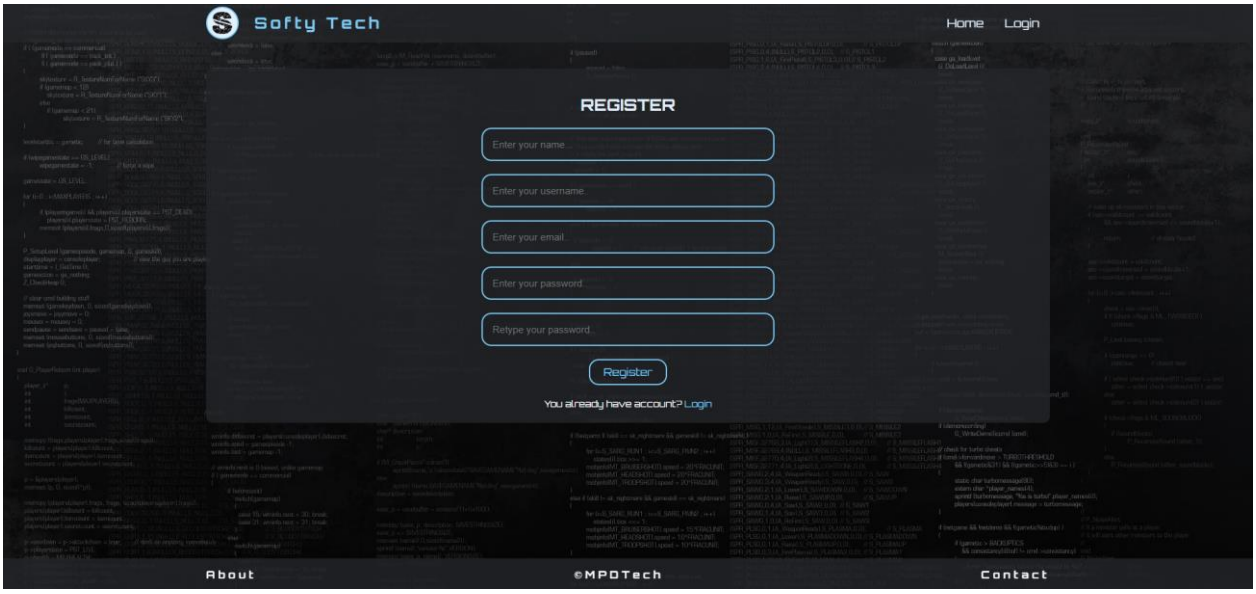


Slika 33 Prikaz forme za prijavu - *LoginForm* klasa

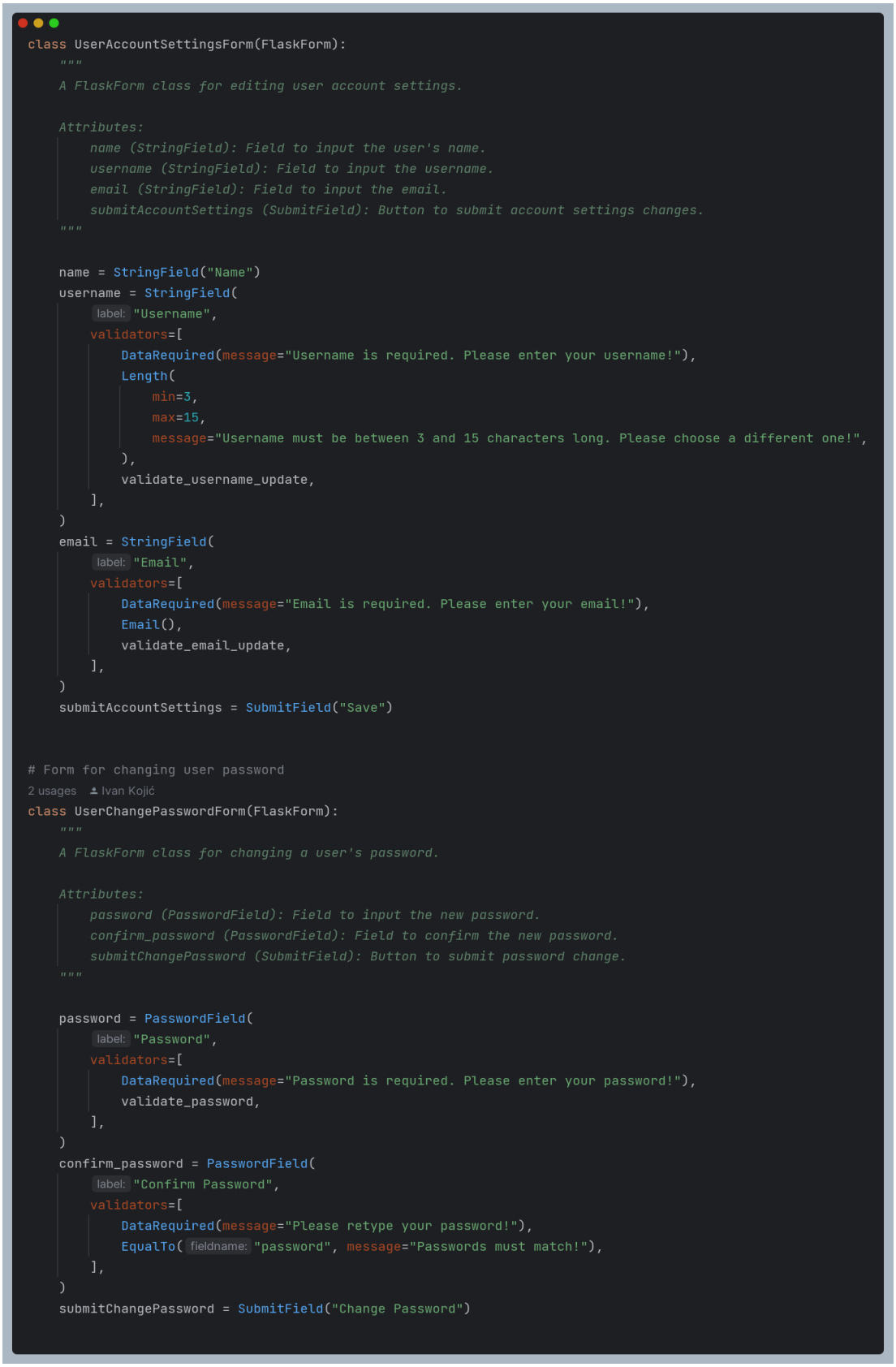


```python
class RegisterForm(FlaskForm):
    """
    Form for user registration.

    Attributes:
        name (StringField): The name field (optional).
        username (StringField): The username field with custom validation.
        email (StringField): The email field with custom validation.
        password (PasswordField): The password field with custom validation.
        confirm_password (PasswordField): The confirmation password field.
        submit (SubmitField): The submission button for registration.
    """

    name = StringField( label: "Name", validators=[Length(max=50)])  # Name field (optional)
    username = StringField(
        label: "Username",
        validators=[
            validate_username,  # Custom username validation
            DataRequired(message="Username is required. Please enter your username."),
            Length(
                min=3,
                max=15,
                message="Username must be between 3 and 15 characters long. Please choose a different one!",
            ),
        ],
    )
    email = StringField(
        label: "Email",
        validators=[
            validate_email,  # Custom email validation
            DataRequired(message="Email is required. Please enter your email."),
            Email(),
        ],
    )
    password = PasswordField(
        label: "Password",
        validators=[
            DataRequired(),
            validate_password,  # Custom password validation
        ],
    )
    confirm_password = PasswordField(
        label: "Confirm Password",
        validators=[
            DataRequired(
                message="Confirm Password is required. Please retype your password."
            ),
            EqualTo(
                fieldname: "password", message="Passwords must match!"
            ),  # Check password confirmation
        ],
    )
    submit = SubmitField("Register")  # Submit button for registration
```

Slika 34 Definiranje forme za registraciju - *RegisterForm* klasa

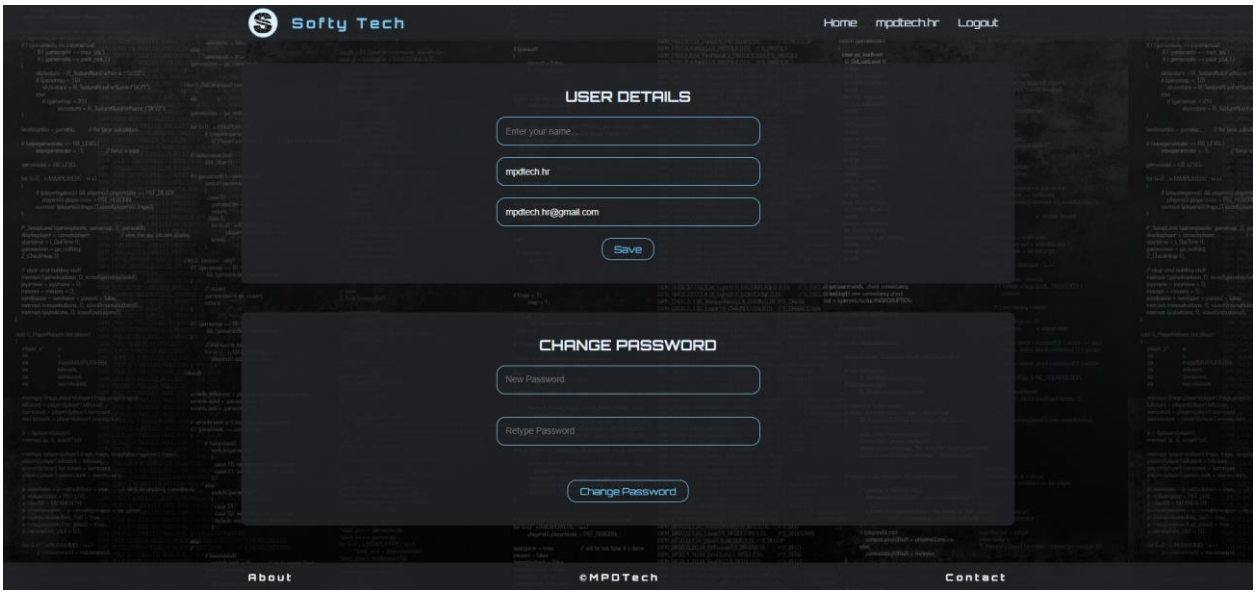


Slika 35 Prikaz forme za registraciju - *RegisterForm* klasa



```python
class UserAccountSettingsForm(FlaskForm):
    """
    A FlaskForm class for editing user account settings.

    Attributes:
        name (StringField): Field to input the user's name.
        username (StringField): Field to input the username.
        email (StringField): Field to input the email.
        submitAccountSettings (SubmitField): Button to submit account settings changes.
    """

    name = StringField("Name")
    username = StringField(
        label: "Username",
        validators=[
            DataRequired(message="Username is required. Please enter your username!"),
            Length(
                min=3,
                max=15,
                message="Username must be between 3 and 15 characters long. Please choose a different one!",
            ),
            validate_username_update,
        ],
    )
    email = StringField(
        label: "Email",
        validators=[
            DataRequired(message="Email is required. Please enter your email!"),
            Email(),
            validate_email_update,
        ],
    )
    submitAccountSettings = SubmitField("Save")


# Form for changing user password
2 usages  Ivan Kojić
class UserChangePasswordForm(FlaskForm):
    """
    A FlaskForm class for changing a user's password.

    Attributes:
        password (PasswordField): Field to input the new password.
        confirm_password (PasswordField): Field to confirm the new password.
        submitChangePassword (SubmitField): Button to submit password change.
    """

    password = PasswordField(
        label: "Password",
        validators=[
            DataRequired(message="Password is required. Please enter your password!"),
            validate_password,
        ],
    )
    confirm_password = PasswordField(
        label: "Confirm Password",
        validators=[
            DataRequired(message="Please retype your password!"),
            EqualTo( fieldname: "password", message="Passwords must match!"),
        ],
    )
    submitChangePassword = SubmitField("Change Password")
```

Slika 36 Definiranje forme za promjenu korisničkih podataka - *UserAccountSettingsForm* i *UserChangePasswordForm* klase



Slika 37 Prikaz formi za promjeni korisničkih podataka - *UserAccountSettingsForm* i *UserChangePasswordForm* klase

Na slikama 32 do 37 prikazan je način kreiranja formi za prijavu i registraciju korisnika te njihov izgled. Svaki atribut pripadajuće klase odgovara polju forme, pa tako *StringField* postaje polje za unos teksta, a *PasswordField* polje za unos lozinke. Svaki atribut ima parametre kao što su naziv (*label*) i funkcije validacije tog polja.

