Usporedba SPA i Web komponenata za razdvajanje frontalnog i pozadinskog razvoja u Adobe Experience Manageru

Hip, Antonio

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Hip

USPOREDBA SPA I WEB KOMPONENATA ZA RAZDVAJANJE FRONTALNOG I POZADINSKOG RAZVOJA U ADOBE EXPERIENCE MANAGERU

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Hip

Matični broj: 0016123262

Studij: Informacijski sustavi

USPOREDBA SPA I WEB KOMPONENATA ZA RAZDVAJENJE FRONTALNOG I POZADINSKOG RAZVOJA U ADOBE EXPERIENCE MANAGERU

DIPLOMSKI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, kolovoz 2023.

Antonio Hip

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U ovom radu cilj je usporediti pristupe za odvajanje frontalnog i pozadinskog razvoja u alatu Adobe Experience Manager. Kod izrade aplikacija u AEM-u obično se nude tri pristupa: klasičan pristup gdje su frontalni i pozadinski dio zajedno, SPA pristup te pristup koristeći web komponente. Za početak objasnit ću web aplikacije te što je zapravo SPA, koje su prednosti i nedostaci. Objasnit ću što je CMS (sustav za upravljanje sadržajem) i prednosti razvoja i korištenja takvog sustava. Nakon uvoda u AEM prikazat ću način izrade aplikacije za svaki navedeni pristup te ih usporediti. Na kraju će biti prikazana implementacija aplikacije F1 App.

Ključne riječi: web aplikacija; SPA; MPA; CMS; AEM; React; web komponente; izrada aplikacije u AEM-u

Sadržaj

1. Uvod

U današnje vrijeme na raspolaganju imamo pregršt različitih web aplikacija i stranica. Stvoren je aktivni moderni web u kojemu i sami korisnici sudjeluju pri stvaranju sadržaja. Cilj svake stranice je da se sadržaj kreira i objavi brzo i jednostavno. Zbog ovog zahtjeva došlo je do pojave sustava koji omogućuje upravljanje sadržajem.

Sustav za upravljanje sadržajem (CMS) je softver koji omogućava korisnicima kreiranje web sadržaja korištenjem grafičkih sučelja. Kako bi korisnik kreirao web sadržaj koristeći CMS ne treba poznavati tehnologije koje se primjenjuju za frontalni razvoj web aplikacije. Jedan od takvih softvera je Adobe Experience Manager.

Kod izrade web aplikacija nastoji se odvojiti frontalni razvoj od pozadinskog razvoja kako bi timovi na projektu radili paralelno i odvojeno, neovisni jedni o drugima. Pri frontalnom razvoju aplikacija sve se više koriste napredni okviri za izradu robusnih komponenata koje se ističu fluidnošću, brzim prijelazima i animacijama. Od nedavno su takvi pristupi podržani i u AEM-u. Prilikom izrade aplikacije u AEM-u važno je odabrati pravi pristup. Najčešće su na raspolaganju standardni ili tradicionalni pristup, SPA pristup te pristup koristeći web komponente.

Ovaj rad može se podijeliti u tri cjeline:

- Teorijska obrada web aplikacija, CMS-a i AEM-a
- Prikaz i primjer implementacije web aplikacije za svaki od spomenutih pristupa
- Usporedba pristupa
- Praktični dio – izrada aplikacije u AEM-u

2. Web aplikacija

Web aplikacija je interaktivni program koji se za razliku od standardnih programa i aplikacija ne izvodi na računalu, nego na web serveru kojem korisnici pristupaju koristeći internetski preglednik. Korisnik koristeći web preglednik šalje zahtjev web poslužitelju koji zatim prosljeđuje zahtjev aplikaciji. Aplikacija je zadužena za izvođenje zadatka i generiranje rezultata koje vraća web poslužitelju, a poslužitelj generirane rezultate prosljeđuje korisniku u web preglednik. U današnje vrijeme web aplikacije su sve više zastupljene i kompleksnije pa mogu izvršavati gotovo sve zadatke kao što su npr. web trgovina, slanje mailova, uređivanje

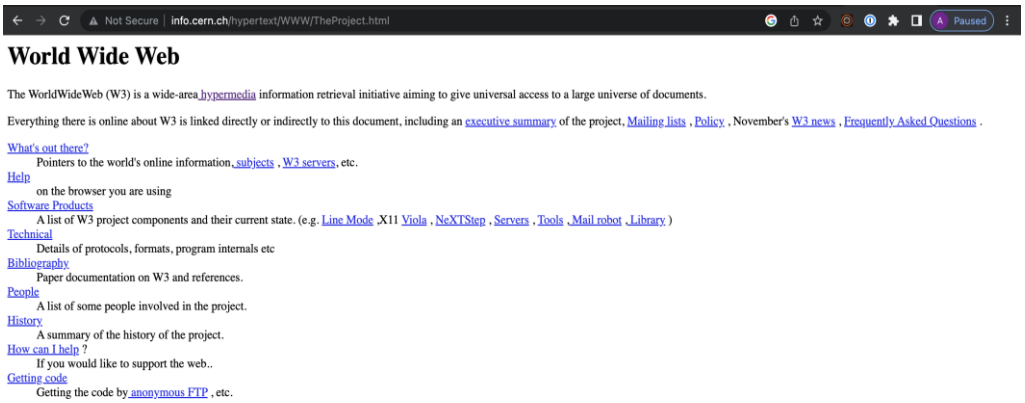
dokumenata, igranje igrica… Najpoznatije web aplikacije su Gmail, Amazon, Facebook, YouTube, Netflix… Neke od prednosti web aplikacija su sljedeće [1]:

- Jednostavno održavanje – web aplikacija ima samo jednu verziju koja je dostupna svima pa nema problema oko kompatibilnosti
- Jednostavno korištenje – za korištenje web aplikacije nije potrebno preuzimanje niti instalacija
- Dostupnost – web aplikacije uvijek su dostupne te im može pristupiti više korisnika u isto vrijeme i na bilo kojoj platformi: Windows, Linux, MacOS, Android, iOs…

Web aplikacije često se poistovjećuju s web stranicama, stoga je potrebno razlikovati ovo dvoje. Web stranica se sastoji od statičkog sadržaja koji je dostupan korisnicima samo za čitanje ili gledanje, dok je web aplikacija namijenjena za interakcije s korisnicima. Kod web aplikacije korisnici mogu čitati i gledati sadržaj, ali također mogu manipulirati tim sadržajem, pretraživati ili zadavati neke zadatke koje će aplikacija odraditi. Web aplikacija najčešće ima i proces autentifikacije kojom se izgled ili ponašanje aplikacije može u potpunosti mijenjati od korisnika do korisnika.

2.1. Povijest web aplikacija

Temelji web aplikacije sežu u 1989. godinu kada je britanski računalni znanstvenik Tim Berners-Lee započeo razvijanje World Wide Weba. Tim Berners-Lee u svom dokumentu „Information Management: A Proposal“ [2] navodi da se pojavilo pitanje kako će izgledati upravljanje sve većim projektima u budućnosti. U svojem radu daje osvrt na svoje iskustvo korištenja nelinearnog tekstualnog sustava poznatog kao „hypertekst“. Radi se o tekstualnoj strukturi koja se sastoji od međusobno povezanih jedinica informacije, tako da tekst sadrži poveznicu na drugi sadržaj. U zaključku Tim smatra da je jedini način razviti sustav povezanih informacija koji će omogućiti korisnicima pretraživanje i objavljivanje informacija neovisno o računalnim platformama. Smatra da takav sustav treba imati dvije odvojene cjeline. Prva cjelina će biti sustav za pohranu podataka, a druga cjelina sustav za prikaz podataka. Te dvije cjeline treba povezati dobro definiranim sučeljem koje će fizički odvajati korisnika i bazu podataka. Barners-Lee napravio je prvi web server, preglednik (WorldWideWeb), HTTP protokol (eng. Hypertext Transfer Protocol) te jezik za označavanje teksta nazvan HTML (eng. Hypertext Markup Language). Zanimljivo je da se na stranicama CERN-a može pristupiti prvoj web stranici, a kako ona izgleda prikazano je na sljedećoj slici.



Slika 1: Prva web stranica (preuzeto sa [3])

Prva web stranica sastavljena je od samog teksta i hypertexta, tj. poveznica koje vode na druge sadržaje.

U 1991. godini WorldWideWeb je postao javno dostupan pa je sve više organizacija težilo ostvarenju vlastitih web stranica čiji je broj eksponencijalno rastao. U tom razdoblju nastale su i prve web stranice za trgovinu poput Amazona i EBaya, kao i internetske tražilice Yahoo i AltaVista. Opisano razdoblje trajalo je do 2004. godine, nastalo je 250.000 stranica, koristilo ga je 45 milijuna korisnika, a naziva se Web 1.0.

Web 2.0 postao je poznat 2004. godine kada su Tim O'Reilly i Dale Dougherty održali konferenciju u San Franciscu na kojoj se vodila rasprava o World Wide Webu [4]. Novost kod Web 2.0 je da korisnici osim pregledavanja sadržaja mogu također i sudjelovati u procesu njegove izrade pa je ovaj web poznat kao „read-write web“. Revolucija u ovom razdoblju bila je korištenje AJAX (Asynchronus Javascript and XML) i JavaScript tehnologije. Time su web stranice postale dinamičnije i brže, a korisnici su mogli uživati u funkcionalnostima kao što su sortiranje, pretraživanje, pregledavanje slika i videa, komentiranje i sl. U ovom razdoblju došlo je do razvoja društvenih mreža poput Twittera, Facebooka, MySpacea na kojima su korisnici mogli izmjenjivati sadržaje i poruke. Pojavile su se i aplikacije za razmjenu video sadržaja poput YouTubea. U ovom razdoblju broj web stranica ili aplikacija skočio je na 80.000.000 koje je koristilo više od milijardu korisnika [5].

Brzi razvoj Weba 2.0 opisuje sam Tim O'Reilly u svojoj knjizi „Web Squared: Web 2.0 Five Years On“ [6] u kojoj navodi da je veliki razvoj weba krenuo 2007. godine kada su se pojavili mobilni uređaji. Dolaskom Weba na mobilne uređaje razvio se prilagodljiv, responzivni (eng. Responsive) web. Web stranice i aplikacije sada su bile prilagođene gotovo svakoj veličini zaslona.

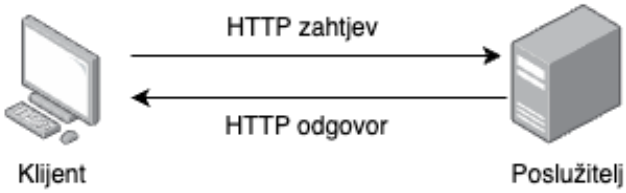
Sve ove tehnologije bile su zaslužne za razvijanje Weba 3.0, nazvanog još i „Semantički Web“, a opisan je kao Web za čitanje, pisanje i izvršavanje. Javile su se dinamičke i interaktivne aplikacije koje su međusobno komunicirale. Javlja se i izraz „Internet of Things“ što označava da gotovo svaki uređaj dobiva senzore kojima prikuplja informacije i razmjenjuje ih s drugim uređajima pa se tako javljaju i pametne kuće koje mogu same otvarati prozore, paliti i gasiti svjetla i slično. Dolazi do razvoja umjetne inteligencije i strojnog učenja što uvelike pomaže kod obrade velike količine podataka. I sami smo svjedoci nevjerojatne tehnologije te korištenjem interneta svaki dan svjedočimo umjetnoj inteligenciji kada pričamo o nekom proizvodu, a za par sati na pametnim uređajima dobivamo oglase o spomenutom proizvodu. Tako možemo reći da internet i web aplikacije postaju čovjekov osobni asistent i u potpunosti se prilagođavaju svakoj osobi.

2.2. Arhitektura web aplikacija

Arhitektura web aplikacije opisuje komponente web aplikacije te kako one međusobno komuniciraju. Već je spomenuto da je Tim Berners-Lee još kod traženja rješenja za web zaključio da je jedino rješenje razviti sustav s dvije cjeline. Prva cjelina će obrađivati i pohranjivati podatke, a druga će te podatke prikazivati korisnicima. Tu se već pojavila inicijalna klijent-poslužitelj struktura web aplikacije koja se nije mijenjala.

- Klijent ili korisnički (frontalni) dio aplikacije - cjelina na kojoj korisnik vidi web aplikaciju te upravlja njome. Tehnologije, tj. jezici koji se koriste za razvoj korisničkog dijela aplikacije su HTML, CSS i JavaScript.
- Poslužitelj ili stražnji (pozadinski) dio aplikacije je cjelina koja sadrži poslovnu logiku i podatke. Zadužen je za obradu zahtjeva koje prima od klijenta. Tehnologije, tj. jezici koji se koriste za razvoj poslužitelja su Java, C#, Python, PHP, Ruby…

Kako navodi Casimir Saternos [7], osnovna ideja weba se nije promijenila. Klijent šalje HTTP zahtjeve poslužitelju koji taj zahtjev obrađuje, a onda klijentu vraća rezultat koji se prikazuje krajnjem korisniku. Komunikacija klijenta i poslužitelja prikazana je na sljedećoj slici.



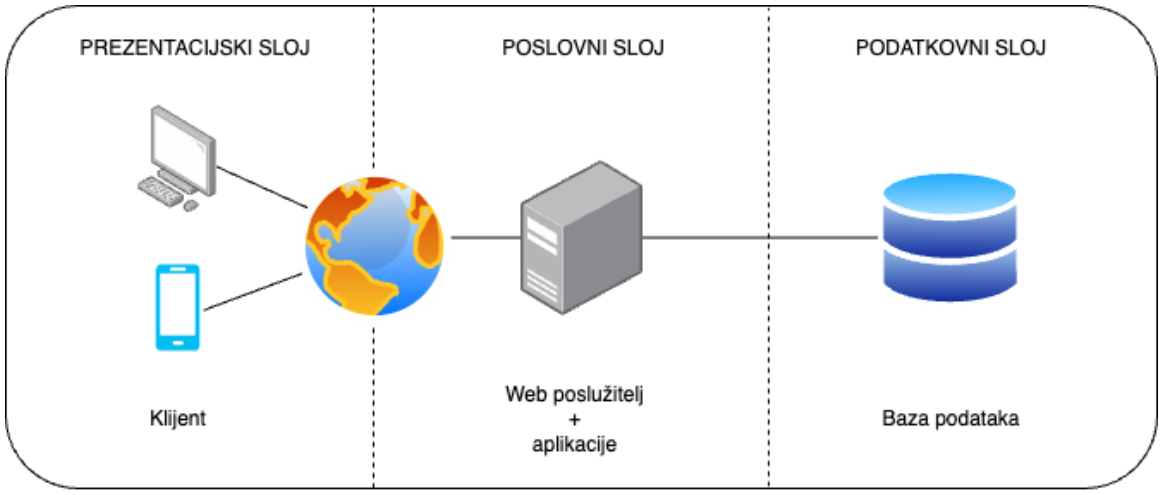
Slika 2. Komunikacija klijenta i servera (prema Casimiru Saternosu [7])

Moderne web aplikacije građene su slojevito. Prednost takve arhitekture je da svaki sloj obavlja i odgovoran je za svoje zadatke, ne postoji ovisnost između slojeva, a komunikacija je definirana tako da svaki sloj može komunicirati samo sa svojim susjednim slojem. Kada promatramo građu aplikacije po slojevima, dizajn se naziva po broju slojeva pa tako imamo jednoslojne, dvoslojne, troslojne, četveroslojne… aplikacije.