WTForms biblioteka sadržava i validatore koji su korišteni kako bi uneseni podaci bili ispravni prije rada s istima ili prije interakcije s bazom podataka.

Neki od validatora korištenih u izradi aplikacije su:

- *DataRequired* – validator koji provjerava popunjenost polja forme
- *Length* – validator koji provjerava broj znakova unesenih u polje forme
- *Email* – validator koji provjerava odgovara li unos email formatu
- *EqualTo* – validator koji provjerava odgovara li vrijednost unosa isti u više polja forme. Navedeni validator korišten je kod *password* i *confirm_password* polja u kojima je potrebno unijeti lozinku te ponoviti isti unos kao potvrdu.

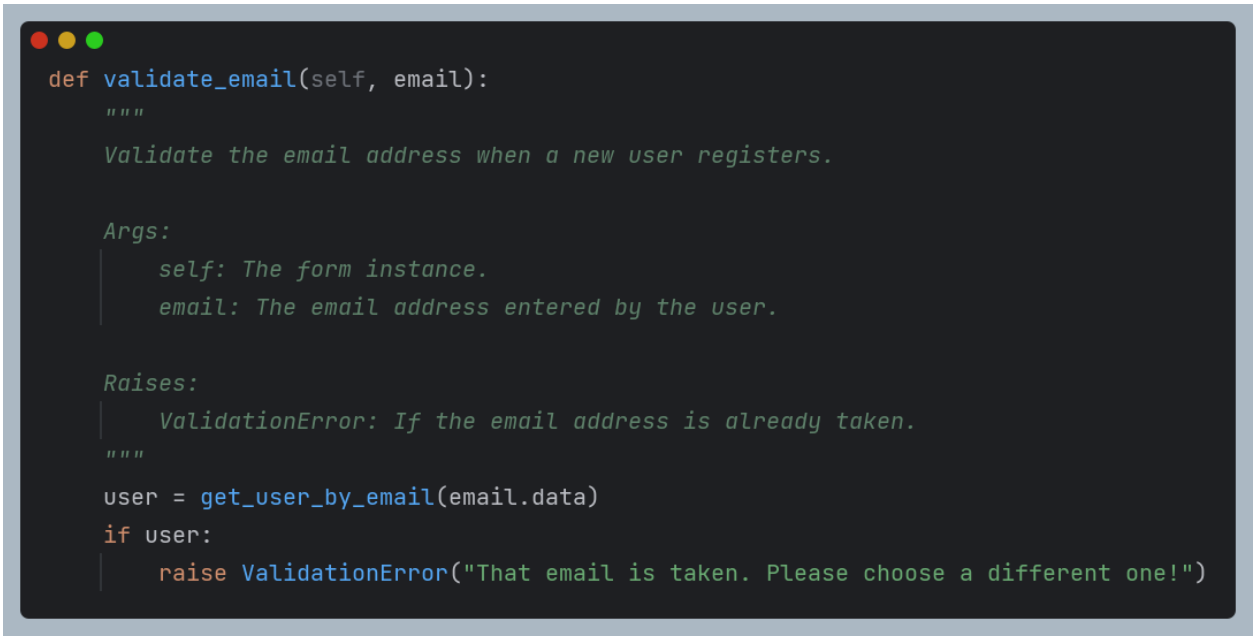
Također, izrađeni su i vlastiti validatori kao što su:

- *validate_username* – validator koji provjerava zauzetost korisničkog imena prilikom registracije korisnika
- *validate_email* – validator koji provjerava zauzetost email adrese prilikom registracije korisnika
- *validate_password* – validator koji provjerava odgovara li unesena lozinka potrebnom formatu. Potrebno je imati najmanje pet znakova, jedno veliko slovo i jednu znamenku.
- *validate_username_update* – validator koji provjerava zauzetost korisničkog imena prilikom mijenjanja korisničkog imena korisnika
- *validate_email_update* - validator koji provjerava zauzetost email adrese prilikom mijenjanja email adrese korisnika

Navedeni validatori, prikazani na slikama 38 do 42, kao parametar primaju polje forme. Kod validacije korisničkog imena i email adrese, provjerava se postojanje istih pozivom funkcije koja sadrži upit na bazu te ovisno o tome provjeravaju unos navedenog polja. Kod validacije lozinke unos polja provjerava se na tri načina, te ovisno o rezultatu prikazuju grešku.

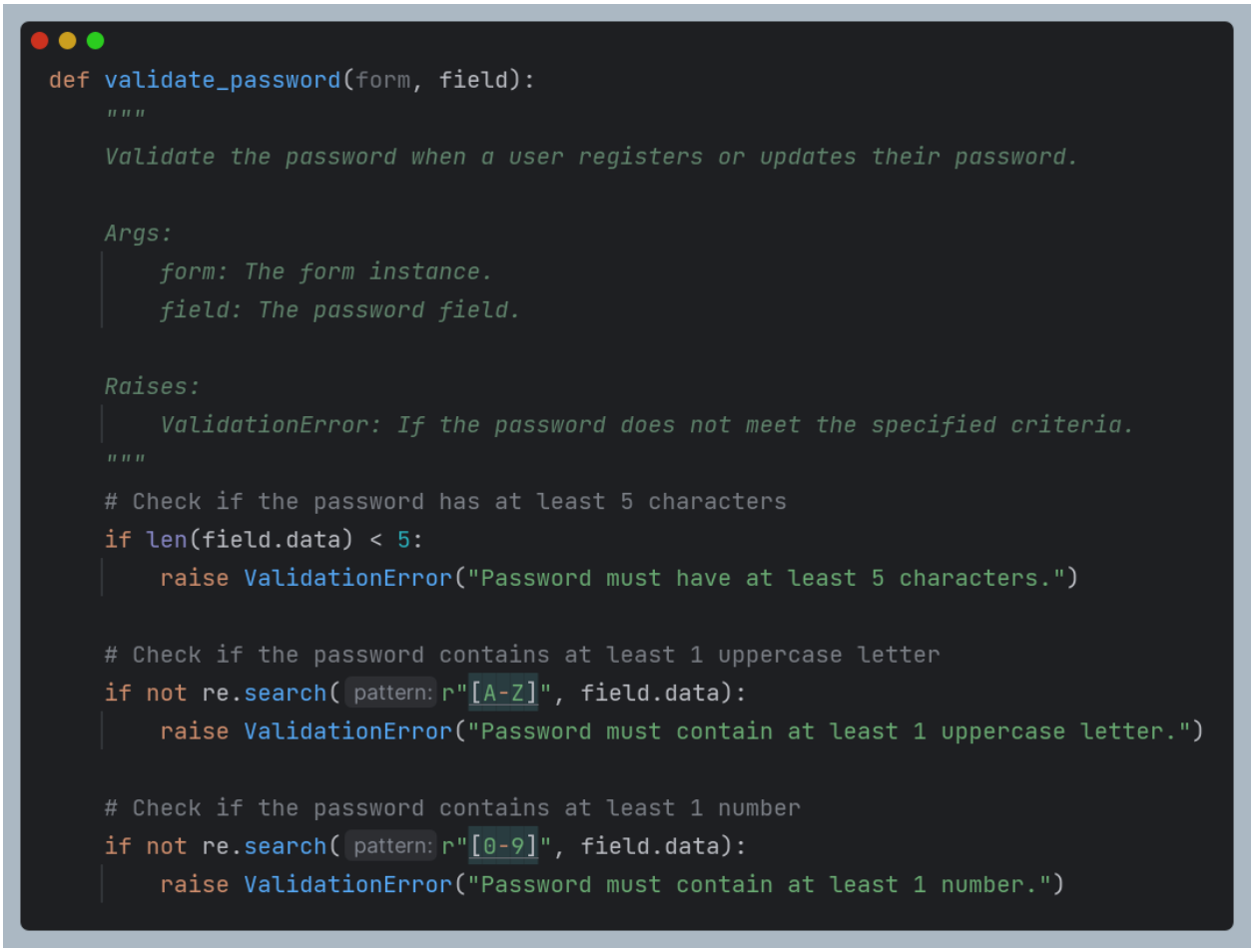
```python
def validate_username(self, username):
    """
    Validate the username when a new user registers.

    Args:
        self: The form instance.
        username: The username entered by the user.

    Raises:
        ValidationError: If the username is already taken.
    """
    user = get_user_by_username(username.data)
    if user:
        raise ValidationError("That username is taken. Please choose a different one!")
```

Slika 38 Validator za provjeru zauzetosti korisničkog imena



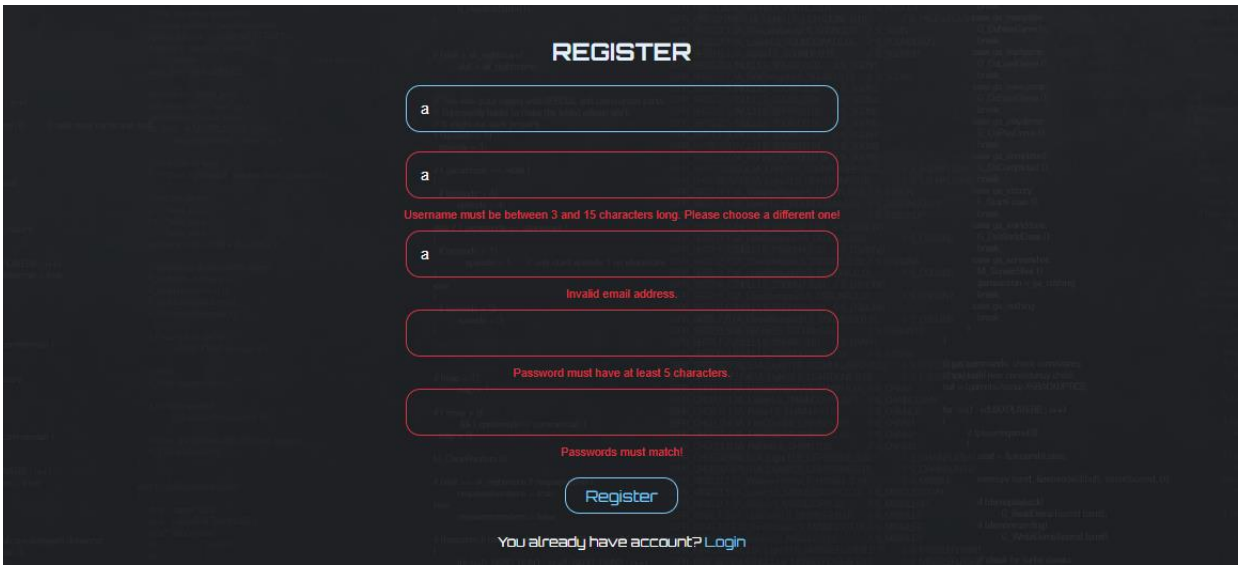
```python
def validate_email(self, email):
    """
    Validate the email address when a new user registers.

    Args:
        self: The form instance.
        email: The email address entered by the user.

    Raises:
        ValidationError: If the email address is already taken.
    """
    user = get_user_by_email(email.data)
    if user:
        raise ValidationError("That email is taken. Please choose a different one!")
```

Slika 39 Validator za provjeru zauzetosti email adrese korisnika

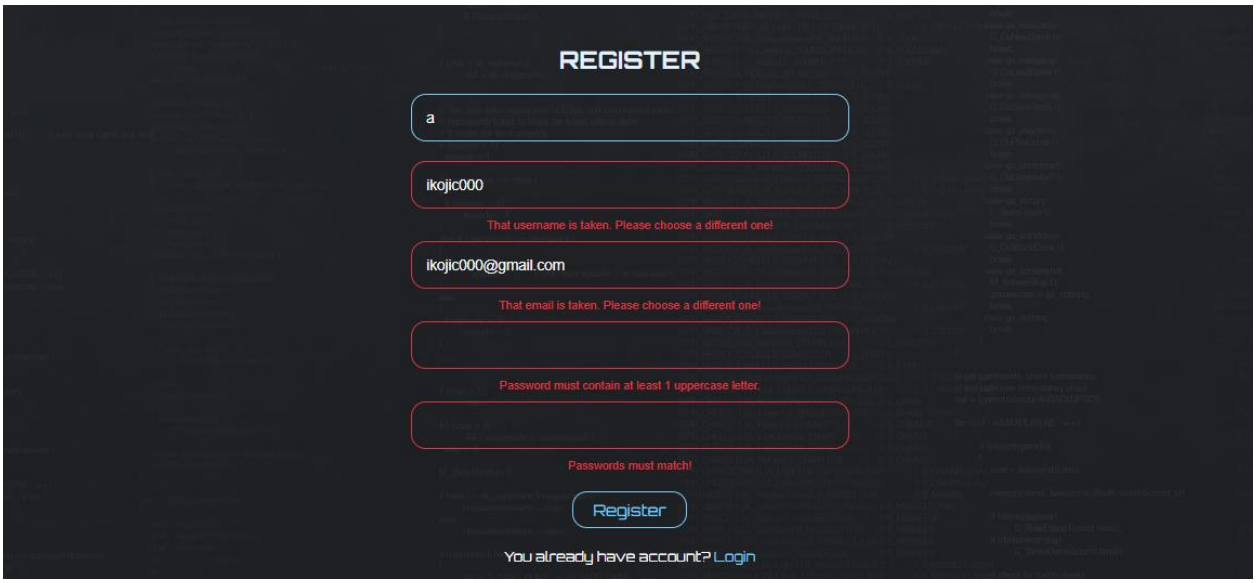


```python
def validate_password(form, field):
    """
    Validate the password when a user registers or updates their password.

    Args:
        form: The form instance.
        field: The password field.

    Raises:
        ValidationError: If the password does not meet the specified criteria.
    """
    # Check if the password has at least 5 characters
    if len(field.data) < 5:
        raise ValidationError("Password must have at least 5 characters.")

    # Check if the password contains at least 1 uppercase letter
    if not re.search( pattern: r"[A-Z]", field.data):
        raise ValidationError("Password must contain at least 1 uppercase letter.")

    # Check if the password contains at least 1 number
    if not re.search( pattern: r"[0-9]", field.data):
        raise ValidationError("Password must contain at least 1 number.")
```

Slika 40 Validator za provjeru valjanosti lozinke

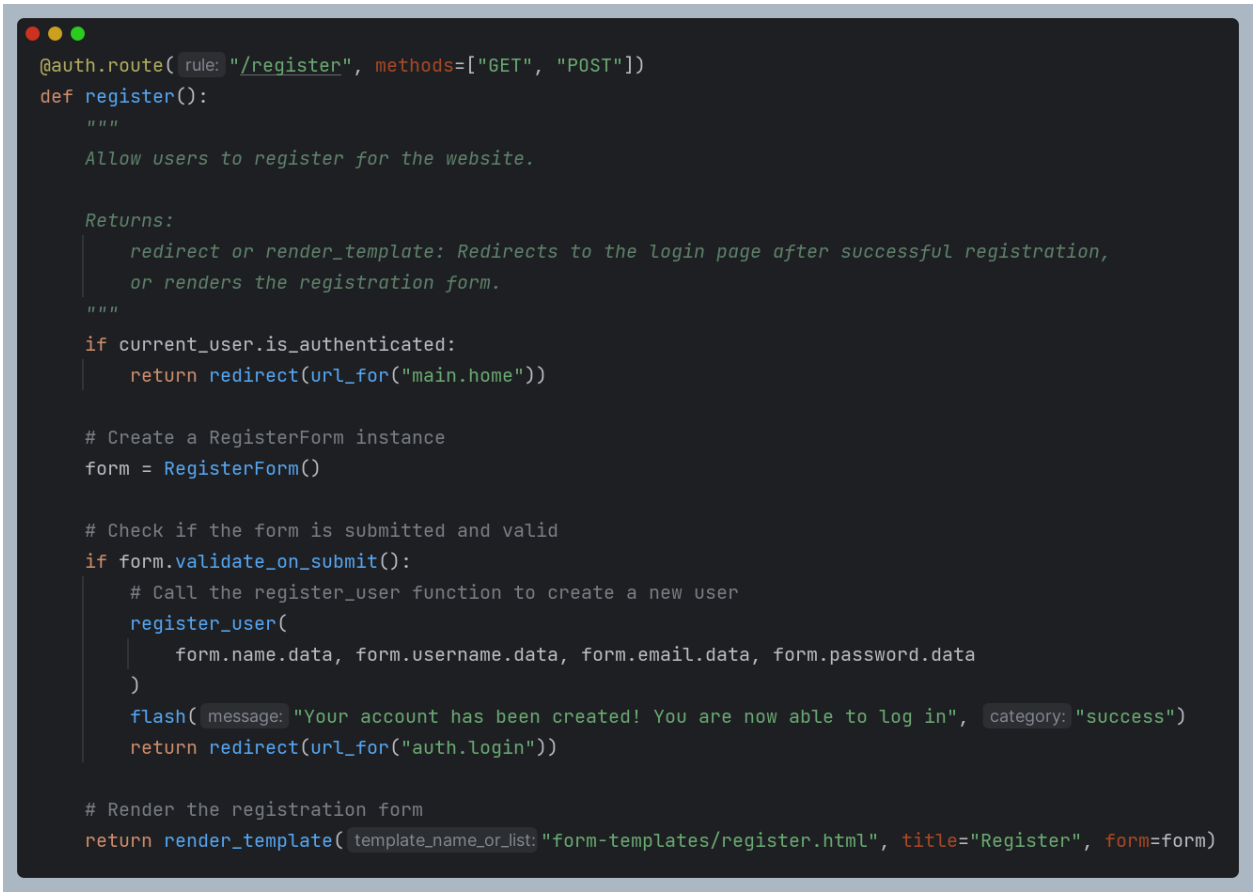


Slika 41 Prikaz unosa krivih podataka u formu koja koristi napravljene validatore



Slika 42 Prikaz unosa krivih podataka u formu koja koristi napravljene validatore

Na slikama 41 i 42 prikazane su forme s unesenim neispravnim podacima. Greške koje su prikazane korisniku su one definirane u gore navedenim validatorima.



```python
@auth.route( rule: "/register", methods=["GET", "POST"])
def register():
    """
    Allow users to register for the website.

    Returns:
        redirect or render_template: Redirects to the login page after successful registration,
        or renders the registration form.
    """
    if current_user.is_authenticated:
        return redirect(url_for("main.home"))

    # Create a RegisterForm instance
    form = RegisterForm()

    # Check if the form is submitted and valid
    if form.validate_on_submit():
        # Call the register_user function to create a new user
        register_user(
            form.name.data, form.username.data, form.email.data, form.password.data
        )
        flash( message: "Your account has been created! You are now able to log in", category: "success")
        return redirect(url_for("auth.login"))

    # Render the registration form
    return render_template( template_name_or_list: "form-templates/register.html", title="Register", form=form)
```

Slika 43 Primjer korištenja *validate_on_submit()* funkcije koja uz pomoć zadanih validatora provjerava ispravnost unosa

Na slici 43 prikazano je korištenje funkcije *validate_on_submit()*. Navedena funkcija dio je *FlaskForm* klase te poziva sve validatore na svim poljima forme na koju se poziva navedena funkcija. Kod unutar uvjeta izvršava se samo ako svi unosi u formi zadovolje validaciju.

4.2.5 Prijava i autorizacija korisnika

Jedna od glavnih značajki svake moderne web aplikacije su upravo prijava i autorizacija korisnika. Naime, registracijom, odnosno kreiranjem vlastitih korisničkih računa, korisnici čitatelji otvaraju opciju ostavljanja komentara. S druge strane, administratori CMS sustava moraju imati korisničke račune s odgovarajućim ulogama kako bi upravljali sadržajem na portalu. Uloge koje korisnici mogu imati su *Reader*, *Admin* i *Superadmin*. *Reader* ulogu ima svaki novoregistrirani korisnik, dok su *Admin* i *Superadmin* uloge ostavljene za administratore CMS sustava. Administratori mogu dodavati, ažurirati i brisati sadržaj, imaju pregled svih korisnika kojima mogu dodavati ili ukloniti prava te ih brisati, kao i pregled svih komentara koje također mogu brisati. Administratori sa *Superadmin* ulogama jedini imaju pristup zapisima zahtjeva te zapisima grešaka sustava.

Za prijavu i autorizaciju korisnika korištene su biblioteke Flask-Login i Flask-User. Flask-Login biblioteka omogućuje upravljanje korisničkim sesijama te prijavu i odjavu korisnika. Flask-User biblioteka također omogućuje prijavu i autorizaciju korisnika, kao i upravljanje korisnicima. Flask-User korišten je kako bi se korisnicima omogućilo resetiranje i povrat izgubljene lozinke. Nadalje, Flask-User ima mogućnost autorizacije putem uloga. Implementiranje toga vrši se dodjeljivanjem *@roles_required* dekoratera.