Jednoslojne aplikacije objedinjuju klijent, poslužitelj i bazu podataka u jedan sloj. Takve aplikacije najčešće se izrađuju za testiranje okoline ili vježbanje određenog programskog jezika. Takve web aplikacije gotovo ne postoje u produkciji, ali ako se radi o maloj web aplikaciji namijenjenoj za malu grupu korisnika moguća je i stvarna implementacija.

Dvoslojna aplikacija sastoji se od dva sloja. Obično je prvi sloj prezentacijski sloj, a drugi sadrži poslovnu logiku i bazu podataka. Možemo zaključiti da je ova građa aplikacije prvotno zamišljena kod pojave weba gdje je ideja bila da se aplikacija sastoji od klijenta i poslužitelja s dobro definiranim sučeljem između njih.

Troslojna aplikacija sastoji se od tri sloja, a to su prezentacijski sloj, poslovni sloj te podatkovni sloj. Većina modernih aplikacija građena je upravo troslojno. Građu troslojne aplikacije možemo vidjeti na sljedećoj slici.



Slika 3. Troslojna arhitektura web aplikacije (prema [8])

Troslojna arhitektura sastoji se od [7]:

- Prezentacijski sloj – nalazi se na strani korisnika i pruža korisničko sučelje za interakciju korisnika i same web aplikacije. Ovaj sloj komunicira s web poslužiteljem kako bi dobio rezultate korisnikovog zahtjeva te ih prikazao u web pregledniku.
- Poslovni sloj – nalazi se na strani poslužitelja i srce je svake troslojne web aplikacije. Sadrži svu poslovnu logiku aplikacije te je zadužen za obradu svakog korisničkog

zahtjeva te isporuku rezultata prezentacijskom sloju. To uključuje procesiranje zahtjeva, manipulaciju podacima i izvršavanje poslovnih pravila.

- Podatkovni sloj – nalazi se na strani poslužitelja i zadužen je za upravljanje podacima. Odgovoran je za pohranu podataka te opskrbu poslovnoj sloja podacima. Sadrži metode i opise koje poslovni sloj mora pozvati kako bi dobio potrebne podatke.

Prednosti troslojne arhitekture su sljedeće [7]:

- Daje mogućnost nadogradnje ili ažuriranja jednog sloja bez utjecaja na drugi sloj.
- Omogućuje članovima razvojnog tima da se podijele po slojevima i steknu specifične kompetencije za određeni sloj. Tako pojedinac može biti ekspert za razvoj frontalnog dijela aplikacije bez potrebe razumijevanja tehnologija za razvoj pozadinskog dijela aplikacije. Ovakav pristup također omogućuje brži razvoj aplikacije.
- Odvajanjem baze podataka u posebni sloj povećana je razina sigurnosti jer korisnici nemaju direktan pristup podacima.
- Odvajanjem aplikacije u tri cjeline lakše je identificirati i riješiti problem.

2.3. Vrste web aplikacija

S obzirom na logiku aplikacije, komunikaciju klijenta i poslužitelja, način rada aplikacije, njezine funkcionalnosti i najbolje prakse prilikom izrade, postoji više vrsta aplikacija. Tako imamo web aplikacije na više stranica (MPA), web aplikacije na jednoj stranici (SPA), progresivne web aplikacije (PWA), mikroservise…

Za ovaj rad bitno je objasniti aplikacije na više stranica i aplikacije na jednoj stranici.

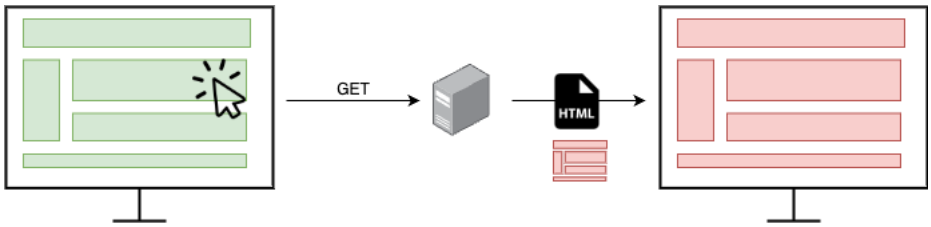
2.3.1. Web aplikacija na više stranica (MPA)

Web aplikacije na više stranica su tradicionalne web aplikacije koje se sastoje od više stranica popunjenih statičkim sadržajem kao što su tekst, slike, a svaka stranica sadrži poveznice koje vode na ostale stranice aplikacije. Iako su neke komponente stranice zajedničke svakoj stranici, npr. navigacija, logo, podnožje itd., svakim klikom na poveznicu preuzima se cijeli sadržaj tražene stranice.

Slika 4. Životni ciklus MPA aplikacije (prema [9])

Na slici 4. prikazan je životni ciklus MPA aplikacije. Prilikom dolaska na web stranicu korisnik šalje inicijalni GET zahtjev poslužitelju koji vraća HTML sa svim sadržajem koji se nalazi na traženoj stranici. Ako korisnik klikne na neku poveznicu, tj. želi pristupiti drugoj stranici ili osvježiti podatke pojedine komponente, šalje sljedeći GET ili POST zahtjev. Poslužitelj obrađuje zahtjev te ponovo vraća HTML sa svim sadržajem koji se nalazi na traženoj stranici.

Kao što je ranije navedeno, ako korisnik pošalje zahtjev čiji rezultat treba biti promjena samo jedne komponente koja se nalazi na stranici, poslužitelj će ponovo vratiti cijelu HTML datoteku sa svim sadržajem kao što je prikazano na slici 5. Takvim pristupom dolazi do nepotrebnog učitavanja nepromijenjenog sadržaja čime se uvelike produljuje vrijeme odaziva web stranice.



Slika 5. MPA učitavanje promjene na stranici (autorski rad)

Solovei, Olshevska i Bortsova [10] navode da su prednosti MPA web aplikacije sljedeće:

- Optimizirane su za web pretraživanje – s obzirom da se web aplikacija sastoji od više stranica koje imaju svoj naziv, veća je vjerojatnost da će se aplikacija pojaviti u web tražilici za određeni pojam
- Mogućnost kreiranja novog sadržaja na novim stranicama
- Brz i jednostavan razvoj zbog postojanja velikog broja primjera
- Dostupnost bez JavaScripta

Nedostaci MPA web aplikacija su sljedeći:

- Loše performanse – kao što je već spomenuto, za veliki broj korisničkih zahtjeva potrebno je svaki puta učitavati cijelu stranicu što produljuje vrijeme odaziva
- Teže održavanje – razvojni inženjeri trebaju održavati svaku stranicu posebno

MPA aplikacije svoju primjenu najčešće pronalaze kod web aplikacija koje sadrže ogromnu količinu sadržaja, veliki broj stranica, različite konfiguracije po stranici, opširnu navigaciju što pronalazimo u web aplikacijama kao što su e-trgovine, blogovi, forumi, a najbolji primjer za izdvojiti su Amazon i eBay.

2.3.2.Web aplikacija na jednoj stranici (SPA)

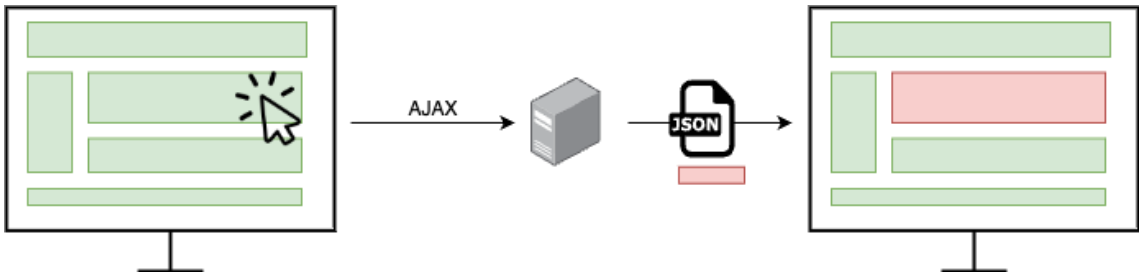
Prema Fernandu Monteriu [11] SPA (eng. Single Page Application) je web aplikacija koja se izvodi na jednoj stranici, a cilj joj je pružanje fluidnog korisničkog iskustva i bogato sučelje. Kod takve web aplikacije inicijalno se učita cijela stranica sa svim sadržajem, a svakim korisničkim zahtjevom poslužitelj pregledniku šalje samo dio aplikacije koji je potrebno promijeniti bez potrebe za učitavanjem čitave stranice.

Emmit A. Scott [12] piše da je ideja za razvoj ovakvog tipa došla od želje razvojnih inženjera da naprave web aplikaciju što više sličnu desktop ili mobilnoj aplikaciji jer su imali iskustva sa razvojem takvih aplikacija. Nakon mnogo pokušaja i različitih tehnologija poput IFramea, Java appleta, Adobe Flash-a, došli su do rješenja oko 2000. godine kada se pojavio AJAX. Kombinacijom AJAX-a i JavaScripta dobila se mogućnost dinamičke promjene web stranice bez potrebe za osvježavanjem cijelog sadržaja.

Slika 6. Životni ciklus SPA web aplikacije (prema [7])

Na slici 6 prikazan je životni ciklus SPA web aplikacije. Prilikom dolaska na web aplikaciju, korisnik poslužitelju šalje inicijalni GET zahtjev koji poslužitelj obradi te vraća sav sadržaj (HTML, CSS, JS). Ako korisnik klikne na neku poveznicu, tj. želi pristupiti drugoj stranici ili

osvježiti podatke pojedine komponente šalje se AJAX poziv na koji poslužitelj odgovara slanjem JSON datoteke s podacima koje je potrebno promijeniti. Možemo primijetiti da se kod ovog pristupa cijela stranica učitava samo kod inicijalnog zahtjeva, a svakim sljedećim zahtjevom za promjenom mijenja se samo sadržaj ili komponenta koju je potrebno promijeniti kao što je prikazano na slici 7. Takvim pristupom uvelike se skraćuje vrijeme odgovora što je ključno za današnje aplikacije.



Slika 7. SPA učitavanje promjene na stranici (autorski rad)

Neke od prednosti SPA web aplikacija su sljedeće [8]:

- Brzina odgovora – web aplikacija se prvi puta učitava sa svim sadržajima, a sljedeći zahtjevi rezultiraju brzim odgovorima koji sadrže samo potrebne podatke
- Bogato korisničko sučelje
- Jednostavna tranzicija na mobilnu aplikaciju – SPA web aplikacije vrlo su slične mobilnim aplikacijama stoga je moguće iskoristiti kod prilikom implementacije mobilne aplikacije
- Jednostavnije ispravljanje grešaka – u pregledniku postoji mogućnost praćenja Internet zahtjeva te sami preglednici imaju mogućnost ispravljanja pogrešaka

Nedostaci SPA web aplikacija su sljedeći [8]:

- Loša optimizacija za pretraživanje – ne postoji više stranica s opisima sadržaja koje aplikacija nudi pa tražilice nemaju sve informacije o funkcionalnostima aplikacije
- Funkcionalnost tipke „nazad“ – „nazad“ u pregledniku neće korisnika vratiti na prethodno stanje aplikacije nego na prethodnu stranicu
- Sigurnost – SPA web aplikacije izložene su XSS (eng. cross-site scripting) napadima

3. Sustav za upravljanje sadržajem

Prema definiciji Deanea Barkera [13] sustav za upravljanje sadržajem (eng. Content Management System – CMS) je softver koji pruža automatizam za zadatke koji su nužni kako bi učinkovito upravljali sadržajem. Prema tome CMS pruža uređivačima da kreiraju novi sadržaj, uređuju postojeći sadržaj, nadograđuju sadržaj, publiciraju sadržaj i sl. Da bi razumjeli definiciju, prvo treba objasniti sadržaj. Sadržaj je svaki dio informacije koji postoji u aplikaciji kao npr. tekst, slika, crtež, video, datoteka, grafikon. Gotovo sve što je vidljivo u web aplikaciji smatra se sadržajem.

3.1. Uvod u CMS

Kao što je spomenuto, pojavom Weba 2.0 osim pregledavanja sadržaja korisnici su počeli aktivno sudjelovati u kreiranju, tj. stvaranju sadržaja. Time se javlja potreba za razvojem sustava koji će omogućiti što lakše kreiranje i upravljanje sadržajem.

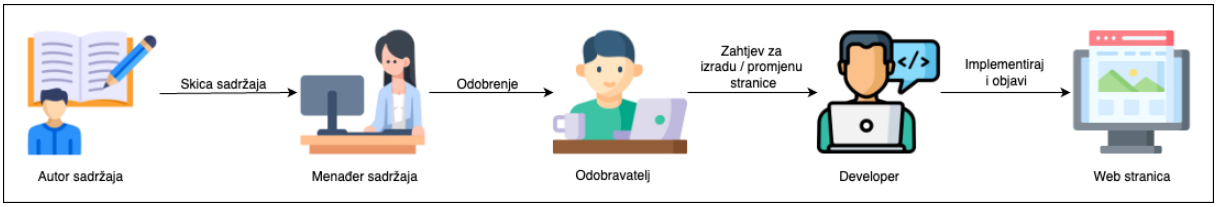
CMS je zapravo softver koji omogućava korisnicima da kreiraju, uređuju i objavljuju sadržaj korištenjem gotovih jednostavnih sučelja. Tako korisnici upravljaju web stranicom bez potrebe za poznavanjem programskih jezika i ostalih tehnologija koje se koriste za izradu web aplikacija. Glavna osobina CMS sustava je nazvana WYSIWYG (eng. *What You See Is What You Get*) što znači da prilikom uređivanja sadržaja korisnik vidi sadržaj na isti način na koji će taj sadržaj izgledati i nakon objave.