S obzirom da određene rute omogućuju korisnicima mijenjanje vlastitog korisničkog računa, bilo je potrebno napraviti vlastiti dekorater *own_account_required*.

```python
def own_account_required(view_func):
    """
    Decorator that checks if the current user owns the requested account.

    This decorator is used to restrict access to views where users can update their own account settings,
    ensuring that only the user who owns the account can access and modify their settings.

    Args:
        view_func: The view function being decorated.

    Returns:
        function: A decorated view function.
    """


    Ivan Kojić
    @wraps(view_func)
    def decorated_view(*args, **kwargs):
        # Get the user_id from the route parameters
        user_id = kwargs.get("user_id")

        # Check if the current user is authenticated and owns the requested account
        if current_user.is_authenticated and current_user.id == user_id:
            return view_func(*args, **kwargs)
        else:
            # If not authenticated or authorized, flash a message and redirect to the login page
            flash("You must be logged in and authorized to access this page.")
            return redirect(
                url_for("auth.login")
            )  # You can change 'login' to your login route


    return decorated_view
```

Slika 44 Prikaz funkcije za izradu *own_account_required* dekoratera

Navedena funkcija dekoratera iz parametara rute čita ID korisnika koji joj pokušava pristupiti. Nakon provjere autentificiranosti odgovara li ID prijavljenog korisnika ID-ju korisnika koji joj pokušava pristupiti, korisnik može pristupiti traženoj ruti ovisno o rezultatima provjere.

S obzirom da većina korisnika za registraciju i prijavu bira svoj Google ili Facebook račun, koristeći Flask-OAuthlib biblioteku, korisnicima je omogućeno da svoje korisničke račune naprave upravo na jedan od navedenih načina.

Slika 45 Rute i funkcije za autentifikaciju i autorizaciju putem Google vanjskog servisa

Na slici 45 prikazane su dvije rute. Odlaskom na rutu za prijavu putem Google računa, korisnik je preusmjeren na Google prijavu, a nakon prijave preusmjeren je na *google_authorized()* rutu. U navedenoj ruti najprije se vrši provjera postojanja odgovora s Google servisa. Ukoliko je prijava bila uspješna i Google servis vrati odgovor, izdvajaju se odgovarajući korisnički podaci te se pomoću njih provjerava postojanje korisnika u bazi podataka. Ukoliko postoji, korisnik biva prijavljen, no ukoliko ne postoji kreira se račun. Nakon prve prijave, bilo putem Google ili Facebook računa, korisnik zaprima email poruku dobrodošlice i svoje korisničke podatke za prijavu.

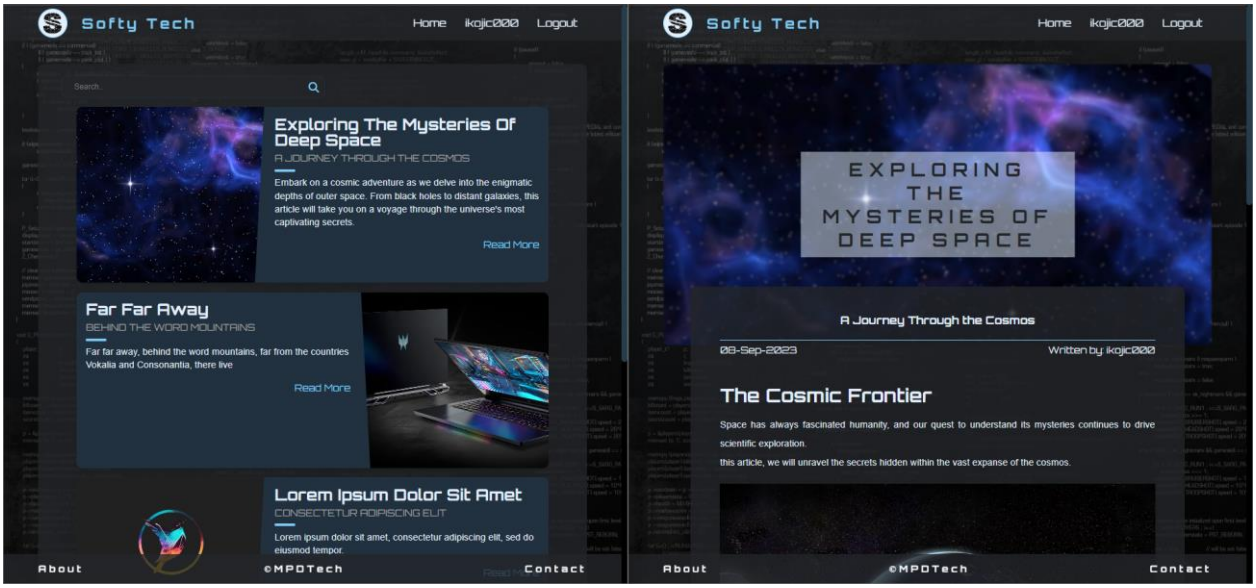
Korištenjem navedenih vanjskih servisa korisnici imaju mogućnost prijave na portal na tri načina - putem korisničkog računa napravljenog izravno na portalu te putem Google ili Facebook računa. Također, svaki novoregistrirani korisnik na svoju email adresu zaprima poruku dobrodošlice.

4.2.6 Dizajn korisničkog sučelja

Kako bi se izbjeglo korištenje gotovih rješenja, izrađen je vlastiti dizajn. Naime, namjena izrađenog web portala je objavljivanje sadržaja vezanog za računalnu i *gaming* tematiku, stoga je dizajn prilagođen ovom konceptu koristeći tamne boje, pri čemu je svijetloplava odabrana kao primarna boja.

Dizajn web portala namijenjenog posjetiteljima napravljen je ručno koristeći CSS stilski jezik, kako bi HTML elementi poprimili odgovarajući izgled.

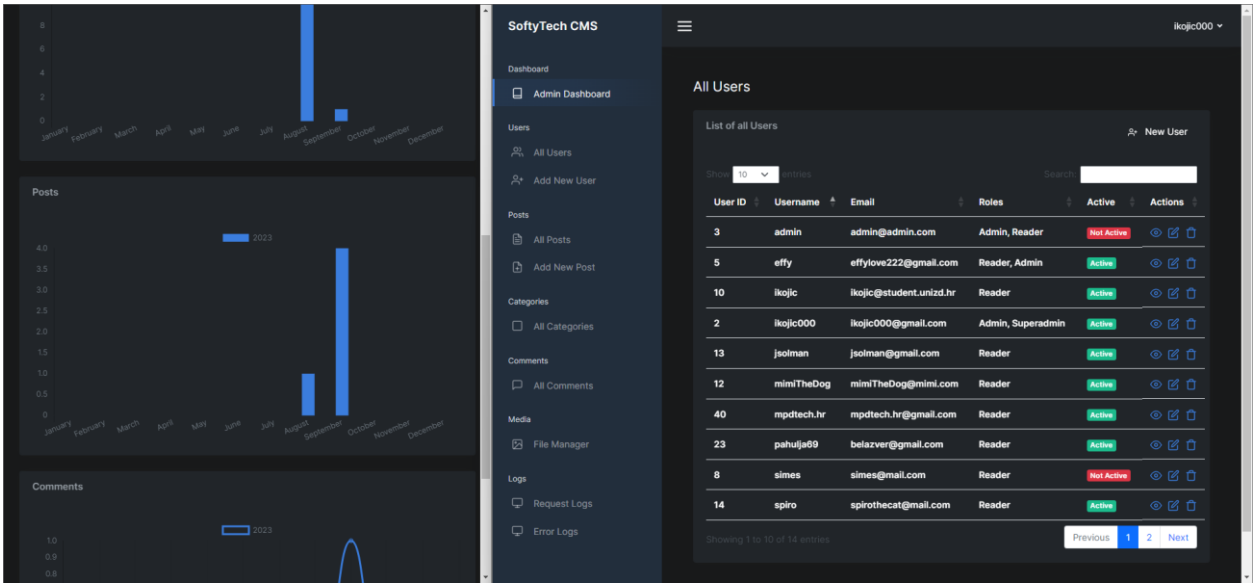
CSS selektori podijeljeni su na nekoliko CSS datoteka, kako bi se osiguralo lakše praćenje razvoja te kako bi se olakšalo daljnje održavanje, a navedene datoteke smještene su u mapu *Static* u kojoj se ujedno nalaze se i Javascript datoteke. Za raspored elemenata koristio se Bootstrap CSS okvir, točnije elementi Bootstrap-Grid, kao što su redci (eng. *Rows*) i stupci (eng. *Columns*). Korištenjem Bootstrap okvira postignut je responzivni dizajn, kako bi web portal bio upotrebljiv na svim uređajima.



Slika 46 Prikaz izgleda korisničkog sučelja

Admin korisničko sučelje dizajnirano je pomoću AdminKit predloška. AdminKit koristi Bootstrap kao osnovu za svoje elemente i klase što ga ujedno čini kompatibilnim s Bootstrap klasama, a zahvaljujući tome i admin korisničko sučelje postalo je responzivno. Kako bi administratori portala i korisnici CMS sustava imali bolji uvid u brojčano stanje portala, uključujući broj korisnika, članaka te komentara, početna stranica admin korisničkog sučelja sadrži i grafove. Korištena je Chart.js Javascript biblioteka za prikaz grafova koja koristeći AJAX i GET pozive na API metode, navedene grafove opskrbljuje s točnim podacima, odnosno brojkama iz baze podataka. Admin korisničko sučelje sadrži i brojne tablice koje omogućuju prikaz podataka napravljene pomoću DataTables biblioteke.

Kod projekta nalazi se u GitHub repozitoriju pod nazivom *SoftyTechCMS* dostupnom na web adresi https://github.com/ikojic000/SoftyTechCMS.

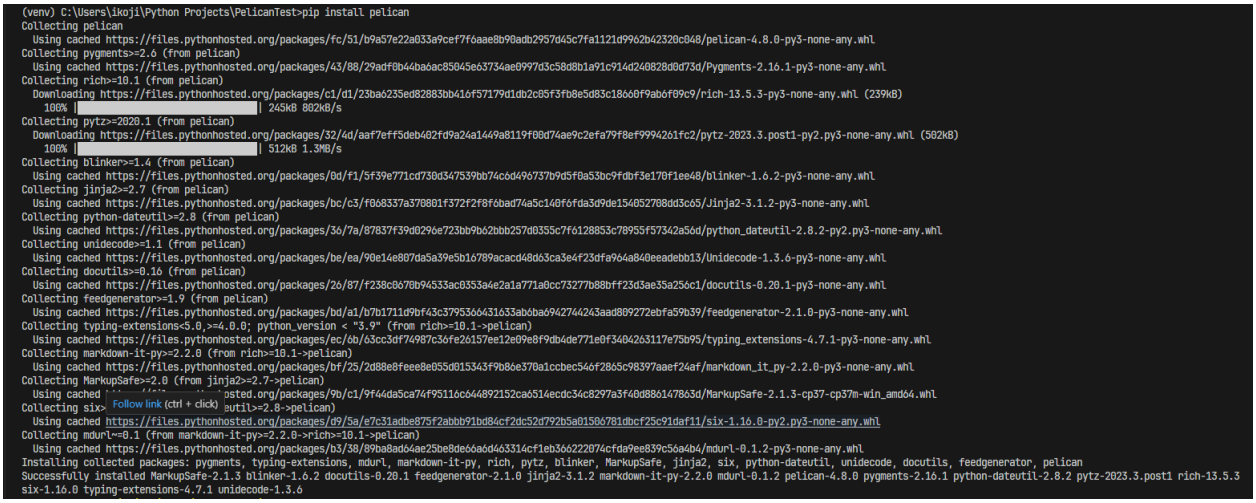


Slika 47Prikaz izgleda admin grafičkog sučelja

4.3 Razvoj web portala s Pelican generatorom statičnih stranica

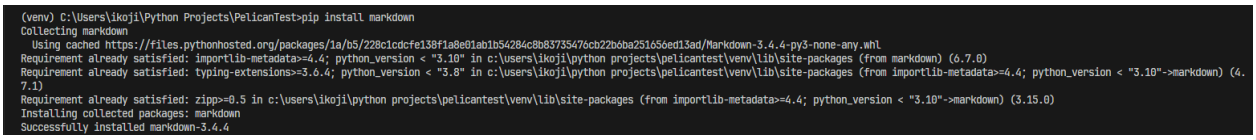
Zbog svoje jednostavnosti i brzine razvijanja projekta, Pelican se natječe za jednog od najboljih generatora statičnih stranica te se može nazvati najboljim generatorom statičnih stranica u Python okruženju.

Prateći službenu dokumentaciju, u samo nekoliko minuta Pelican generator statičnih stranica je spreman. Koristeći naredbu *pip install pelican*, Pelican biva instaliran u virtualnom okruženju projekta.



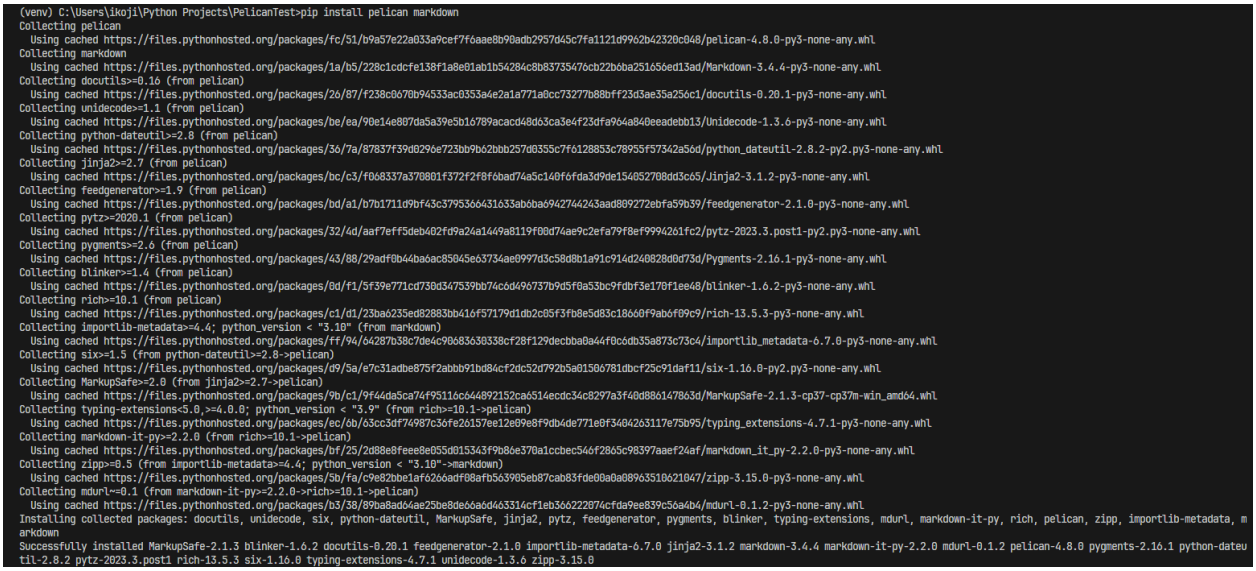
Slika 48 Konzolni prikaz instalacije Pelican-a

S obzirom da Pelican koristi markdown datoteke za generiranje sadržaja potrebno je pokrenuti i *pip install markdown* naredbu.



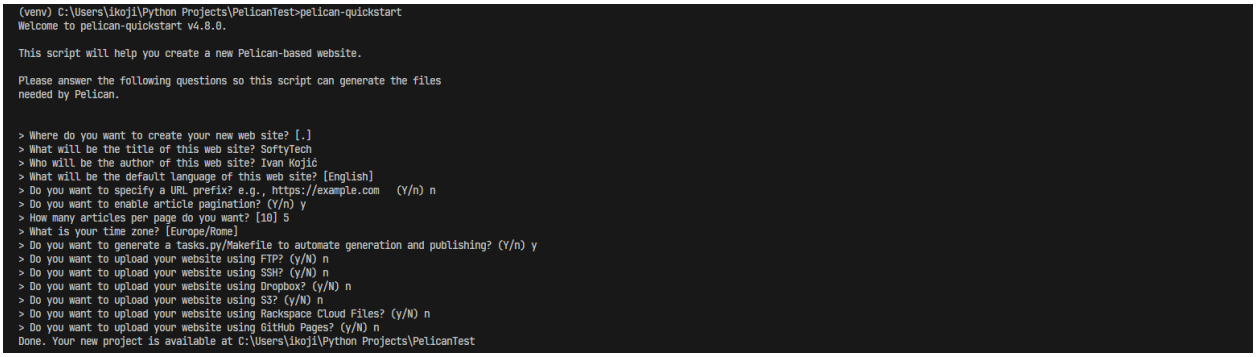
Slika 49 Konzolni prikaz instalacije markdown biblioteke

Navedene naredbe mogu se kombinirati, stoga umjesto pojedinačno možemo pokrenuti naredbu *pip install pelican markdown* za instaliranje svih potrebnih biblioteka.



Slika 50 Konzolni prikaz instalacije svih potrebnih biblioteka za rad s Pelican-om

Jednostavnost korištenja i razvoja Pelican pokazuje već na samom početku s naredbom *pelican-quickstart*. Navedena naredba vodi nas kroz niz od nekoliko pitanja potrebnih za kreiranje projekta. Pitanja su ključna za inicijalno postavljanje projekta, međutim, sve se postavke projekta naknadno mogu promijeniti. Pitanja su prikazana na slici 51.



Slika 51 Konzolni prikaz početnog postavljanja pomoću naredbe *pelican-quickstart*

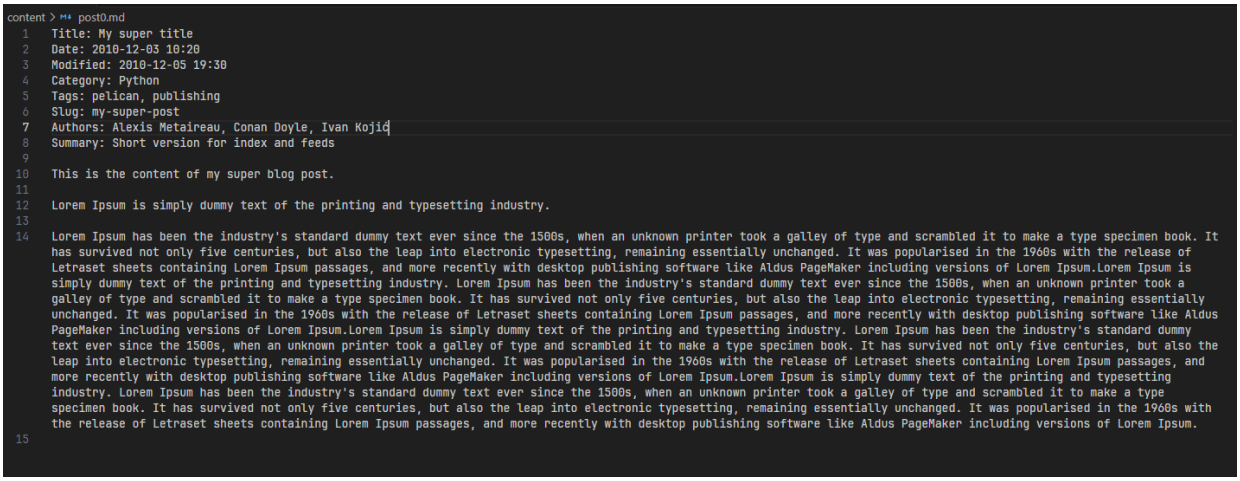
Nakon navedenih pitanja struktura projekta je automatski generirana te sadrži sve potrebno za pisanje i generiranje vlastitog sadržaja.

```
yourproject/
├── content
│   └── (pages)
├── output
├── tasks.py
├── Makefile
├── pelicanconf.py       # Main settings file
└── publishconf.py       # Settings to use when ready to publish
```

Slika 52 Prikaz strukture Pelican projekta

Dodavanje sadržaja vrši se na način da se sadržaj napisan u markdown formatu smješta unutar mape *content*. Svaka markdown datoteka predstavlja jednu podstranicu (eng. *Subpage*) u projektu. Markdown datoteka sastavljena je od dva dijela - meta podaci te sadržaj generirane stranice. Meta podaci (eng. *Metadata*) smješteni su na vrhu iznad samog sadržaja te prate uzorak *Naziv:Vrijednost* ukoliko se radi o Markdown formatu, te *:naziv: vrijednost* ukoliko se radi o reStructuredText formatu.

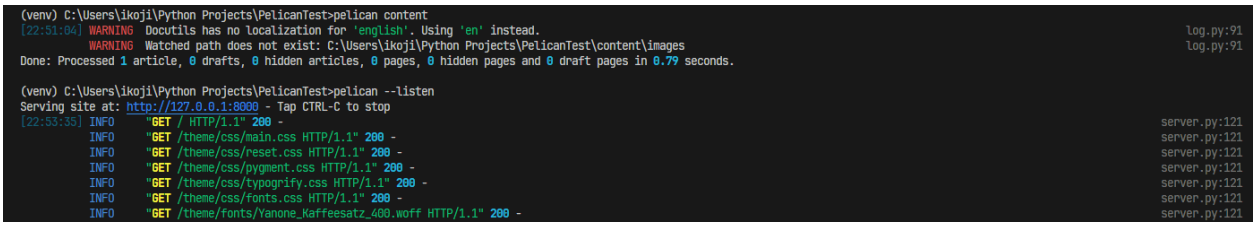
Primjer jedne takve stranice u markdown formatu prikazan je na slici 53.



Slika 53 Prikaz primjera stranice u markdown formatu

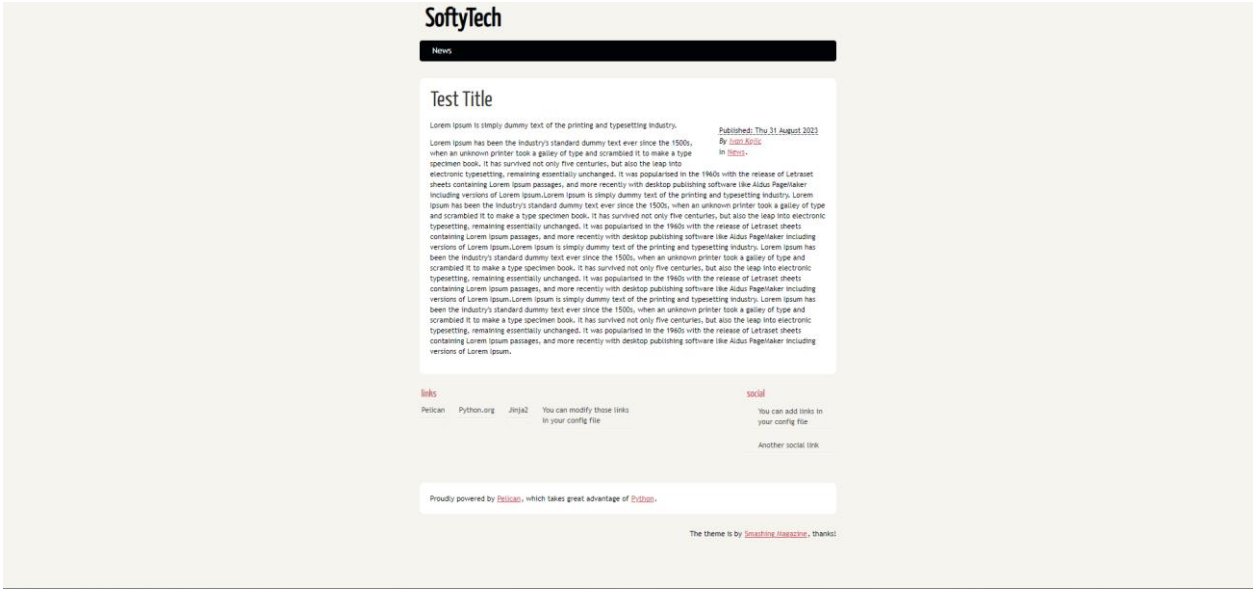
Nakon dodavanja sadržaja potrebno je pokrenuti naredbu *pelican content* kako bi Pelican generirao stranice u *output* mapu.