Jednostavan opis CMS sustava daje Gaurav Kathuria [14] koji za primjer uzima web stranicu dnevnih novina. Može se primijetiti da je struktura svakog broja novina jednaka. Uvijek se prvo javlja naslov pa opis, slika i tekst članka. Ostale komponente kao horoskop, vremenska prognoza ili vicevi su također strukturom bez promjena, jedino se mijenja sadržaj tih komponenata. S obzirom da se novine mijenjaju svaki dan, tj. objavljuju se novi članci, tako se treba mijenjati HTML stranice što bi svakodnevno zahtijevalo prisutnost razvojnog inženjera koji bi izrađivao stranice za novine. Uređivanje stotina takvih stranica na dnevnoj bazi oduzelo bi im puno vremena za tako jednostavne zadatke. Tako bi svaka organizacija koja posjeduje web stranicu trebala podršku web razvojnog inženjera kada bi htjeli promijeniti sadržaj što bi u današnje vrijeme bio veliki problem jer je najvažnije objaviti sadržaj što prije. Zbog opisanog javlja se potreba za CMS sustavom. Neke od potreba koje CMS treba zadovoljiti su sljedeće:

- Smanjena potreba za IT stručnjacima ili razvojnim inženjerima prilikom uređivanja i održavanja web stranice
- Smanjeni troškovi održavanja web stranice

- Pojedinci bez tehničkog znanja o programiranju trebali bi sami održavati vlastite web stranice
- Brza isporuka sadržaja bez nepotrebnog kašnjenja
- Automatizacija prilikom kreiranja, odobravanja te objave sadržaja

Na sljedećoj slici prikazan je način upravljanja sadržajem bez korištenja CMS-a.

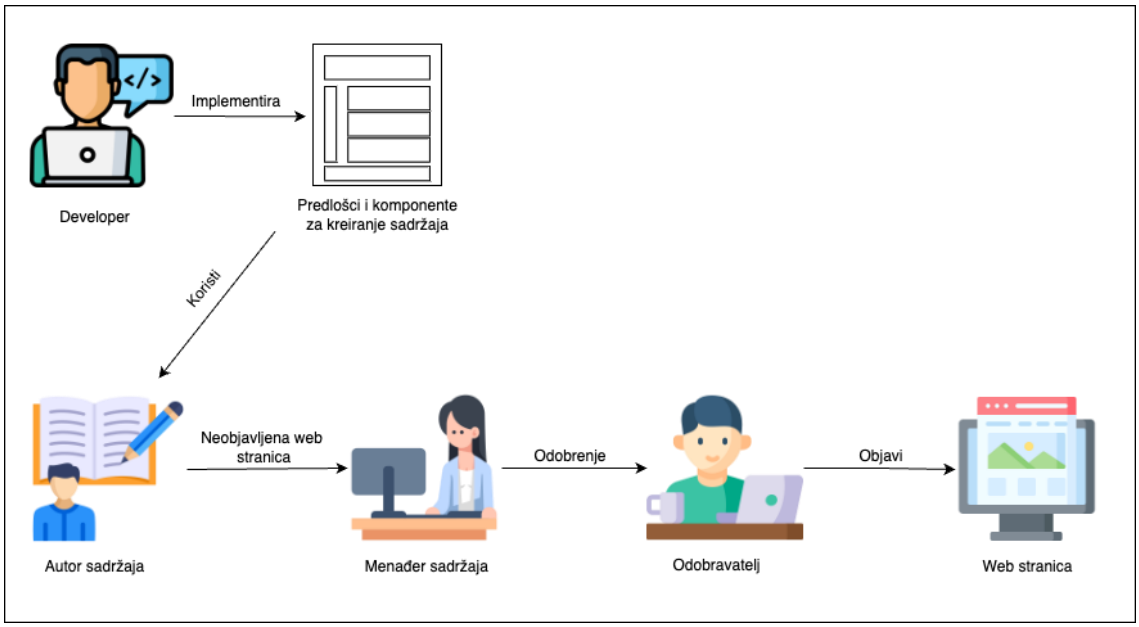


Slika 8. Upravljanje sadržajem bez korištenja CMS-a (prema [14])

Proces započinje autor sadržaja koji prikuplja novosti te osmišljava izgled i sadržaj web stranice. Izrađenu skicu sadržaja šalje menadžeru koji treba provjeriti sadržaj te ukoliko sadržaj zadovoljava sve potrebne kriterije šalje skicu odobravatelju. Odobravatelj provjerava sadržaj te ukoliko smatra da je sadržaj spreman za objavu, šalje zahtjev razvojnom inženjeru da izradi web stranicu. Nakon što implementira zahtijevanu web stranicu, sadržaj je spreman za objavu. Ovaj proces troši puno vremena te sadržaj više nije aktualan kada se prođu svi koraci.

Korištenjem CMS-a korisnici dobivaju gotove predloške komponenata kao što su tekst, slika, navigacija te predložak web stranice na kojima se već nalaze spomenute komponente. Za kreiranje sadržaja korisnici koriste jednostavna sučelja u kojima popunjavaju informacije te se HTML automatski generira na temelju korisničkog unosa. Na taj način svaki sudionik u ranije spomenutom procesu objave može vidjeti kako će taj sadržaj izgledati nakon objave. Ovaj pristup zahtijeva od razvojnog inženjera da prikupi informacije o zahtjevima organizacije kako bi kreirao spomenute predloške i komponente. To je jednokratan proces i nakon implementacije i postavljanja CMS-a njegova pristunost više nije potrebna jer korisnici sami mogu upravljati svojom stranicom.

Na slici 9. prikazano je upravljanje sadržajem koristeći CMS.



Slika 9. Upravljanje sadržajem koristeći CMS (prema [14])

3.2. Vrste CMS-a

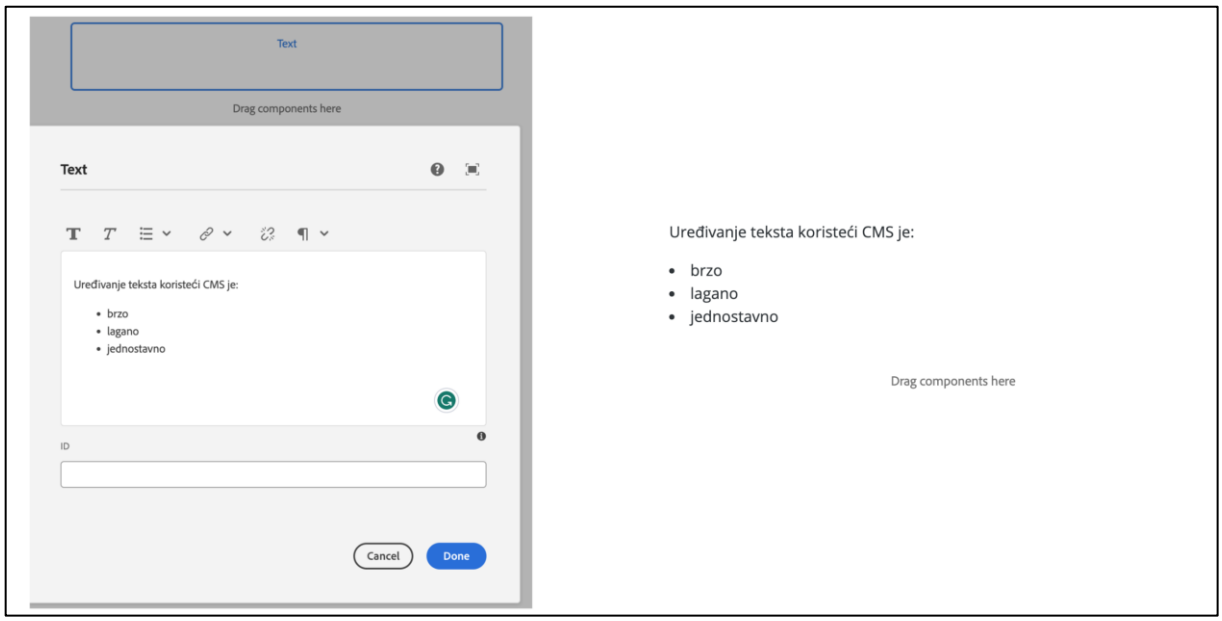
Prema Barkeru [13] CMS se može podijeliti na sljedeće vrste:

- Web upravljanje sadržajem (WCM – *Web Content Management*) – sustav za upravljanje sadržajem prvenstveno namijenjen masovnoj isporuci putem web stranice. WCM se ističe u odvajanju sadržaja od prezentacije te objavljivanjem sadržaja na više kanala.
- Poslovno upravljanje sadržajem (ECM – *Enterprise Content Management*) – upravljanje općenitim poslovnim sadržajem koji je objedinjen u organizaciji (životopisi zaposlenika, informacije o plaći, bilješke o zaposlenicima). Ističe ga kontrola pristupa i upravljanje sadržajem po ulogama.
- Upravljanje digitalnom imovinom (DAM – *Digital Asset Management*) – upravljanje i manipulacija digitalnom imovinom kao što su slike, zvuk i video
- Komponentni sustav za upravljanje sadržajem (CCMS – *Component content management system*) – kreiranjem i korištenjem komponenata stvara se sadržaj web stranica. Npr. naslov, podnaslov, slika, tekst, slika s tekstom…
- Sustav za upravljanje sadržajem za učenje (LCMS – *Learning Content Management System*) – koriste ga škole i fakulteti za kreiranje sadržaja potrebnog za učenje nekog predmeta ili kolegija.

3.3. Tehnologije i način rada

Kako bi CMS mogao kreirati sadržaj web stranice na temelju korisničkih zahtjeva koje korisnik postavlja u sučeljima koriste se sljedeće tehnologije:

- Programski jezici za implementaciju pozadinskog dijela aplikacije – koriste se za implementaciju modela, servisa, servleta… Najčešće korišteni jezici su Java, PHP, Python..
- HTML – standardni jezik za označavanje koji se koristi za kreiranje web stranica i aplikacija. HTML koristi oznake kojima se stvaraju elementi dokumenta. Te oznake govore pregledniku na koji način prikazati određeni sadržaj koji se nalazi unutar oznake.
- CSS – (*Cascading Style Sheet*) – jednostavan mehanizam koji opisuje kako će HTML elementi biti prikazani na zaslonu. Dodavanjem stilova moguće je mijenjati boju, poziciju, raspored, veličinu i ostale vizualne karakteristike HTML elementa.
- JavaScript – jednostavan programski jezik namijenjen razvoju interaktivnih stranica. Odvija se na strani klijenta, a zaslužan je za dinamičku manipulaciju HTML elementima bez osvježavanja web stranica.



Slika 10. Uređivanje i dodavanje teksta koristeći CMS (Autorski rad)

Kao što je vidljivo na slici, na web stranici postoji predložak koji sadrži mjesto gdje možemo dodati komponente. Dodavanjem *Text* komponente možemo otvoriti dijalog. U

dijalogu nam se nudi uređivač teksta koji izgledom podsjeća na *Microsoft Word* pa je uređivanje teksta intuitivno, jednostavno i brzo. Da bi ovaj tekst napisali i prikazali na stranici bez korištenja CMS-a trebali bi poznavati HTML jezik kao što je prikazano na slici 11.

Slika 11. Uređivanje i dodavanje teksta bez korištenja CMS-a (Autorski rad)

Implementacija CMS-a svodi se upravo na kreiranje takvih predložaka i komponenata koje će zadovoljiti sve korisničke zahtjeve kako bi korisnici sami mogli stvarati bogati sadržaj za vlastite web stranice.

3.4. Prednosti CMS-a

Već je spomenuto da je glavna ideja CMS-a bila omogućiti pojedincima stvaranje web stranica bez poznavanja naprednih web tehnologija. Još neke prednosti korištenja CMS-a su sljedeće:

- Prilagođen korisnicima – korisnik ne treba znati kako CMS funkcionira. Jednostavnim navigiranjem kroz sučelja dolazi do akcija i operacija koje su mu potrebne za stvaranje i uređivanje sadržaja.
- Opcije prilagodbe – CMS pruža široki spektar prilagodba kao što su teme, dizajn, izgled stranice. Svaki CMS može se također proširiti dodacima kao što su web trgovine, email servisi, forme. CMS je fleksibilan pa može zadovoljiti potrebu korisnika koji objavljuju minimalne količine sadržaja, ali i one koji svakodnevno objavljuju članke, slike, videozapise, dokumente i slično.
- Olakšava suradnju – postojanjem uloga u kreiranju sadržaja doprinosi se brzini kreiranja i objavljivanja istog. Npr. za neki članak dizajner će dodati i urediti

sliku, dok će autor napisati tekst. Ove radnje mogu se izvoditi paralelno. Na ovaj način cijeli tim može sudjelovati u izradi sadržaja istovremeno.

- Brze promjene – ukoliko je potrebno brzo promijeniti ili objaviti članak ne mora se čekati razvojni inženjer već autor sam može promijeniti sadržaj i jednim klikom ga objaviti.
- Automatska objava u određeno vrijeme – u današnje vrijeme svjedočimo raznim tzv. influencerima koji svakodnevno objavljuju web sadržaje. „Pauziranjem“ objava na nekoliko dana izgubili bi vjerne pratitelje. CMS u ovom slučaju omogućava automatsko objavljivanje u postavljeno vrijeme. Na taj način sadržaj je moguće kreirati unaprijed, tj. moguće je danas izraditi web stranicu te konfigurirati da se ista automatski objavi sutra u 20 sati. Ovom funkcionalnošću moguće je pripremiti sadržaj za sljedeći dan, tjedan ili čak mjesec.
- Prilagođeno za tražilice – SEO (eng. Search Engine Optimization) je skup procesa kojima se nastoji podići rang stranice za tražilicu. CMS nudi alate i strategije kako bi povisili rang za SEO.
- Pristupačnost – sve više CMS-a prelazi na oblak (eng. Cloud) tehnologiju. Samim time sustav postaje dostupan bilo kad i bilo gdje.
- Prilagođeno mobilnim uređajima – u današnje vrijeme stranice se više posjećuju koristeći mobilni uređaj nego računalo ili laptop. Postojanjem opcija za respnozivni dizajn CMS izrađuje web stranice koje su prilagođene i manjim zaslonima poput mobitela i tableta.
- Jeftinije održavanje – korištenjem CMS-a nije potrebno svakodnevno plaćati razvojne inženjere. Također, poznata je izreka „vrijeme je novac“. Svakim kašnjenjem objave najnovijeg sadržaja smanjuje se potencijalna zarada.

3.5. AEM i ostali CMS

Najpoznatiji sustavi za upravljanje sadržajem su WordPress, Joomla, Drupal, Wix, AEM… WordPress je apsolutni pobjednik u ovoj kategoriji. Prema Colorlibovom istraživanju 810 milijuna web stranica kreirano je koristeći WordPress što je gotovo 43% svih web stranica koje postoje. Tržišni udio WordPressa iznosi 63.1% od globalnog udjela CMS-a [15].