Izgled generirane stranice, odnosno web portala ili bloga, možemo vidjeti pokretanjem naredbe *pelican –listen*.



Slika 54 Konzolni prikaz generiranja sadržaja te pokretanja lokalnog servera

Odlaskom na adresu *http://127.0.0.1:8000/* prikazanu u konzolnom izlazu može se vidjeti izgled generiranog portala sa stranicama spremljenim u *content* mapi.



Slika 55 Prikaz generiranog portala s zadanom temom

4.3.1 Dizajn korisničkog sučelja

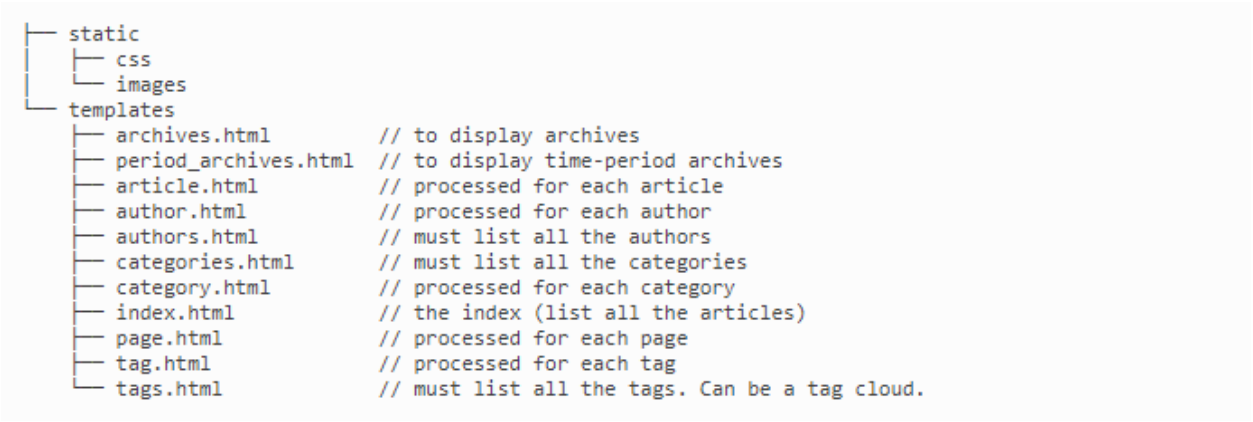
Prema današnjim standardima modernih web aplikacija i stranica, zadana tema koju Pelican automatski generira može se smatrati zastarjelom i neusklađenom s vlastitim dizajnom. S obzirom na ovu situaciju, Pelican nudi mogućnost korištenja različitih tema koje su izrađene od strane drugih developera. Navedene teme mogu se pronaći na GitHub repozitoriju pod nazivom *pelican-themes* na web adresi https://github.com/getpelican/pelican-themes, na web stranici *pelican-themes Preview* na web adresi https://pelicanthemes.com, ali i na mnogim drugima.

Kako bi generirane stranice poprimile novi izgled, potrebno je dodati jednu liniju koda u datoteci *pelicanconf.py* s putanjom do željene teme. Navedena postavka prikazana je na slici 54. Nakon tog koraka sve što preostaje je pokrenuti naredbu *pelican content* kako bi se stranice nanovo generirale.

Slika 56 Prikaz mijenjanja Pelican teme

Međutim, istraživanjem već postojećih tema nije pronađena niti jedna koja izgledom odgovara dizajnu napravljenog web portala i pripadajućeg CMS sustava.

Unatoč tome, izrada vlastite teme nije kompliciran i zahtjevan proces koji je detaljno opisan u službenoj dokumentaciji Pelican-a. Za izradu teme koristi se Jinja sustav za predloške. Sve datoteke teme potrebno je smjestiti u jednu mapu te se držati odgovarajuće strukture mape prikazane na slici 57.



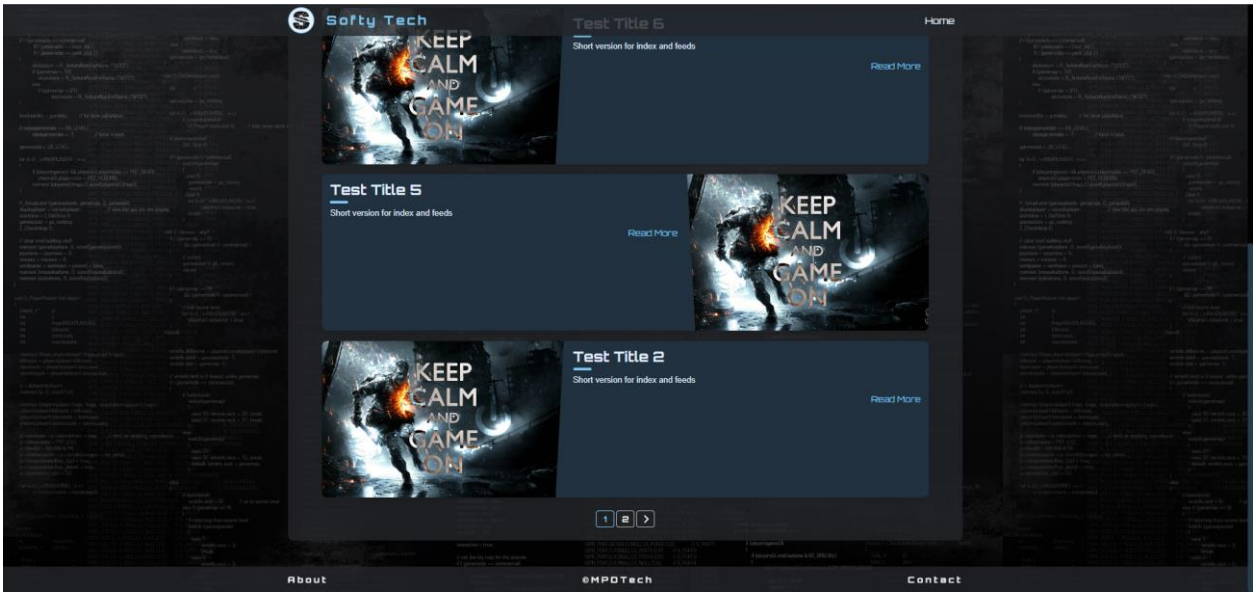
Slika 57 Prikaz strukture mape s datotekama za izradu Pelican teme

Promatrajući strukturu mape Pelican teme može se primijetiti kako svaki tip stranice koja može biti generirana ima svoj predložak.

Predlošci za stranice potrebne za izradu ovog projekta, odnosno stranice koje će biti prikazane korisniku, uključujući naslovnu stranicu, stranicu pojedinačnog članka/objave te info stranice, smješteni su u mapu *softyTechTheme* čija je struktura prikazana na slici 58.

Slika 58 Prikaz strukture mape softyTechTheme teme

Postavljanjem vlastite teme generirani sadržaj poprima odgovarajući dizajn prikazan na slici 59.



Slika 59 Prikaz izgleda naslovne stranice generiranog sadržaja

4.3.2 Dodatci

Rezultat generiranja stranice Pelican generatorom skup je statičnih stranica. Korištenjem dodataka (eng. *Plugin*) statične stranice mogu imati neke značajke dinamičkih stranica, ali i dalje ostati statične.

Korisnicima je potrebno omogućiti značajke poput ostavljanja komentara te mogućnost kontaktiranja putem kontakt forme.

Jedan od najpopularnijih dodataka za ostavljanje komentara je Disqus. Kako bi navedeni dodatak bio integriran na statične stranice potrebno se registrirati na službenoj Disqus stranici, te registrirati web stranicu. Prateći dokumentaciju dodatka u *article.html* predlošku, potrebno je staviti kratki Javascript isječak koji povezuje web stranicu s Disqus portalom prikazan na slici 60.

```
(function () { // DON'T EDIT BELOW THIS LINE
    var d = document, s = d.createElement('script');
    s.src = 'https://softytech.disqus.com/embed.js';
    s.setAttribute('data-timestamp', +new Date());
    (d.head || d.body).appendChild(s);
})();
```

Slika 60 Prikaz Javascript funkcije potrebne za dodavanje Disqus komentara

Rezultat svega navedenog je dodatna funkcionalnost koja korisnicima omogućuje ostavljanje komentara, a administratorima i autorima sadržaja administraciju ostavljenih komentara putem Disqus platforme.

FormKeep je također dodatak koji proširuje funkcionalnosti statičnih web stranica. Naime, nakon registracije na službenom FormKeep portalu, dodavanja te postavljanja postavki za kontakt formu dobiva se URL kojeg je potrebno dodati u predložak navedene forme prikazan na slici 61. Generirani rezultat navedene kontakt forme i korištenja FormKeep platforme korisnicima daje mogućnost kontaktiranja autora web portala, dok administratori imaju uvid u pristigle poruke.

Kod projekta nalazi se u GitHub repozitoriju pod nazivom *SoftyTech_Pelican* dostupnom na web adresi https://github.com/ikojic000/SoFtYTech_Pelican.

```
<form action="https://formkeep.com/f/7533dfcdb21d" accept-charset="UTF-8" enctype="multipart/form-data"
      method="POST">
    <p>Feel free to contact us..</p>

    <input class="form-input " id="name" name="name" placeholder="Enter your name..." required=""
           type="text"
           value="">

    <input class="form-input" id="email" name="email" placeholder="Enter your email..." required=""
           type="text"
           value="">

    <input class="form-input" id="subject" name="subject" placeholder="Enter message subject..." required=""
           type="text" value="">

    <textarea class="form-input form-textarea" id="message" name="message"
              placeholder="Type your message..."
              required=""></textarea>

    <input class="form-submit-button" id="submit" name="submit" type="submit" value="Send">
</form>
```

Slika 61 Prikaz predloška kontakt forme na statičnoj stranici koristeći FormKeep platformu

5 Usporedba

Usporedba između CMS sustava i generatora statičnih stranica igra važnu, ako ne i ključnu ulogu u procesu odlučivanja između navedenih načina izrade web stranica i objavljivanja sadržaja na istima. Svaki od tih alata ima jedinstvene karakteristike, prednosti i mane koje utječu na funkcionalnost, performanse i upravljivost web stranica.

5.1 Prednosti i mane CMS sustava

Web je prepun raznih CMS sustava što ukazuje na njihovu popularnost i široku prihvaćenost upravo zbog svoje jednostavnosti i praktičnosti. Glava prednost gotovih CMS sustava je upravo ta fleksibilnost i jednostavnost u korištenju, ali važno je napomenuti da osim gotovih opcija poput Wordpress-a, postoji i opcija izrade vlastitog CMS sustava čije se značajke određuju po vlastitom nahođenju i potrebama. Ono što ih čini posebnima je njihova intuitivna priroda koja pruža mogućnost objavljivanja i uređivanja vlastitog sadržaja u jako kratkom vremenu bez ikakvog programerskog znanja. Također, korisničko sučelje CMS sustava u pravilu je oblikovano s korisnikom na umu te ima jasne i jednostavne alate i forme za dodavanje, uređivanje i brisanje sadržaja. Pojedinci i organizacije čija potreba je često i brzo objavljivanje sadržaja te informacija ili vijesti mogu brzo ažurirati svoje stranice bez potrebe za zapošljavanjem ili angažiranjem tima developera. Laka integracija različitih komponenti, poput e-trgovine ili društvenih mreža na postojeću web stranicu, proširuje njene funkcionalnosti te omogućuje bolju povezanost s korisnicima te stranice koji uvijek imaju svjež sadržaj. Slijedom navedenog, jedna od ključnih i značajnih prednosti CMS sustava je upravo mogućnost dinamičkog sadržaja.

Unatoč brojnim prednostima, CMS sustavi imaju i svoje mane. Postavljanje i konfiguriranje CMS sustava vremenski može biti jako zahtjevan i izazovan proces, naočito ako se postavlja velik broj značajki te ukoliko to rade ljudi bez programerskog posla. Samim time, neizbježan je gubitak vremena i resursa tijekom upoznavanja rada navedenog sustava. Također, CMS sustavi mogu biti podložni napadima stoga zahtijevaju i visoku razinu sigurnosti. Nadalje, CMS sustavi oslanjaju se na razne dodatke kako bi proširili svoje funkcionalnosti, a samim time korisnici su primorani dodatno istražiti svoje opcije, što također otežava postavljanje CMS sustava. Jedna od naglašenih mana ovog sustava je i brzina učitavanja sadržaja te ovisnost CMS sustava o hardverskim specifikacijama servera na kojem se nalazi, a upravo je to jedan od glavnih razloga zašto se developeri okreću drugim načinima upravljanja sadržajem.

Međutim, kao što je već spomenuto, uz gotova rješenja postoji i opcija izrade vlastitog CMS sustava. Iako je sama izrada dugotrajan i zahtjevan proces, krajnji rezultat može biti naročito pozitivan. Glavna prednost izrade vlastitog CMS sustava je njegova personalizacija i mogućnost potpune prilagodbe i kontrole nad funkcionalnostima i značajkama, kao i nad sigurnosti web stranice. Developeri prilikom izrade sami biraju programski jezik, bazu podataka te web okvir koji najbolje odgovara njima samima, te onaj koji najbolje rješava probleme tijekom izrade funkcionalnosti. Također, omogućeno je biranje prijave i upravljanjem korisnika izradom vlastitog sigurnosnog sustava ili oslanjanjem na postojeće davatelje usluga. Uz navedene prednosti, bitan faktor je i mogućnost dizajniranja po vlastitim smjernicama. Tijekom planiranja te same izrade CMS sustava developeri imaju potpunu slobodu u odabiru funkcionalnosti te alata potrebnih za izradu.

Međutim, uz sve navedene prednosti, glavni nedostaci izrade vlastitog CMS sustava su vrijeme i tehničko znanje. Kako se i najsitniji detalji prilagođavaju po vlastitoj potrebi, izrada postaje dugotrajan proces. Također, izrada zahtjeva iznimno znanje programiranja i razvoja web aplikacija. Ukoliko se za izradu vlastitog CMS sustava unajmi tim developera, razvoj vlastitog CMS sustava može postići jako visoku cijenu. No, nakon izrade vlastitog CMS sustava može doći i do naknadnih problema, jer takvi sustavi zahtijevaju redovito održavanje i nadogradnje kako bi održali funkcionalnost i sigurnost.

5.2 Prednosti i mane generatora statičnih stranica

Generatori statičnih stranica nastali su kao alternativa CMS sustavima upravo kako bi riješili neke od problema koji mogu biti prisutni u radu s CMS sustavima. S obzirom da generatori statičnih stranica generiraju web stranice unaprijed kao čiste HTML, CSS i JavaScript datoteke, glavna prednost generatora statičnih stranica su upravo brzina i performanse. Generiranjem unaprijed izbjegava se potreba za složenim dinamičkim obrascima, zahtjevima prema serveru poslužitelju te prema bazi podataka svaki put kada korisnik otvori stranicu, što rezultira gotovo trenutnim učitavanjem stranica bez obzira na količinu prometa. Navedena prednost je od značajne važnosti za web stranice s visokim brojem posjetitelja te za one od kojih se zahtjeva brzo učitavanje bitnih informacija. Iskustvo korištenja takvih stranica znatno se poboljšava što rezultira zadržavanjem na stranicama te smanjuje vjerojatnost napuštanja istih. Također, prednost generatora statičnih stranica leži i u jednostavnosti korištenja istih. Postavljanje generatora statičnih stranica zahtjeva manje tehničkog znanja jer se radi upravo o statičnim stranicama te nije potrebno konfigurirati bazu podataka niti složenije servere. Smanjivanje

složenosti rezultira s bržim vremenom postavljanja i nižim troškovima održavanja. Osim troškova održavanja, niski su i troškovi hostinga. Također, mnogi pružatelji usluga, kao na primjer GitHub, nude čak i besplatan hosting statičnih stranica. Osim navedenih pogodnosti, sigurnost igra veliku ulogu kao prednost generatora statičnih stranica. Naime, stranice se sastoje od statičnih datoteka i nemaju komponente za komuniciranje s bazom podataka, stoga je smanjen rizik sigurnosnih prijetnji. Ukoliko je korisnicima sigurnost na prvom mjestu, statične stranice izrađene putem generatora pružaju jedan od najsigurnijih oblika objavljivanja sadržaja te izrade web stranica.

Iako su prednosti generatora statičnih stranica značajne, postoje i bitni nedostaci te ograničenja ovog pristupa. Generatori statičnih stranica nisu prikladni za stranice koje zahtijevaju složeni dinamički sadržaj ili interakciju s korisnicima. Iako je podešavanje i generiranje stranica putem generatora jednostavan i brz proces, osiguravanje dodatnih funkcionalnosti može otežati podešavanje i produžiti vrijeme izrade. Kako bi se osigurala dodatna funkcionalnost, statične stranice oslanjaju se na razne dodatne dodatke koji sa sobom povlače određene probleme, kao što su odabir odgovarajućih, njihova cijena i slično. Izlaženje iz okvira jednostavnih stranica izrađenih putem generatora statičnih stranica često otežava njihovu izradu. Također, generatori statičnih stranica najčešće nemaju korisničko sučelje prilagođeno korisniku, već se oslanjaju na konzolni rad. Ukoliko korisnik nema nikakvog tehničkog znanja ni iskustva, isto može dovesti do zbunjenosti i poraznog osjećaja koji odbija od takvog pristupa upravljanjem sadržaja. Nadalje, problem može predstavljati i potreba za sustavom za upravljanje korisnicima, pogotovo u slučajevima kada je to jedan od zahtjeva, a generatori statičnih stranica nemaju tu mogućnost. Bitno je naglasiti i komplikacije kod velikog broja generiranih stranica. Naime, svaki put kada se doda nova stranica ili promijeni nešto na postojećima, sve stranice moraju se nanovo generirati, stoga vrijeme generiranja raste s povećanjem sadržaja i broja stranica. Generatori statičnih stranica nemaju odgovarajući sustav za upravljanje datotekama, pa s vremenom postaje sve teže i teže pratiti sve datoteke te održavati stranice i njima sve pripadajuće. Također, jedan od nedostataka je i nemogućnost objavljivanja s bilo kojeg uređaja. S obzirom da generatori nemaju administrativno sučelje i bazu podataka kojima se može pristupiti s bilo kojeg preglednika ili uređaja, potrebno je imati podešen jedan uređaj na kojem se upravlja sadržajem, generira novi te objavljuje isti.

6 Zaključak

Objavljivanje i upravljanje web sadržajem predstavlja neizostavan dio svakodnevnog digitalnog okruženja, stoga su u ovom radu uspoređena dva ključna pristupa: korištenje generatora statičnih stranica i korištenje CMS sustava. Nadalje, kroz temeljito istraživanje povijesti i budućnosti ovih pristupa te kroz praktičnu izradu web portala primjenom oba sustava, postalo je jasno da i generatori statičnih stranica i CMS sustavi dolaze sa svojim prednostima i ograničenjima te da ne postoji univerzalno ispravan izbor.

Naime, prednosti generatora statičnih stranica leže u njihovoj brzini, jednostavnosti te brzini učitavanja generiranih stranica. Također, prikladniji su za jednostavnije stranice, portale ili blogove koji ne zahtijevaju veće funkcionalnosti ili dinamičke značajke. Nadalje, zbog navedenih prednosti te njihove sigurnosti prikladni su i za tvrtke koje žele svoje informacije prezentirati svojim klijentima. No, ono gdje stranice izrađene putem generatora zaostaju, upravo su naprednije značajke koje CMS sustavi mogu lako implementirati. Osim navedenog, generatori statičnih sustava moraju biti podešeni i instalirani na jednom uređaju te nemaju mogućnost upravljanja korisnicima pa bi većim tvrtkama i organizacijama s više autora bolja opcija bila CMS sustavi koji navedeno najčešće imaju ugrađeno u sebi. Prednost CMS sustava u odnosu na generatore statičnih stranica leži i u njihovoj prenosivosti. Svim takvim sustavima može se pristupiti sa svih uređaja, što u kombinaciji sa sučeljem orijentiranim prema korisnicima uvelike olakšava upravljanje sadržajem. CMS sustavi nemaju ograničene funkcionalnosti jer se bilo kakva funkcionalnost može naknadno dodati ili promijeniti. Značajke i funkcionalnosti CMS sustava ograničene su samo s mogućnostima developera ili tehnologija s kojom su izrađeni. Međutim, bitno je naglasiti i izazove razvijanja web portala pomoću generatora statičnih stranica i razvijanja vlastitog CMS sustava. Naime, u brzini razvijanja veliku prednost imaju generatori statičnih stranica jer se jedan jednostavan portal može izraditi u jako kratkom vremenu, dok to nije slučaj kod izrade vlastitog CMS sustava koja zahtjeva puno vremena, znanja, truda te istraživanja. Također, važno je omogućiti i odgovarajuće sigurnosne značajke, što također zahtijeva određenu razinu tehničkog znanja.

Pobjednik između generatora statičnih stranica i CMS sustava ne postoji te je važno sagledati prednosti oba načina. Pojedincima koji žele što brže i jednostavnije objavljivati svoj sadržaj te upravljati istim više će odgovarati opcija generatora statičnih stranica, dok će tvrtkama i organizacijama koje objavljuju razne sadržaje poput tekstova, slika, videa, proizvoda u e-trgovini i mnoge druge, više odgovarati prednosti koje dolaze s izradom vlastitog CMS sustava.