Nakon ovog podatka javlja se pitanje zašto odabrati AEM koji se između ostalog plaća, dok je WordPress besplatan ali se naplaćuju različite komponente, dizajn, teme… Prije usporedbe AEM-a s ostalim CMS sustavima valja istaknuti najveće prednosti AEM-a:

- Upravljanje digitalnim sadržajem – AEM nudi spremanje i upravljanje velikom količinom digitalnog sadržaja te time osvaja korisnike. Moguće je spremiti milijarde fotografija, videa i dokumenata kojima se neovisno o količini vrlo brzo pristupa.
- Integracija na oblaku – integracijom AEM-a na oblak moguće je povezati digitalni sadržaj i marketing te tako dostavljati sadržaj ciljanim skupinama korisnika.
- Brzo pretraživanje – AEM nudi hijerarhijsku bazu podataka koja podsjeća na datotečni sustav.
- Pretvorba slika i videa – postoje procesi koji se pokreću prilikom postavljanja slika, a zadaća im je generirati različite veličine (rendicije) slike i videa. Prednost ovoga je brzina učitavanja stranice. Ukoliko se stranici pristupa s mobilnog uređaja, nije potrebno prikazati sliku velikih dimenzija, nego se učitava slika koja je primjerena za veličinu uređaja s kojeg se pristupa.
- Kompatibilnost s drugim Adobe proizvodima – Adobe nudi različite proizvode poput Commerce i Analitics. Povezivanje s drugim proizvodima veliki je benefit.

Kriterij	Drupal	WordPress	AEM
Jednostavnost upravljanja sadržajem	• Umjereno jednostavno • Zahtjeva tehničko znanje	• Jednostavno • Zahtjeva poznavanje tehnika za razvoj prednjeg dijela aplikacije	• Vrlo jednostavno
Prilagođavanje tema i dizajna	• Umjereno • Sadržajem se upravlja preko CMS, ali za promjenu dizajna zahtjeva akcije razvojnog inženjera	• Jednostavno • Predefinirane teme i dizajni	• Jednostavno • „Povuci i spusti“ • Predefinirani dizajn
Upravljanje dokumentima	• Dobro • Ugrađen menadžer za dokumente	• Dobro • Ugrađen menadžer za dokumente	• Odlično • Pruža napredni servis za upravljanje dokumentima

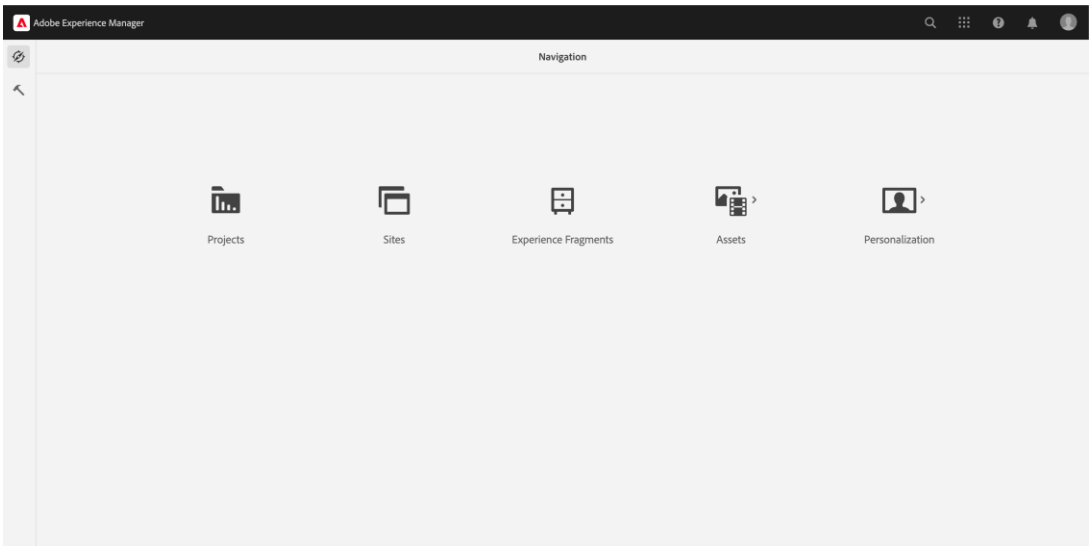
Upravljanje pristupom	• Dobro na aplikacijskoj razini • Zahtjeva tehničko znanje za naprednim postavkama	• Umjereno • Zahtjeva tehničko znanje za naprednim postavkama	• Jako dobro • Sigurnost definirana arhitekturom
Slobodna radna mjesta (Indija)	• 9.000	• 7.500	• 31.000
Cijena licence	• GNU GPLv2 – besplatna	• GNU GPLv2 – besplatna	• Od 100.000$ pa nadalje
Cijena poslužitelja	• 60$ - 500$ mjesečno	• 60$ - 500$ mjesečno	• Vlastiti poslužitelj

Tablica 1. Usporedba Drupal – WordPress – AEM (prema [16])

4. Adobe Experience Manager

Adobe Experience Manager (AEM) je sustav za upravljanje sadržaja koji omogućuje korisnicima stvaranje, uređivanje, vođenje i objavljivanje web sadržaja [17]. Neke od glavnih značajki i prednosti korištenja AEM-a su sljedeće:

- Kreiranje i uređivanje sadržaja – AEM pruža jednostavno korisničko sučelje koje omogućuje autorima stvaranje i uređivanje sadržaja bez potrebe za tehničkim znanjem. Funkcionalnosti poput *„povuci i ispusti“*, uređivanje na mjestu, dijalozi za kreiranje sadržaja omogućuju korisnicima stvaranje sadržaja metodom „WYSIWYG“ (što vidiš, to i dobiješ) što označava da je krajnji rezultat upravo ono što korisnik vidi kada uređuje sadržaj.
- Upravljanje digitalnom imovinom – AEM također nudi korisničko sučelje za upravljanje digitalnom imovinom, tj. slikama, videima, dokumentima i slično.
- Komponente – web stranice stvaraju se korištenjem komponenti. Prednost postojanosti komponenti je da se iste mogu ponovo koristiti što ubrzava proces stvaranja sadržaja.



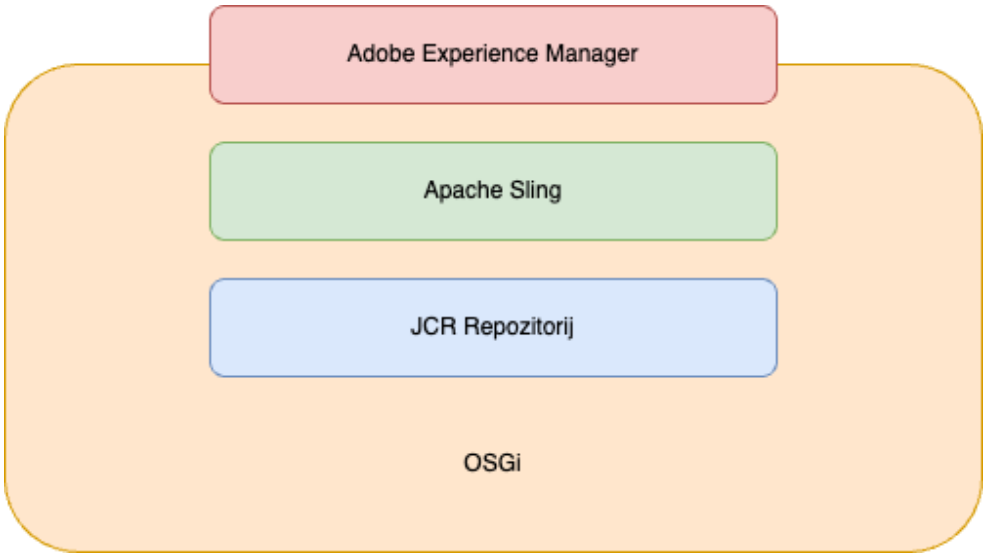
Slika 12. Početna stranica AEM-a (autorski rad)

Početna stranica zapravo predstavlja navigaciju na kojoj možemo primijetiti dva glavna modula, a to su stranice (eng. Sites) koji služi za kreiranje, uređivanje i objavljivanje stranica te digitalna imovina (eng. Assets) kod koje se spremaju, uređuju i objavljuju digitalni sadržaji.

4.1. Arhitektura AEM-a

AEM je web bazirani klijent – poslužitelj sustav izgrađen od nekoliko infrastrukturnih razina i aplikacijskih funkcija. Na prvi pogled kod AEM-a može se primijetiti troslojna arhitektura. Prezentacijski sloj čine brojna intuitivna sučelja koja korisnicima olakšavaju rad. AEM nudi dvije vrste sučelja. To su Touch-enabled UI koji je dizajnirao Adobe kako bi pružio konzistentnost sučelja različitih proizvoda te Classic UI koji nudi tradicionalno iskustvo korištenja sučelja. Podatkovni sloj čine OSGi (eng. Open Services Gateway Initiative), Apache Sling te Apache Felix, dok podatkovni sloj čini JCR (eng. Java Content Repository).

Prema Adobeu [18] arhitektura se može pogledati i s najviše razine. Najvažnija arhitekturna komponenta je OSGi koji nudi glavne funkcije za modularni razvoj servisa, modela te komponenti. Sljedeći je JCR, repozitorij u kojem se spremaju podaci. Slijedi razvojni okvir baziran na REST tehnologijama, tj. Apache Sling. Zadnja komponenta je AEM koji autorima nudi različite funkcionalnosti za upravljanje stranicama, digitalnom imovinom te obrascima.



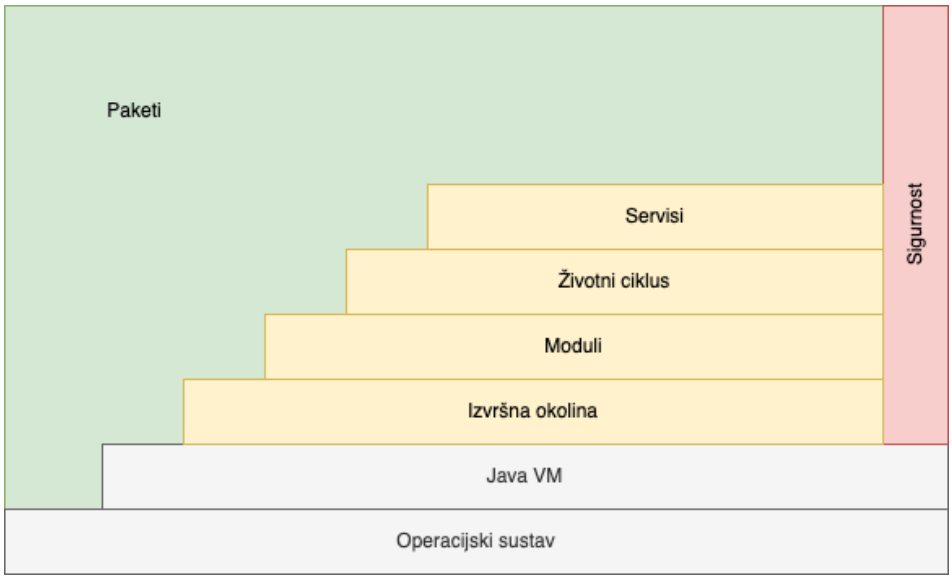
Slika 13. Arhitektura AEM-a (prema [18])

4.2. OSGi

OSGi je temeljni element AEM-ove arhitekture, a sastoji se od specifikacija koje definiraju dinamičke komponente. Takve specifikacije omogućavaju razvoj aplikacije koja je građena od niza različitih manjih komponenata koje se „pakiraju“ u male pakete (eng. bundle) pa tako paket može sadržavati jednu ili više komponenti. Svaki takav paket se može instalirati, pokrenuti, zaustaviti ili deinstalirati neovisno jedan o drugome bez da ugroze rad cijelog sustava. Takvom tehnologijom komponente mogu sakriti svoju implementaciju od ostalih komponenata te mogu komunicirati preko servisa.

Glavna svrha OSGi tehnologije je učitavanje klase iz manjih paketa. Kod tradicionalnih aplikacija JAR paketi su vidljivi na linearnoj listi koju je potrebno cijelu pretražiti kako bi došli do željene klase. Kod OSGi aplikacija jasno je određeno koji paket sadrži koju klasu pa nije potrebno pretraživati sve pakete što doprinosi brzini rada aplikacije [19].

Na slici 14 prikazana je OSGi arhitektura.



Slika 14. Arhitektura OSGi (Abobe systems [19])

- Paketi – sadrže jednu ili više OSGi komponenti
- Servisi – povezuju komponente, tj. preko njih komponente međusobno komuniciraju, svaki paket može koristiti nijedan ili više servisa
- Životni ciklus – API koji instalira, pokreće, zaustavlja, ažurira i deinstalira pakete
- Moduli – definiraju kako će paketi uvesti ili izvesti kôd
- Sigurnost – upravlja sigurnosnim aspektima sustava
- Izvršna okolina – definira metode i klase koje su dostupne specifičnoj platformi

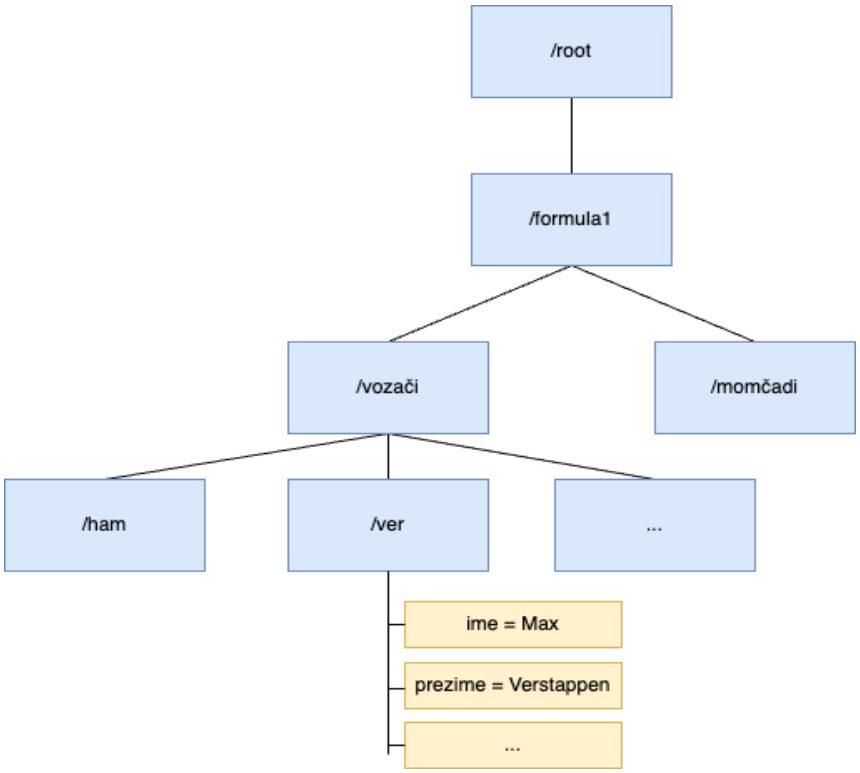
4.3. Java Content Repository (JCR)

Java Content Repository je baza podataka koja strukturom podsjeća na datotečni sustav, a služi za spremanje svih podataka koji pripadaju AEM-u, odnosno samoj aplikaciji, pa se tako u JCR spremaju i HTML, JS, CSS datoteke kao i slike, dokumenti i ostali strukturirani ili nestrukturirani sadržaji.

Velika prednost ovakve baze podataka je da podržava pretraživanje po tekstu, indeksiranje, kontrolu pristupa i verzioniranje. Već je spomenuto da ovakva baza više podsjeća na datotečni sustav nego na klasičnu relacijsku bazu podataka. Tako možemo reći da JCR koristi kombinaciju dobrih značajki iz obje tehnologije. Od značajki relacijskih baza podataka JCR koristi transakcije, upite i integritet, dok od datotečnog sustava koristi mogućnost direktnog zapisa binarnih vrijednosti, hijerarhiju te kontrolu pristupa pojedinim zapisima. Ostale značajke kojima se JCR može pohvaliti su spremanje više vrijednosti za jedno svojstvo, sortiranje i slično [20].

4.3.1. Čvorovi i svojstva

Prema službenoj dokumentaciji [17], JCR ima strukturu stabla koja se sastoji od čvora i svojstva. Čvorovi predstavljaju strukturu, a svojstva su zadužena za spremanje podataka.



Slika 15. Prikaz čvorova i svojstava (autorski rad)

Na slici 15 prikazan je primjer spremanja podataka u JCR. Važno je da čvorovi imaju jasno i jedinstveno definirano ime jer u JCR ne postoji identifikacijski ključ koji bi jednoznačno definirao čvor, odnosno vrijednost, već se podacima pristupa određenom putanjom pa tako u gornjem primjeru do informacija pojedinog vozača dolazimo putanjom /forumula1/vozači/ver.

Čvorovi u JCR mogu imati:

- Čvor roditelj ili čvor dijete koji su prezentirani svojom putanjom
- Tip koji određuje pravila samog čvora ili čvora niže razine, tj. određuje koja svojstva moraju biti zadana tim čvorovima. Npr. čvor tipa cq:page mora imati dijete čvor jcr:content tipa cq:PageContent
- Bilo koji broj svojstava

Svojstva čvora definirana su imenom i vrijednosti. Već je spomenuto da JCR pruža mogućnost spremanja više vrijednosti u jedno svojstvo. Takva svojstva koja poprimaju više vrijednosti ponašaju se kao struktura polja (polje[ ]).

Svi čvorovi i svojstva pripadaju određenom imenskom prostoru. U tablici 1 prikazani su imenski prostori i njihovo objašnjenje.

Imenski prostor	Objašnjenje
jcr:	Osnovna pohrana podataka
nt:	Temeljni tip čvora
rep:	Interni repozitorij
mix:	Standardni mixin tip čvora
sling:	Tip čvora ili svojstva koji dodaje Sling razvojni okvir
cq:	Tip čvora ili svojstva koje dodaje AEM aplikacija

Tablica 2. Imenski prostori i objašnjenje (prema Adobe Systems [17])

Neki od tipova čvora ili svojstva za pohranu podataka iz imenskog prostora jcr su:

- jcr:createdBy – svojstvo koje sprema korisničko ime korisnika koji je kreirao čvor
- jcr:lastModified – svojstvo koje sprema datum i vrijeme zadnje promjene čvora
- jcr:title – svojstvo koje sprema naslov čvora ili resursa

Temeljni tipovi čvorova predstavljaju strukturu pohranjivanja pa tako imamo:

- nt:file – predstavlja datoteku
- nt:folder – predstavlja direktorij
- nt:unstructured – podržava bilo koju kombinaciju čvora djeteta sa svojstvima, koristi se za spremanje nestrukturiranih sadržaja

Neki od čvorova koje definira AEM su sljedeći:

- cq:Page – sprema svojstva i sadržaj web stranice
- cq:Template – definira koji predložak je korišten za stvaranje stranice
- cq:Component – predstavlja komponentu
- cq:Design – definira koji dizajn je korišten kod kreiranja stranice
- cq:Dialog – definira optimiziran dijalog komponente
- cq:Tag – definira oznaku koje se koriste kako bi se kategorizirao sadržaj

Kada bi uspoređivali JCR repozitorije, primijetili bi da im je početna struktura jednaka, tj. svaki repozitorij sadrži sljedeće direktorije:

- /apps – sadrži sve prilagođene komponente vezane uz aplikaciju kao i predloške, pakete, i18n prijevode, biblioteke i slično. Apps direktorij važan je i za nasljeđivanje, tj. modifikaciju osnovnih ili sistemskih komponenti i funkcija koje izvorno koristi AEM, a smještene su u /libs direktoriju. Svaku takvu komponentu koja se nalazi u /libs, a želimo ju prilagoditi, potrebno je „preslikati“ u /apps. Prilikom rezolucije URL-a i dohvaćanja skripti ili resursa prvo se pretražuje /apps direktorij te ako nema rezultata u /apps, pretražuje se /libs. Time se sprječava direktno mijenjanje jezgrinih funkcija i komponenata.
- /conf – sadrži sve konfiguracije web stranice poput dinamičkih predložaka i pravila koja određuju koja komponenta smije biti na kojoj poziciji.
- /content – sadrži sav sadržaj koji se nalazi na web stranicama. Svaka web stranica spremljena je kao čvor te sadrži svojstva koja definiraju sadržaj same stranice. U ovom čvoru također se sprema sva digitalna imovina poput slika, videa, dokumenata koja je učitana u AEM.
- /etc – sadrži resurse vezane uz uslužne programe i alate. Obično se tu spremaju oznake (eng. tags) koje kategoriziraju sadržaj, radni tokovi i slično.
- /home – sadrži podatke vezane za korisnike i korisničke grupe.
- /libs – kao što je već spomenuto, libs direktorij sadrži sve sistemske i jezgrine komponente i biblioteke koje pripadaju AEM-u.
- /oak:index – sadrži definicije Jackrabbit Oak indeksa. Svaki čvor specificira detalje jednog indeksa koji služe za poboljšanje performansi pretraživanja.
- /temp – služi kao direktorij za privremeno spremanje podataka.
- /var – sadrži datoteke koje ažurira sustav. To su najčešće datoteke poput dnevnika, statistike, zapisi o upravljanju događajima.

4.3.2. CRXDE

CRXDE (eng. Content Repository Extreme/Development Environment) je alat koji služi za upravljanje JCR repozitorijem u pregledniku. Koristeći ovaj alat moguće je kreirati, ažurirati, brisati te pretraživati čvorove i njihova svojstva. CRXDE također omogućuje kreiranje paketa sa sadržajem. Ovom tehnologijom moguće je kreirati paket sa željenim sadržajem, spremiti ga i preuzeti, ali isto tako možemo instalirati pakete pa je time omogućena neka vrsta kopiranja odnosno premještanja repozitorija.

Na slici 16. prikazan je CRXDE alat.

Slika 16. CRXDE (autorski rad)

4.4. Apache Sling

Prema definiciji [21] Apache Sling je „razvojni okvir za RESTful web aplikacije bazirane na proširivom sadržaju oblikovanog u stablo“. Sling je orijentiran prema resursima, a zadaća mu je mapiranje URL-a HTTP zahtjeva na resurs bazirano na putanji, nastavku i selektoru. Sav sadržaj u JCR repozitoriju izložen je kao HTTP resurs. Nakon određivanja sadržaja, Sling određuje koji će servlet ili skripta rukovati zahtjevom na temelju [17]: svojstva čvora traženog sadržaja, HTTP metoda zahtjeva te standardnim ključnim riječima zadanim u samom URL-u.

Kod svakog zahtjeva prvi korak Slinga je rastaviti URL na smislene cjeline. Rastavljanje URL-a prikazano je na sljedećoj tablici.

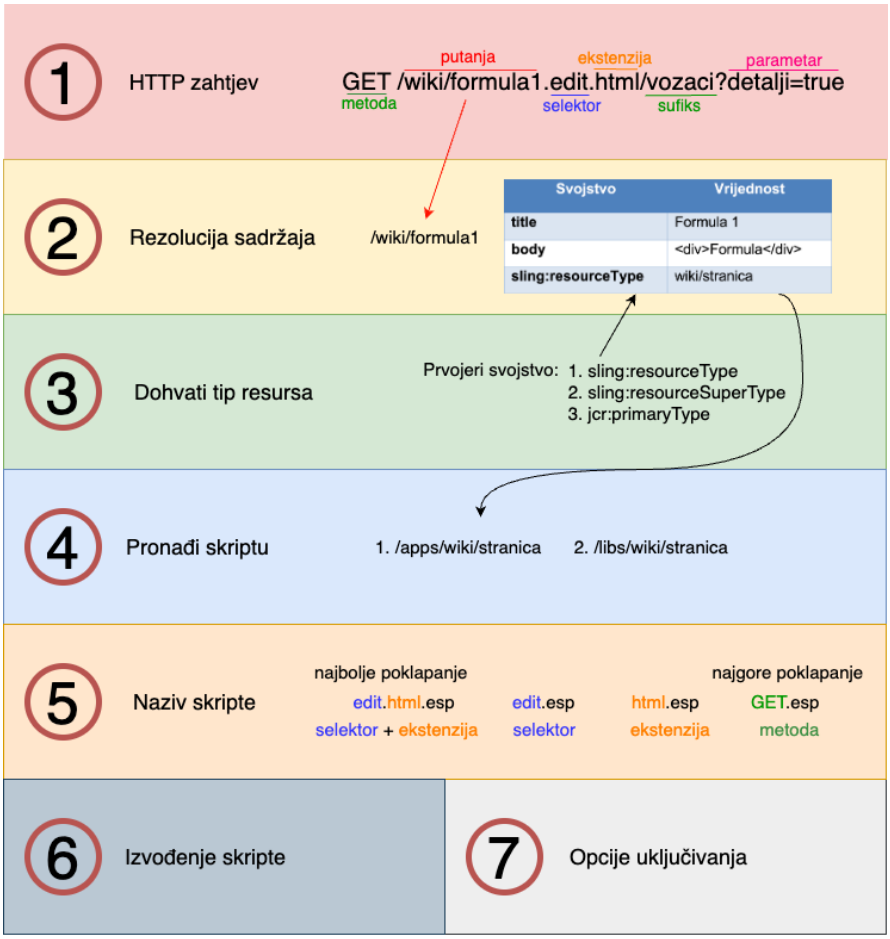
Protokol	Domaćin	Putanja sadržaja	Selektor	Ekstenzija		Sufiks		Parametar
http://	localhost	/content/formula1	.vozaci	.html	/	a/b	?	detalji=true

Tablica 3. Dekompozicija URL-a (prema [17])

- Domaćin – naziv domaćina web stranice
- Putanja sadržaja – putanja koja definira lokaciju sadržaja
- Selektor – korišten za alternativne metode prikazivanja sadržaja
- Ekstenzija – format sadržaja koji definira skriptu korištenu za prikaz
- Sufiks – korišten za specificiranje dodanih informacija
- Parametar – korišten za prosljeđivanje podataka

4.4.1. Sling dekompozicija URL-a

Nakon inicijalne dekompozicije URL-a na cjeline cilj Slinga je pronaći servlet ili skriptu koja će rukovati zahtjevom. Detaljan proces URL dekompozicije i traženje skripte prikazan je na sljedećoj slici.



Slika 17. Sling URL dekompozicija (prema [22])

Proces URL dekompozicije, odnosno određivanje servleta ili skripte koja će rukovati zahtjevom sastoji se od 7 koraka. U prvom koraku URL se dijeli na prije spomenute smislene cjeline. Kako bi se krenulo s rezolucijom sadržaja, važno je promatrati putanju jer ona određuje sadržaj kojem se pristupa. U drugom koraku promatra se čvor sadržaja, točnije njegova svojstva. Ovdje je zadatak Slinga pronaći tip resursa traženog sadržaja. Svojstva koja određuju tip resursa su *sling:resourceType, sling:resourceSuperType* te *jcr:primaryType*. Sling upravo ovim redoslijedom, tj. prioritetom traži tip resursa. Svojstvo tipa resursa je zapravo putanja na kojoj se nalazi traženi resurs. Već je kod JCR spomenuto da se sistemske i jezgrine komponente i funkcije moraju „preslikati“ iz /libs direktorija u /apps ukoliko ih je potrebno mijenjati. Upravo zbog toga se u četvrtom koraku prilikom traženja resursa prvo pretražuje /apps direktorij dok se /libs pretražuje ako nema rezultata unutar /apps direktorija. Nakon dohvaćanja resursa odabire se servlet ili skripta koja ima najbolje podudaranje s navedenim

parametrima. Nakon pronalaska skripte kreće se s izvršavanjem te uključivanjem dodatnih metoda ukoliko je potrebno.

4.5. AEM instance

AEM instanca je zapravo „kopija“ AEM-a koja se izvodi na serveru. Svaka kopija često se izolira u posebno okružje kako bi se osigurala odvojenost, tj. podjela odgovornosti. Samim time ovakav pristup pridonosi sigurnosti jer se za svaku instancu mogu posebno kreirati korisnici i dodijeliti im posebna prava, za svaku instancu moguće je kreirati odvojene konfiguracije, a prednost je također da se akcije na svakoj instanci mogu odvijati paralelno.

Za rad AEM-a potrebne su najmanje dvije instance, a to su autor i *publish* (izdavač). Te dvije instance članovi razvojnog tima postavljaju lokalno kako bi razvijali komponente, modele, servise te provjerili radi li sve kao što je očekivano.

4.5.1. Autor

Autor instancu koriste kreatori sadržaja, uređivači sadržaja te administratori kako bi pripremili, tj. kreirali ili uredili sadržaj koji će kasnije biti objavljen na publish instanci kojoj pristupaju krajnji korisnici. Autor instanca pruža grafičko sučelje namijenjeno manipulaciji sadržaja koje olakšava i ubrzava procese kreiranja i uređivanja sadržaja.

Neke od glavnih značajki rada na autoru su sljedeće [23]:

- Korištenje predefiniranih predložaka za kreiranje stranica – autori si olakšavaju proces kreiranja stranica na način da prvo kreiraju predloške za najčešće korištene stranice (naslovnica, stranica kategorije, stranica sa sadržajem, galerija)… Prilikom kreiranja nove stranice autori biraju potreban predložak te dobivaju „kostur“ tj. temeljni izgled stranice sa svim blokovima, tj. mjestima namijenjenim za pojedine komponente.
- Kreiranje, uređivanje i brisanje sadržaja – korištenjem sučelja autori imaju različite prozore poput navigacijske ploče, kartice s alatima, vrpce s opcijama, različite poglede na strukture stranica, stablo komponenata na stranici, pretraživač komponenti, filter i slično. Samo kreiranje, uređivanje i brisanje sadržaja svodi se na već spomenute komponente od kojih svaka sadrži svoj dijalog za unos podataka koji su potrebni kako bi komponenta znala kako i koji sadržaj treba prikazati.
- Kreiranje, uređivanje, brisanje i upravljanje digitalnom imovinom – sav medijski sadržaj poput slika, videa i dokumenata prenosi se na autor instanci, a

objavljivanjem pojedinog medijskog sadržaja ili stranice koja sadrži iste, taj sadržaj postaje dostupan na publish instanci.