7 Popis literature

Amsler, S. & Churchville, F., 2021. *Content management system (CMS).* [Mrežno]
Dostupno na: https://www.techtarget.com/searchcontentmanagement/definition/content-management-system-CMS
[Pokušaj pristupa 17 08 2023].

Butti, R., n.d. *Headless CMS Explained: What is it? Why does it matter?.* [Mrežno]
Dostupno na: https://www.storyblok.com/tp/headless-cms-explained#headless-cms-explained-what-is-it-why-does-it-matter
[Pokušaj pristupa 18 08 2023].

Cloudflare, n.d. *What is a static site generator?.* [Mrežno]
Dostupno na: https://www.cloudflare.com/learning/performance/static-site-generator/
[Pokušaj pristupa 15 07 2023].

DB-Engines, 2023. *DB-Engines Ranking of Relational DBMS.* [Mrežno]
Dostupno na: https://db-engines.com/en/ranking/relational+dbms
[Pokušaj pristupa 04 09 2023].

Full Stack Python, n.d. *Pelican.* [Mrežno]
Dostupno na: https://www.fullstackpython.com/pelican.html
[Pokušaj pristupa 03 09 2023].

Gartner Glossary, n.d. *Web Content Management (WCM).* [Mrežno]
Dostupno na: https://www.gartner.com/en/information-technology/glossary/wcm-web-content-management
[Pokušaj pristupa 16 08 2023].

GitHub, 2023. *GitHut 2.0.* [Mrežno]
Dostupno na: https://madnight.github.io/githut/#/pull_requests/2023/2
[Pokušaj pristupa 02 09 2023].

Hawksworth, P., 2020. *What is a Static Site Generator? And 3 ways to find the best one.* [Mrežno]
Dostupno na: https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/
[Pokušaj pristupa 16 07 2023].

Heslop, B., 2018. *History of Content Management Systems and Rise of Headless CMS.* [Mrežno]
Dostupno na: https://www.contentstack.com/blog/all-about-headless/content-management-

systems-history-and-headless-cms
[Pokušaj pristupa 18 08 2023].
Jamstack, n.d. *Pelican.* [Mrežno]
Dostupno na: https://jamstack.org/generators/pelican/
[Pokušaj pristupa 03 09 2023].
Jekyll, n.d. *Jekyll Simple, blog-aware, static sites.* [Mrežno]
Dostupno na: https://jekyllrb.com/
[Pokušaj pristupa 20 07 2023].
Kinsta, 2023. *What Is a Content Management System (CMS)?.* [Mrežno]
Dostupno na: https://kinsta.com/knowledgebase/content-management-system/
[Pokušaj pristupa 16 08 2023].
Kohan, B., 2010. *What is a Content Management System (CMS)?.* [Mrežno]
Dostupno na: https://www.comentum.com/what-is-cms-content-management-system.html
[Pokušaj pristupa 15 08 2023].
Mancini, J., 2017. *The Next Wave - Moving from ECM to Intelligent Information Management.* s.l.:an.
MOVEABLE TYPE, n.d. *WHAT IS MOVEABLE TYPE?.* [Mrežno]
Dostupno na: https://www.movabletype.org/
[Pokušaj pristupa 16 07 2023].
MySQL, n.d. *What is MySQL?.* [Mrežno]
Dostupno na: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html
[Pokušaj pristupa 04 09 2023].
Nanoc, n.d. *Why Nanoc?.* [Mrežno]
Dostupno na: https://nanoc.app/about/
[Pokušaj pristupa 17 07 2023].
Neumegen, M., 2022. *The 'After Jekyll' era.* [Mrežno]
Dostupno na: https://cloudcannon.com/blog/ssg-history-2-after-jekyll/
[Pokušaj pristupa 21 07 2023].
Neumegen, M., 2022. *The 'Before Jekyll' era.* [Mrežno]
Dostupno na: https://cloudcannon.com/blog/ssg-history-1-before-jekyll/
[Pokušaj pristupa 16 07 2023].
Optimizely, n.d. *Decoupled CMS.* [Mrežno]
Dostupno na: https://en.wikipedia.org/wiki/Monolithic_system
[Pokušaj pristupa 18 08 2023].

Pelican, n.d. *Pelican 4.8.0.* [Mrežno]
Dostupno na: https://docs.getpelican.com/en/latest/
[Pokušaj pristupa 03 09 2023].
Python, n.d. *What is Python? Executive Summary.* [Mrežno]
Dostupno na: https://www.python.org/doc/essays/blurb/
[Pokušaj pristupa 02 09 2023].
TIOBE, 2023. *TIOBE Index for September 2023.* [Mrežno]
Dostupno na: https://www.tiobe.com/tiobe-index/
[Pokušaj pristupa 02 09 2023].
Wikipedia, n.d. *Enterprise content management.* [Mrežno]
Dostupno na: https://en.wikipedia.org/wiki/Enterprise_content_management
[Pokušaj pristupa 15 08 2023].
Wikipedia, n.d. *Monolithic system.* [Mrežno]
Dostupno na: https://en.wikipedia.org/wiki/Monolithic_system
[Pokušaj pristupa 18 08 2023].
Wikipedia, n.d. *MySQL.* [Mrežno]
Dostupno na: https://en.wikipedia.org/wiki/MySQL
[Pokušaj pristupa 04 09 2023].
Wikipedia, n.d. *Python (programming language).* [Mrežno]
Dostupno na: https://en.wikipedia.org/wiki/Python_(programming_language)
[Pokušaj pristupa 02 09 2023].

Comparison of Static Site Generators and Content Management Systems

Summary

In today's digital age there is a growing need for companies, organizations, and individuals to publish and manage content. This challenge of today creates the need to answer the key question: What is the most optimal and practical way to publish and manage content? Research into methods and technologies reveals that the two most popular content management options are CMS systems and static site generators. This final paper analyzes and compares these two leading content management strategies, considering their advantages and limitations in detail. To ensure an accurate comparison, two identical portals were developed, whose development provides a deeper insight into situations in which it is optimal to use a CMS system or static page generators.

Keywords: Static Site Generators, CMS systems, Python, Flask, Pelican

8 Prilozi

Popis slika unutar završnog rada:

Slika 1: Prikaz toka izrade web stranice putem generatora statičnih stranica 3
Slika 2 Ilustracija prikaza rada tradicionalnih CMS sustava, Dostupno na: https://www.linkedin.com/pulse/headless-cms-vs-decoupled-explained-5-minutes-sam-saltis/ 11
Slika 3 Ilustracija pojednostavljenih razlika između tradicionalnog, Decoupled i Headless CMS sustava, Dostupno na: https://www.justrelate.com/headless-cms-and-decoupled-cms-explained-52faddc2d9e51b07 12
Slika 4 Ilustracija razlika između Headless i Decoupled CMS sistema, Dostupno na: https://www.linkedin.com/pulse/headless-cms-vs-decoupled-explained-5-minutes-sam-saltis/ 14
Slika 5 TIOBE Programming Community Index, Dostupno na: https://www.tiobe.com/tiobe-index/ 17
Slika 6 GitHut 2.0 Dostupno na: https://madnight.github.io/githut/#/pull_requests/2023/2 ... 17
Slika 7 Primjer početne "Hello World" Flask web aplikacije 18
Slika 8 Konzolni izlaz kod pokretanja Flask web aplikacije 18
Slika 9 Instalacija Pelican-a u Python virtualnom okruženju 19
Slika 10 Inicijalno pokretanje Pelican-a 19
Slika 11 Naredba za generiranje sadržaja 19
Slika 12 Pokretanje Pelican web servera 19
Slika 13 Primjer Jinja koda 20
Slika 14 Rangiranje MySql baze podataka 21
Slika 15 Primjer koda za kreiranje nacrta (eng. *Blueprint*) 22
Slika 16 Funkcija registracije nacrta u aplikaciji 23
Slika 17 Pozivanje funkcije *routes()* u inicijalnoj datoteci koja registrira nacrte 23
Slika 18 Klasa modela *User* 25
Slika 19 Klasa modela *Role* 26
Slika 20 Klasa pomoćne tablice *UserRoles* 26
Slika 21 Klasa modela *Post* 27
Slika 22 Klasa modela *Category* 28
Slika 23 Klasa modela Comment 28

Slika 24 Klasa modela *RequestLog* 29
Slika 25 Klasa modela *ErrorLog* 29
Slika 26 Primjer dohvata iz baze podataka 30
Slika 27 Primjer zapisa u bazu podataka 31
Slika 28 Primjer brisanja iz baze podataka 32
Slika 29 Primjer funkcije za obradu greške s kodom 500 33
Slika 30 Funkcija koja prikazuje odgovarajući ekran korisniku te poziva funkciju za spremanje podataka o grešci u bazu podataka 34
Slika 31 Primjer prikaza poruke na ekranu ukoliko se dogodi greška 35
Slika 32 Definiranje forme za prijavu - *LoginForm* klase 36
Slika 33 Prikaz forme za prijavu - *LoginForm* klasa 36
Slika 34 Definiranje forme za registraciju - *RegisterForm* klasa 37
Slika 35 Prikaz forme za registraciju - *RegisterForm* klasa 37
Slika 36 Definiranje forme za promjenu korisničkih podataka - *UserAccountSettingsForm* i *UserChangePasswordForm* klase 38
Slika 37 Prikaz formi za promjeni korisničkih podataka - *UserAccountSettingsForm* i *UserChangePasswordForm* klase 39
Slika 38 Validator za provjeru zauzetosti korisničkog imena 40
Slika 39 Validator za provjeru zauzetosti email adrese korisnika 41
Slika 40 Validator za provjeru valjanosti lozinke 41
Slika 41 Prikaz unosa krivih podataka u formu koja koristi napravljene validatore 42
Slika 42 Prikaz unosa krivih podataka u formu koja koristi napravljene validatore 42
Slika 43 Primjer korištenja *validate_on_submit()* funkcije koja uz pomoć zadanih validatora provjerava ispravnost unosa 43
Slika 44 Prikaz funkcije za izradu *own_account_required* dekoratera 44
Slika 45 Rute i funkcije za autentifikaciju i autorizaciju putem Google vanjskog servisa 45
Slika 46 Prikaz izgleda korisničkog sučelja 47
Slika 47Prikaz izgleda admin grafičkog sučelja 48
Slika 48 Konzolni prikaz instalacije Pelican-a 49
Slika 49 Konzolni prikaz instalacije markdown biblioteke 49
Slika 50 Konzolni prikaz instalacije svih potrebnih biblioteka za rad s Pelican-om 50
Slika 51 Konzolni prikaz početnog postavljanja pomoću naredbe *pelican-quickstart* 50
Slika 52 Prikaz strukture Pelican projekta 51
Slika 53 Prikaz primjera stranice u markdown formatu 51

Slika 54 Konzolni prikaz generiranja sadržaja te pokretanja lokalnog servera 52
Slika 55 Prikaz generiranog portala s zadanom temom 52
Slika 56 Prikaz mijenjanja Pelican teme 53
Slika 57 Prikaz strukture mape s datotekama za izradu Pelican teme 53
Slika 58 Prikaz strukture mape softyTechTheme teme 54
Slika 59 Prikaz izgleda naslovne stranice generiranog sadržaja 55
Slika 60 Prikaz Javascript funkcije potrebne za dodavanje Disqus komentara 56
Slika 61 Prikaz predloška kontakt forme na statičnoj stranici koristeći FormKeep platformu 56