- Objavljivanje (eng. publish) ili sakrivanje (eng. unpublish) sadržaja – nakon kreiranog sadržaja isti je potrebno objaviti kako bi postao dostupan na publish instanci. Isto tako je dostupna i reverzna metoda koja poništava objavljen sadržaj te ga briše s publish instance.
- Upravljanje korisnicima – svakom korisniku administrator može dodijeliti određena prava pa tako neki korisnici mogu kreirati, uređivati i brisati sadržaj dok neki mogu samo gledati sadržaj. Sva prava mogu se dodijeliti za pojedinu lokaciju ili čvor u JCR što pruža velike mogućnosti i dodjeljivanje širokog spektra prava.
- Kreiranje, upravljanje i pokretanje radnih tokova – radni tokovi su skup pojedinih radnji ili aktivnosti koje se pokreću samostalno ili na zahtjev korisnika kada se zadovolje neki uvjeti. Npr. česti radni tok je automatsko kreiranje različitih rendicija slike ili videa prilikom prenošenja pojedine slike u AEM, a sa strane autora radni tok može biti proces objave sadržaja gdje se radni tok pokreće kada je autor završio s kreiranjem sadržaja, te radni tok prebacuje na odgovornog menadžera koji može nastaviti radni tok objavom sadržaja ili vraćanjem sadržaja na doradu ili ispravak uz priopćenje komentara.

Autor instancu razvojni inženjeri pokreću lokalno, obično na portu 4502 kako bi provjerili ispravnost svoga koda, odnosno replicirali uočene pogreške na višim razvojnim okolinama te iste otklonili.

4.5.2. Publish

Nakon objave sadržaja na autor instanci, taj sadržaj postaje dostupan na publish instanci. Publish instanca služi kao produkcijsko okružje namijenjeno krajnjim korisnicima. Na publishu je zapravo sav sadržaj koji je objavljen s autor instance, a kojemu korisnici trenutno mogu pristupiti.

Glavne značajke publish instance su sljedeće [23]:

- Servira sadržaj krajnjim korisnicima
- Krajnji korisnici su u interakciji sa stranicama na ovom okruženju
- Krajnji korisnici se mogu prijaviti ili registrirati kako bi pristupili dodatnim sadržajima
- Personalizirani sadržaj može biti dostupan samo određenim segmentima korisnika

- Praćenje ponašanja korisnika – korištenjem dodanih alata kao Adobe Analytics moguće je kreirati različite izvještaje o posjećenosti stranice

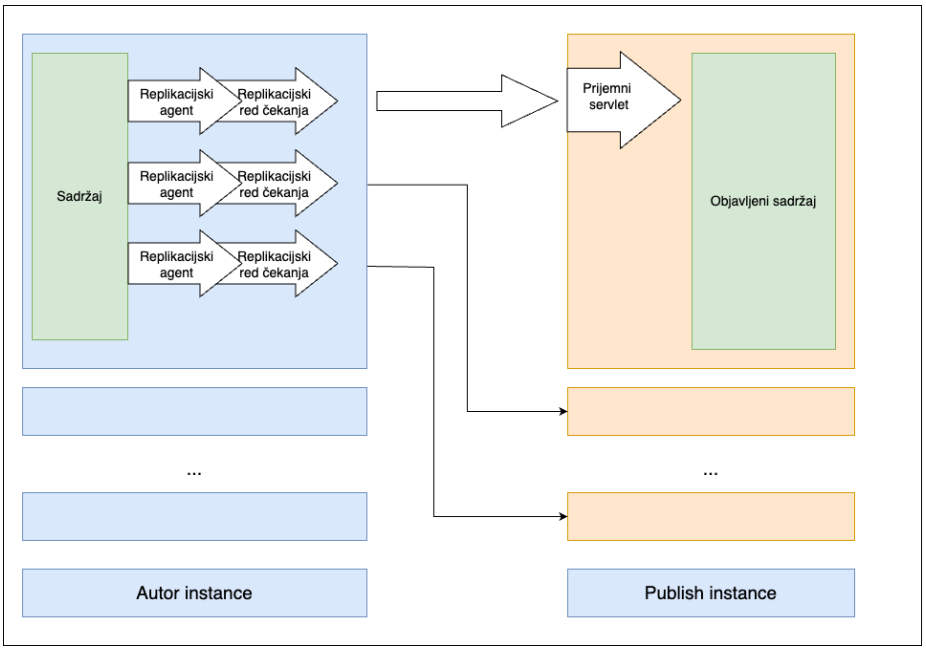
Često se događa da neka funkcionalnost radi na autor instanci, ali ima neočekivane pogreške na publish instanci. Stoga se i publish instanca pokreće lokalno, obično na portu 4503, kako bi razvojni inženjeri replicirali uočene pogreške te ih ispravili.

4.5.3. Replikacija

Kada je sadržaj kreiran i odobren na autor instanci, još uvijek nije dostupan krajnjim korisnicima koji posjećuju web stranice na publish instanci. Kako bi izjednačili, tj. prenijeli sadržaj s autora na publish potrebno je željenu stranicu replicirati tj. aktivirati ili objaviti.

Replikacija je proces prijenosa sadržaja i podataka između dvije različite instance. Cilj ovog procesa je sinkronizacija, tj. izjednačavanje sadržaja na obje instance. Najčešća replikacija provodi se s autora na publish. Za provedbu replikacije zaslužni su replikacijski agenti. Radi se o AEM mehanizmu koji se koriste za:

- Objavu (aktivaciju) stranice s autora na publish
- Čišćenje sadržaja iz keš memorije dispečera
- Vraćanje korisnikovog unosa (npr. forma) s publisha na autor instancu



Slika 18. Replikacija autor – publish (prema [24])

Na slici 18. prikazan je proces replikacije s autor na publish instancu. Koraci su sljedeći [24]:

1. Korisnik zahtjeva objavu stranice ili sadržaja koji želi objaviti. To može učiniti pritiskom na gumb „Objavi“ (eng. Publish) ili može konfigurirati okidač koji će objaviti stranicu u točno određeno vrijeme.
2. Zahtjev za objavu sadržaja prosljeđuje se zadanom replikacijskom agentu.
3. Replikacijski agent stvara paket sa sadržajem koji je potrebno objaviti i stavlja paket u replikacijski red čekanja.
4. Kada taj paket dođe na red, uzima se iz reda čekanja i prenosi se na publish instancu koristeći protokol koji je definiran u konfiguraciji. Najčešći protokol za prijenos je HTTP.
5. Prijemni servlet na publish instanci prima zahtjev te instalacijom paketa objavljuje sadržaj. Zadani prijemi servlet okida se na adresi *https://localhost:4503/bin/receive*

Imajući na umu da je publish instanca ona s kojom su korisnici u interakciji, možemo očekivati da će korisnici unositi neke podatke na tu instancu. Primjerice ispuniti formu, registrirati se, unositi podatke u svoj profil i slično. Navedeni podatci bit će spremljeni na publish instanci. Ukoliko te podatke želimo imati i na autor instanci potrebno je napraviti obrnutu replikaciju (eng. Reverse replication).

4.6. AEM okružja

AEM okružje je zapravo skup instanci (autor, publish), konfiguracija i alata. Postojanje više okružja omogućava izoliranost instanci, tj. promjene na jednom okružju ne utječu na druga okružja. Ideja postojanja više od jednog okružja je obavljanje različitih aktivnosti na pojedinom okružju. Najčešća okružja su dev, stage i prod.

4.6.1. Dev

Dev okružje je posvećeno razvojnim inženjerima za izradu i testiranje komponenata, za isprobavanje novih funkcionalnosti, za testiranje funkcionalnosti nakon određenih modifikacija i promjena.

Dev zapravo služi kao „igraonica“ (eng. sandbox, playground) gdje razvojni inženjeri mogu slobodno eksperimentirati, validirati i isprobavati različite modifikacije, različite slučajeve korištenja te različite implementacije nekog rješenja. Prednost takvog okružja je sama odvojenost, tj. spomenute promjene neće utjecati na rad autor i publish instance na drugom okružju.

Glavne značajke dev okružja su sljedeće:

- Razvojna okolina – kontrolirana okolina koja ne utječe na rad ostalih te time pruža mogućnost razvojnim inženjerima testiranja različitih rješenja
- Testiranje i otklanjanje grešaka – ispravljanje grešaka učinkovitije je na dev okružju zbog realističnosti, tj. varijable okoline i konfiguracija više sliči produkcijskom okružju nego na lokalnoj autor instanci pa je jednostavnije replicirati uočene pogreške. Također na dev okružju moguće je provesti različita testiranja, od jediničnih testova pa sve do integracijskih.
- Suradnja – razvojni inženjeri mogu koristiti jednu instancu te korištenjem Gita ili drugih alata za verzioniranje mogu stvarati svoje grane. Spajanjem grana mogu testirati funkcionalnosti koje su ovisne jedna o drugoj.

4.6.2. Stage

Stage okružje još se naziva kao pre-produkcija, a samo ime govori da se nalazi neposredno prije produkcije, tj. to je okružje na kojem se rade završne provjere prije „puštanja“ sadržaja na produkciju.

Neki od glavnih zadataka koji se obavljaju na stage su sljedeći:

- Testiranje i validacija – s obzirom da je stage okruženje koje najviše sliči produkciji, na njemu autori i uređivači detaljno testiraju vizualnu i funkcijsku ispravnost komponenata
- Pregled sadržaja – autori sadržaja te vlasnici koriste stage kako bi detaljno pregledali ispravnost sadržaja prije puštanja u produkciju
- Funkcionalno i integracijsko testiranje – izvodi se detaljno testiranje funkcionalnosti te se provjerava da li sve izgleda kao što je očekivano. Takvim testiranjem doznaje se ima li kritičnih ili manjih grešaka. Ukoliko greške nisu pronađene ili su minimalne sadržaj se može objaviti te će se iste naknadno ispraviti.
- Testiranje performansi i opterećenja – moguće je izvesti različite scenarije kako bi se testirale performanse pod određenim opterećenjem. Tu se obično simuliraju interakcije korisnika koristeći različite alate. Takvim testiranjem dolazi se do informacija kao što su vrijeme odaziva, što je jedan od najvažnijih kriterija svake web stranice ili aplikacije.

4.6.3. Prod

Prod je okružje koje se referencira na produkciju. Prod sadrži glavnu (eng. live) publish instancu koja servira sadržaj krajnjim korisnicima. Prema tome možemo reći da je prod zadnja stanica životnog ciklusa (lokalno – dev – stage – prod).

Glavne karakteristike prod okružja su:

- Serviranje sadržaja krajnjim korisnicima
- Velika dostupnost i skalabilnost – okružje je prilagođeno velikom opterećenju pa se broj instanci može automatski povećavati ovisno o broju interakcija i korisnika. Automatsko povećavanje broja aktivnih instanci poznato je kao balansiranje opterećenjem (eng. Load Balancing)
- Nadziranje i održavanje – s obzirom da je prod glavno okružje, greške na ovom okružju rezultiraju smanjenu ili nikakvu dostupnost web stranice. Zbog toga važno je kontinuirano praćenje i nadziranje ovog okružja.
- Upravljanje keš memorijom – važno je dobro upravljanje keš memorijom kako bi se smanjio odaziv i kako bi se većina sadržaja koji se ne mijenja spremila u keš memoriju

4.7. Dispečer

Dispečer (eng. Dispatcher) je AEM-ov alat koji služi za keširanje i/ili balansiranje opterećenjem s ciljem da poboljša performanse, skalabilnost i sigurnost prilikom dostavljanja sadržaja. Dispečer se nalazi između krajnjih korisnika i publish instance, a cilj mu je serviranje sadržaja korisniku kako bi smanjio opterećenje na publish instancu.

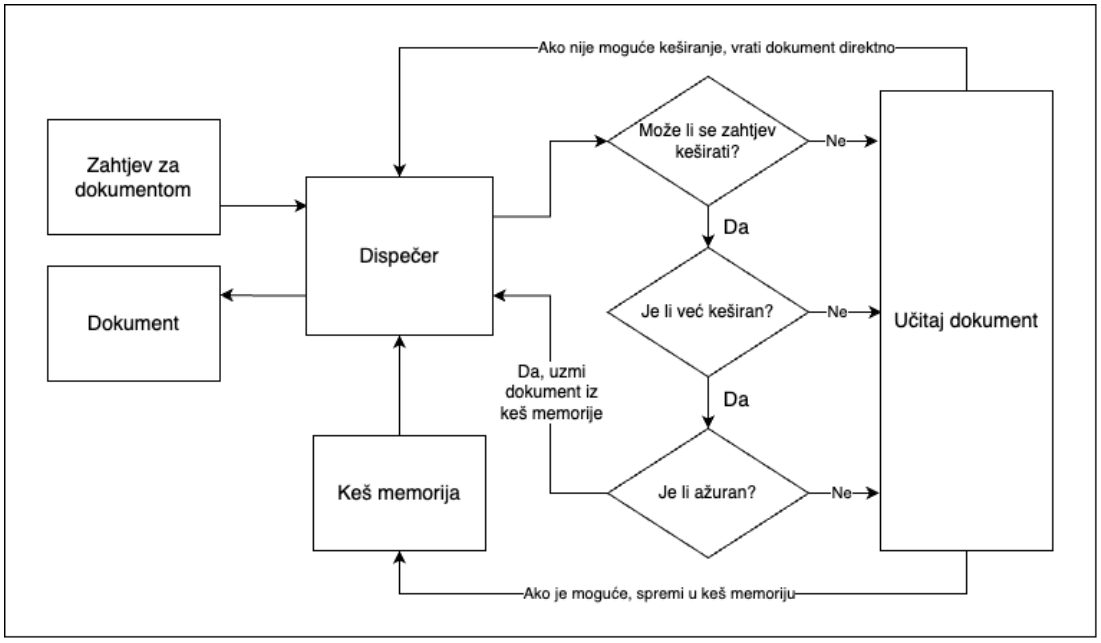
Kako Adobe navodi [25] dva su osnovna pristupa web objavljivanju:

- Statički web poslužitelj – kao Apache ili IIS – jednostavni i brzi
- Poslužitelji za upravljanje sadržajem – pružaju dinamički, trenutni, tj. ažurni, bogati sadržaj ali zahtijevaju više resursa za brze odgovore

Dispečer pomaže stvaranju okružja koje je brzo i dinamično, a radi kao dio statičkog HTML servera poput Apachea s ciljem da sprema, tj. keširа što je više sadržaja moguće u formi statičke web stranice te da pristupa instanci sa sadržajem što manje.

Na taj način statički sadržaj servira se brzinom klasičnog statičkog web servera dok je dinamički sadržaj kreiran po potrebi s ciljem da ne usporava instancu više nego je potrebno.

Na slici 19. prikazano je kako dispečer upravlja i vraća dokumente.



Slika 19. Prikaz rada dispečera (prema [25])

Dispečer očekuje zahtjeve za dokumentom. Kada takav zahtjev stigne od strane publish instance, dispečer prvo provjerava listu dokumenata koji se mogu keširati. Ako se dokument ne nalazi na listi, potrebno je zahtijevati i učitati dokument s publish instance. Dispečer će uvijek dohvaćati dokumente s publish instance ukoliko:

- URL zahtjeva sadrži znak „?“ – upitnik u URL-u označava da se šalje neki parametar te da se radi o dinamičkoj stranici primjerice tražilica. Takvi zahtjevi se ne keširaju.
- Dokumentu nedostaje ekstenzija – web poslužitelj zahtijeva ekstenziju kako bi odredi o kojoj vrsti dokumenta je riječ
- Konfiguracija zaglavlja nalaže da se zahtjev ne kešira

Ako se dokument može keširati, potrebno je provjeriti da li isti već postoji u keš memoriji. Ako traženi dokument ne postoji u keš memoriji, potrebno ga je zahtijevati od publish instance, vratiti direktno te ga spremiti u keš memoriju. Ako dokument već postoji u keš memoriji, potrebno je provjeriti da li je ažuran, tj. treba se uvjeriti da nije došlo do promjene dokumenta nakon što je isti spremljen u keš memoriju. Kako bi dispečer provjerio ažurnost dokumenta, izvodi dva koraka:

- Provjerava da li je dokument element automatske invalidacije što znači da se automatski briše iz keš memorije kada dođe do promjene dokumenta. Samim time,

ako dokument jest element automatske invalidacije i postoji u keš memoriji, smatra se da je ažuran.

- Provjerava se zadnja promjena dokumenta. Ako je dokument jednak zadnjoj promjeni smatra se da je ažuran. Međutim, postoji li novija verzija dokumenta, dispečer zahtijeva tu verziju od publish instance te zamjenjuje svoju staru verziju za noviju koju vraća instanca.

Dispečer nudi brojne konfiguracije kojima se mogu zadavati različita pravila poput liste dokumenata koji se smiju i ne smiju keširati, filteri, pravila za preusmjeravanje, skraćivanje URL adresa i slično.

5. Izrada aplikacije u AEM

Kako bi započeli s izradom aplikacije u AEM-u potrebno je kreirati lokalno okružje, tj. pripremiti potrebne alate i tehnologije npr. Maven za izgradnju projekta, razvojni okvir za pisanje programskog koda , Github za verzioniranje i slično. Nakon toga potrebno je pokretanje i postavljanje author i publish instance.

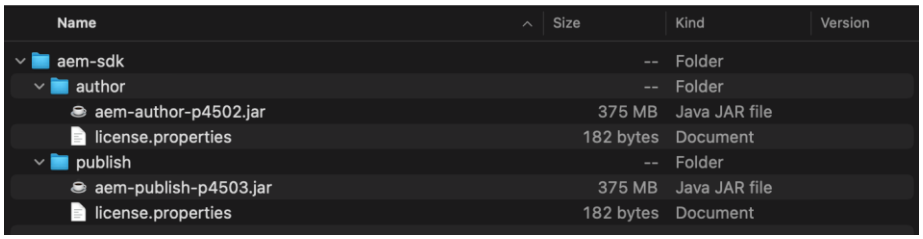
5.1. Postavljanje okružja i projekta

Postavljanje lokalnog okružja prvi je korak prilikom izrade aplikacije u AEM-u. Već je spomenuto da je lokalna AEM instanca kopija koja se pokreće na računalu svakog razvojnog inženjera.

5.1.1. Postavljanje lokalnog okružja

Kako bi ispravno postavili lokalno okružje potrebno je kreirati određenu strukturu datoteka. Novo kreirani direktorij treba se sastojati od dva direktorija – author i publish. Unutar author direktorija potrebno je smjestiti Quickstart JAR datoteku. Radi se o izvršnoj datoteci koja sadrži sve što je potrebno za pokretanje instance. U ovom slučaju koristi se JAR datoteka i licenca poduzeća IBM iX.

Nakon premještanja QuickStart JAR datoteke unutar author direktorija, potrebno ju je preimenovati u „aem-author-p4502.jar“. Isto je potrebno učiniti i za publish direktorij, ali za publish JAR datoteku je potrebno preimenovati u „aem-publish-p4503.jar“. Struktura direktorija prikazana je na sljedećoj slici:



Slika 20. Struktura direktorija za Autor i Publish instancu (autorski rad)

Pokretanje instanci moguće je dvostrukim klikom na izvršnu JAR datoteku ili pokretanjem naredbe:

```
$ java -Xmx2048M -Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=30303 -jar aem-
author-p4502.jar -gui -r"author,localdev"
```

Nakon pokretanja raspakirat će se JAR datoteka, instalirati se i pokrenuti. Rezultat ovog procesa je autor instanca koja se pokreće na localhost-u na portu 4502. Nakon završetka procesa možemo pristupiti instanci na adresi: `https://localhost:4502`

5.2. Postavljanje projekta

Za postavljanje projekta potrebno je instalirati Apache Maven. Radi se o alatu koji upravlja izgradnjom (eng. Build) i isporukom (eng. Deploy) java baziranih projekata.

Postavljanje projekta izvodi se arhetipom (eng. Archetype). Arhetip je Maven predložak tj. originalni uzorak po kojem su izrađeni svi drugi projekti ovog tipa. Korištenjem arhetipa razvojni inženjeri su naviknuti na konzistentnu strukturu projekta koja prati najbolje prakse.

AEM arhetip generirat će minimalni projekt koji slijedi principe najbolje prakse te je početna točka svake AEM stranice. Prednosti korištenja arhetipa su sljedeće [26]:

- Najbolja praksa – korištenje Bootstrapa sa svim najnovijim preporukama od strane Adobea
- Mogućnost za migraciju na oblak – projekt je spreman za migraciju na oblak te ga je u svega nekoliko dana moguće koristiti
- Dispečer – dolazi s konfiguriranim dispečerom koji doprinosi brzini i sigurnosti
- Više stranica – ako je potrebno, moguće je kreirati sadržaj za više jezika ili više regija
- Osnovne komponente – autori mogu kreirati sadržaj koristeći set standardnih komponenata

- Predlošci – moguće je kreirati predloške bez pisanja koda te definirati što autori smiju uređivati
- Responzivni izgled – na predlošcima ili pojedinim stranicama moguće je definirati skaliranje za manje zaslone
- Sustav stilova – kako bi izbjegli kreiranje istih komponenata s drugim stilovima, omogućeno je mijenjanje stilova na komponenti
- Spremnost za WebApp – za stranice koje koriste React ili Angular tehnologiju moguće je koristiti SPA SDK za lakše uređivanje stranice
- Spremnost za Commerce – projekti se mogu integrirati s Adobe Commerce poput Magento
- Primjer koda – postoji HelloWorld komponenta te primjeri modela, servisa, servleta
- Otvoren kod – velika mogućnost pronalaska rješenja

Prilikom kreiranja projekta potrebno je zadati određena svojstva koja će definirati ime, osobine, izgled i arhitekturu projekta [26]:

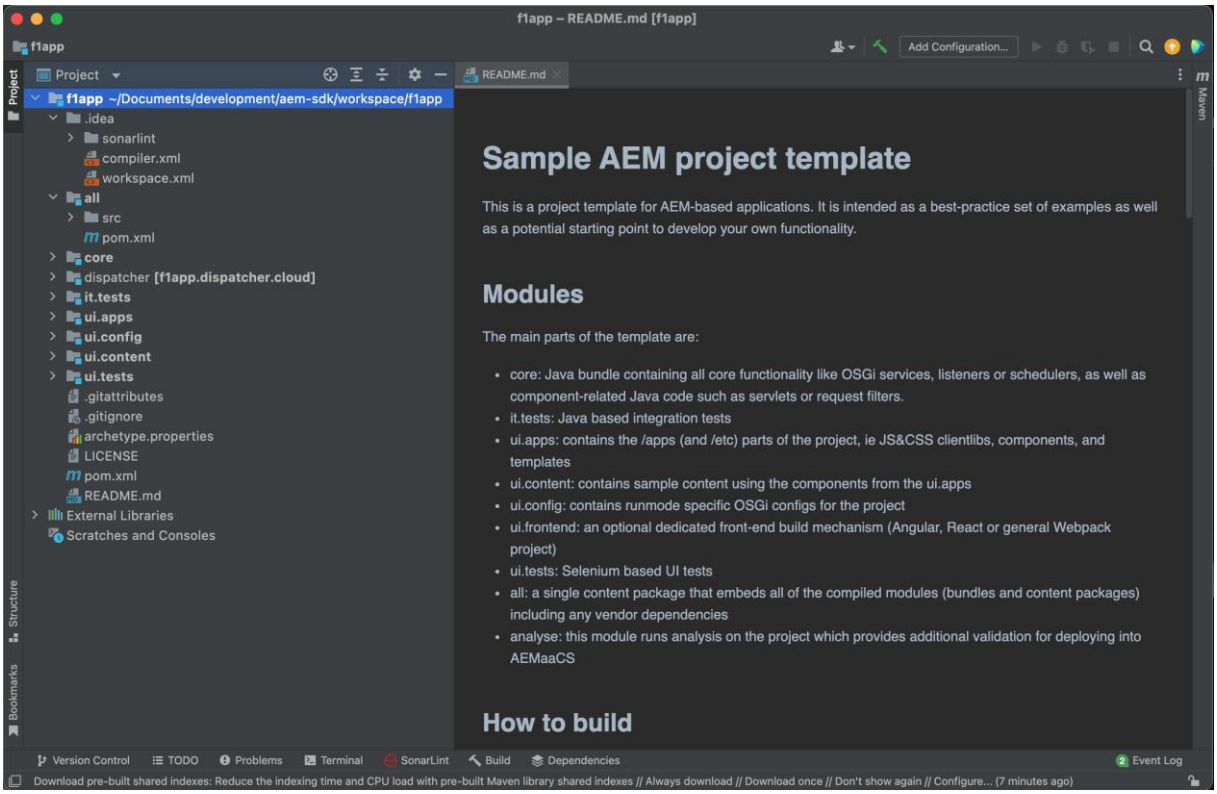
Svojstvo	Zadano	Opis
appTitle		Naziv aplikacije, koristit će se kao naslov za web stranicu. Npr. „F1 aplikacija“
appId		Tehničko ime, korišteno za komponente, konfiguracije, naziv direktorija i slično. Npr. „f1app“
artifactId	${appId}	Osnovni ID Maven artifakta. Npr. „f1app“
groupId		Osnovni ID Maven grupe. Npr „com.f1app“
package	${groupId}	Java izvorišni paket. Npr. „com.f1app“
version	1.0-SNAPSHOOT	Verzija projekta
aemVersion	cloud	Verzija AEM-a. Npr cloud, 6.5.0, 6.4.4
sdkVersion	latest	Ako je odabrana AEM verzija „cloud“, može se postaviti verzija SDK-a.
includeDispatcherConfig	y	Uključuje konfiguraciju dispečera. Može biti y (yes) ili n (no)
frontendModule	general	Uključivanje razvojnog okvira za razvoj korisničke (prednje) strane generiranjem klijentskih biblioteka. Može biti: general, none, angular ili react
language	en	Jezični kod za kreiranje strukture sadržaja
country	us	Kod države za kreiranje strukture sadržaja
singleCountry	y	Uključuje strukturu za glavni jezik (language-master). Može biti y ili n
includeExamples	n	Uključuje stranicu s primjerima komponenti. Može biti y ili n

Tablica 4. Svojstva prilikom generiranja projekta (izvor [26])

Za generiranje projekta potrebno je postaviti željena svojstva. Primjer naredbe za generiranje projekta je:

```
mvn -B archetype:generate
 -D archetypeGroupId=com.adobe.aem
 -D archetypeArtifactId=aem-project-archetype
 -D archetypeVersion=43
 -D aemVersion=cloud
 -D appTitle="F1 App"
 -D appId="f1app"
 -D groupId="foi.ahip20.f1app"
 -D frontendModule=none
 -D includeExamples=n
```

Nakon izvršenja naredbe kreirat će se predložak tj. „kostur“ projekta za izradu AEM aplikacije. Izgled projekta može se vidjeti na sljedećoj slici:



Slika 21. Izgled strukture projekta (autorski rad)

Možemo primijetiti da se projekt sastoji od sljedećih modula:

- all – sadrži sve ugrađene prevedene module kao i ovisnosti
- core – sadrži svu poslovnu logiku aplikacije, tu se smještaju sve osnovne funkcionalnosti aplikacije sa strane poslužitelja. Tu se smještaju Sling modeli, servisi, servleti i slično.
- dispatcher – sadrži sve konfiguracije vezane za rad dispečera. Tu se nalaze pravila za keširanje, filteri, pravila za preusmjeravanje i slično.
- it.tests – sadrži Java bazirane integracijske testove
- ui.apps – sadrži sve potrebno za razvoj na strani poslužitelja. Tu se nalazi HTML prilagođenih komponenata, njihovi dijalozi i opcije. Ovdje se također spremaju CSS i Javascript datoteke.
- ui.config – sadrži sve potrebne OSGi konfiguracije
- ui.content – tu se nalazi web sadržaj koji će biti kreiran prilikom izgradnje i isporuke projekta
- ui.tests – sadrži Selenium bazirane testove

5.3. Pristupi za izradu aplikacije u AEM

U modernom razvoju web aplikacija obično se nastoji razdvojiti frontalni razvoj od razvoja pozadinske logike. Razvojni inženjeri se obično opredijele za učenje i usavršavanje tehnologija razvoja ili frontalnog ili pozadinskog dijela aplikacije. Na taj način svaki razvojni tim se obično dijeli na 2 tima. Jedan tim zadužen je za razvoj pozadinskog (poslužiteljskog) dijela aplikacije, a drugi za frontalni (klijentski) dio aplikacije. Tako timovi mogu raditi paralelno, neovisni jedni o drugima.

Ovisno o tehnologijama korištenim za razvoj frontalnog dijela aplikacija postoji nekoliko pristupa koji se mogu primijeniti prilikom izrade aplikacije u AEM-u. Najčešći pristupi su standardni ili tradicionalni, SPA pristup te pristup korištenjem web komponenti. Za usporedbu navedenih pristupa bit će implementirane jednostavne komponente jer smatram da se razlika najbolje uočava na jednostavnim komponentama.

5.3.1. Standardni pristup

Standardni ili tradicionalni pristup je način izrade aplikacije gdje se izvršavanje aplikacije oslanja na renderiranje na strani poslužitelja. Kod takvog pristupa web stranice su generirane i renderirane na strani servera i potpuni renderirani HTML šalje se korisniku, tj. web pregledniku kako bi prikazao sadržaj.

Kada korisnik pristupi web stranici, komponente dinamički generiraju svoj HTML dohvaćanjem sadržaja iz JCR repozitorija, procesiranjem poslovne logike komponente i renderiranjem odgovarajućeg HTL predloška. Razvoj frontalnog dijela svodi se na kreiranje HTL predložaka, Javascript i CSS datoteka, a razvoj pozadinskog dijela obuhvaća implementacije Sling modela za komponente, servise, servlete i slično.

5.3.1.1. Komponente

AEM komponente su zapravo gradivi blokovi web stranice, a koriste se kako bi spremale, formatirale i prikazivale sadržaj na web stranici. Glavne značajke AEM komponenata su [17]:

- Rezervirano mjesto – kada se komponenta doda na stranicu i nema sadržaja prikazuje se okvir s imenom komponente koji služi kao rezervirano mjesto (eng. Placeholder). Taj okvir vidljiv je samo u autorskom radu, tj. ukoliko se stranica pogleda „pogledaj kao objavljenu“ taj okvir nije vidljiv.
- Validacija dijaloga – svaka komponenta sadrži dijalog u kojem autori unose sadržaj koji komponenta treba prikazati te opcije koje komponenti govore na koji način da prikaže taj sadržaj. Svako polje dijaloga moguće je validirati.
- Sustav stilova – omogućava autorima da definiraju CSS klase nad komponentom kako bi promijenili izgled. Na taj način nije potrebno implementirati istu komponentu s različitim izgledom.
- Sling modeli za komponentu – mogućnost definiranja Java model objekta te mapiranja tog objekta na Sling resurs.
- Izvoz (eng. Export) Sling modela – uključuje anotacije koje se dodaju nad Sling modelom, a definiraju kako će se model izvesti u formatu JSON-a.
- Selektori – pružaju mogućnost odabira skripte koja će biti renderirana kada korisnik zahtijeva stranicu.
- I18n prijevod – pruža mogućnost višejezičnosti. Unutar komponente moguće je staviti oznaku, te za svaki jezik oznaci dodati prijevod.

Glavni elementi komponente su korijenski čvor, vitalna svojstva te vitalni čvorovi djeca. Korijenski čvor komponente možemo prepoznati kao čvor s imenom komponente, a tipa je cq:Component.

Vitalna svojstva komponente su sljedeća:

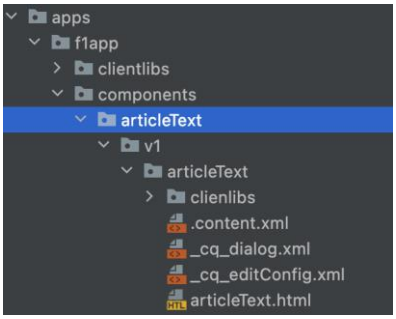
- jcr:title – predstavlja naziv komponente
- jcr:description – predstavlja opis komponente

Vitalni čvorovi djeca su:

- cq:editConfig (cq:EditConfig) – definira svojstva za uređivanje komponente
- cq:dialog (nt:unstructured) – predstavlja dijalog komponente u kojem korisnici unose sadržaj i opcije prikazivanja komponente

Komponente su najčešće podijeljene u dva direktorija a tu su *structure* i *content*. U *structure* direktorij pripadaju komponente koje određuju strukturu stranice, primjerice zaglavlje, podnožje, stranica i slično, dok u *content* pripadaju komponente koje služe za generiranje sadržaja. Npr. tekst, slika, galerija…

Najjednostavnija komponenta koja se može kreirati je komponenta koja će prikazivati tekst. Struktura i datoteke koje komponenta sadrži prikazani su na slici 22.



Slika 22. Struktura i datoteke *ArticleText* komponente (autorski rad)

Datoteka .content.xml sadrži osnovne informacije o komponenti poput imena, primarnog tipa, opisa, grupe i slično. Grupa komponente zadaje se kako bi ograničili korištenje komponente. Ukoliko komponenta ima grupu „F1 base“ takvu komponentu možemo dodati u kontejner samo ako podržava dodavanje komponenata koje pripadaju toj grupi.

Sadržaj .content.xml datoteke je sljedeći:

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root                          xmlns:cq="http://www.day.com/jcr/cq/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0"
          cq:isContainer="{Boolean}false"
          jcr:primaryType="cq:Component"
          jcr:title="Article text"
          componentGroup="F1 base">
    <cq:htmlTag
        jcr:primaryType="nt:unstructured"
        class="component"/>
</jcr:root>
```

Već je spomenuto da dijalozi služe kako bi korisnici unosili sadržaj i opcije prikazivanja komponente. Dijalog se gradi koristeći osnovne coral komponente kao što su *text*, *textarea*, *number*, *select* i slično. Ukoliko se radi o zahtjevnijem dijalozima, moguće je kreirati više kartica kako bi se grupirao unos podataka.

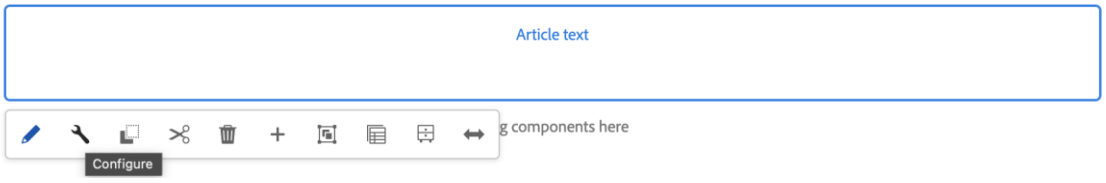
Dijalog se implementira unutar _cq_dialog.xml datoteke. U nastavku je prikazana datoteka jednostavnog dijaloga za *ArticleText* komponentu:

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0"
          xmlns:nt="http://www.jcp.org/jcr/nt/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0"
          jcr:primaryType="nt:unstructured"
          sling:resourceType="cq/gui/components/authoring/dialog"
          jcr:title="Article text">
    <content
        jcr:primaryType="nt:unstructured"
        sling:resourceType="granite/ui/components/coral/foundation/fixedcolumns">
        <items jcr:primaryType="nt:unstructured">
            <column
                jcr:primaryType="nt:unstructured"
                sling:resourceType="granite/ui/components/coral/foundation/container">
                <items jcr:primaryType="nt:unstructured">
                    <article-text
                        jcr:primaryType="nt:unstructured"
                        sling:resourceType="granite/ui/components/coral/foundation/form/textarea"
                        required="{Boolean}true"
                        fieldLabel="Article text"
                        name="./articleText"/>
                </items>
            </column>
        </items>
    </content>
</jcr:root>
```

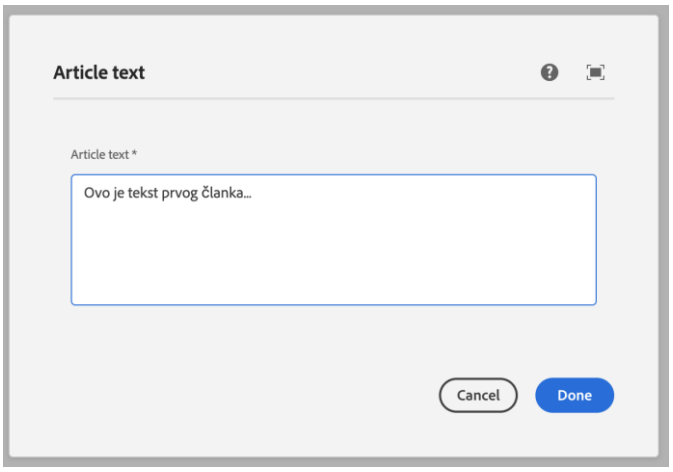
Slika 23. Datoteka dijaloga *ArticleText* komponente

Vidljivo je da se dijalog sastoji od jednog polja tipa *textarea*. Od bitnijih svojstava možemo primijetiti *required* koji označava da li je potrebno unijeti neku vrijednost, *fieldLabel* je zapravo tekst koji opisuje polje za unos, a svojstvo *name* je zapravo mjesto spremanja unosa. U ovom slučaju tekst će biti spremljen kao svojstvo same komponente, a ime svojstva je *articleText*.

Kako bi otvorili dijalog, potrebno je kliknuti na ikonu ključa.



Slika 24. Otvaranje dijaloga (autorski rad)



Slika 25. Dijalog *ArticleText* komponente (autorski rad)

Datoteka cq_editConfig određuje svojstva uređivanja, tj. može definirati radnje koje je moguće obaviti nad komponentom. Primjerice uređivanje, kopiranje, premještanje, brisanje.

Datoteka articleText.html je zapravo HTL predložak.

5.3.1.2. HTL

HTL (eng. HTML Template Langue) je jezik implementiran od strane Adobea, a cilj mu je bio zamijeniti Java Server Pages (JSP). HTL Koristi ekspresije kako bi se ugradio neki sadržaj ili varijabla unutar samog HTML dokumenta. Prilikom prevođenja, ekspresije i atributi tj. dinamičke vrijednosti su učitane iz Java koda na strani poslužitelja pa tako ekspresije i atributi nisu vidljivi u HTML datoteci koja se zatim dostavlja klijentu [27].

Glavni ciljevi HTL-a su sljedeći:

- Pružati poboljšanu sigurnost i spriječiti *cross-site scripting* napade
- Pojednostavniti razvoj komponenata – HTL je jednostavan i piše se brže od HTML-a
- Smanjiti troškove izrade aplikacije
- Povezati pozadinski i frontalni razvoj – HTL može koristiti Sling model, koji će biti uskoro objašnjen, te dohvaćati vrijednosti iz samog modela. Na taj način se poslovna logika odvija u Sling modelu i servisima, a rezultat se prikazuje kao ekspresija.

HTL se sastoji od dva tipa sintaksi:

- HTL blokovi – definiraju strukturalne elemente te izvršavaju HTL specifične atribute. Oznaka blokova je <sly> dok svaki HTL specifični atribut započinje s *data-sly-.*

- HTL ekspresije – označavaju se koristeći znakove ${}. Ako želimo naznačiti da se radi o nekoj dinamičkoj vrijednosti odnosno varijabli, ime varijable stavlja se unutar zagrada. Npr. ${properties.title}.

  Neki od osnovnih HTL atributa, tagova i ekspresija su sljedeći:

  - Komentari
    ```
    <!--/* HTL komentar */-->
    ```
  - Ekspresije
    ```
    ${true}, ${properties.text}
    ```
  - Manipulacija podataka
    ```
    ${text @ context='html'}
    ```
  - HTL Blokovi
    - data-sly-use – koristi se kako bi se pristupilo Sling modelu i njegovim atributima
      ```
      <sly data-sly-use.model="…models.TextArticleModel"></sly>
      <p>${model.vrijednost}</p>
      ```
    - data-sly-test – koristi se kako bi se provjerilo sadrži li varijabla vrijednost. Ako ne sadrži, element neće biti renderiran.
      ```
      <div data-sly-test.text=${model.text}>${text}</div>
      ```

  U nastavku je prikazan HTL predložak komponente *ArticleText*:

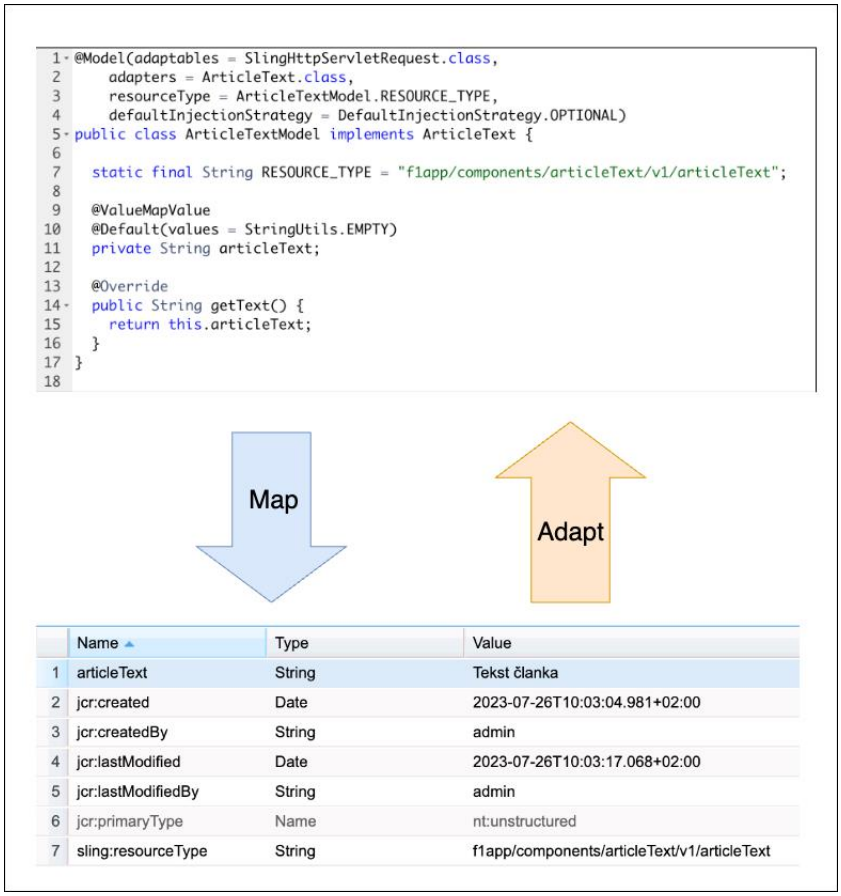
```
<sly data-sly-use.model="foi.ahip20.f1app.core.models.ArticleText"></sly>

<div  data-sly-test.isEmpty="${!model.articleText}"  data-emptytext="Article
text" class="cq-placeholder">

</div>

<p data-sly-test="${!isEmpty}" class="article">${model.articleText}</p>
```

Prvo se u *model* sprema klasa koja će se koristiti kao Sling model komponente. Zatim se provjerava da li postoji tekst. Ako tekst postoji ispisat će se unutar <p> elementa. Ako tekst ne postoji, tj. varijabla je prazna, prikazat će se rezervirano mjesto sa tekstom „Article text“ što autoru daje do znanja da komponenta nije konfigurirana.

5.3.1.3. Sling modeli

Sling modeli su Java klase smještene unutar *ui.core* modula koje mogu biti mapirane na Sling resurs, *SlingHttpServletRequest* ili čak OSGi servis korištenjem metode *adaptTo()*. Sling model je zapravo apstrakcijski sloj koji omogućava AEM komponenti da koristi pozadinski dio aplikacije za implementaciju poslovne logike [19].

Sling modeli su Java klase koje se nalaze u OSGi paketima koje su anotirane sa *@Model* te ključnom riječju *adaptables* označene na koju klasu se adaptiraju. Na sljedećoj slici prikazano je mapiranje i adaptiranje.



Slika 26. Mapiranje i adaptiranje (prema [19])

U primjeru prikazanom na slici možemo vidjeti da je klasa anotirana sa *@Model* te da *adaptables* određuje da će se adaptirati na *SlingHttpServletRequest*. U klasi također primjećujemo anotaciju *@ValueMapValue* koja služi za mapiranje istoimenih svojstava. Klasa također sadrži *getText* metodu kojom HTL dohvaća vrijednost varijable.

Prednosti korištenja Sling modela su sljedeće [19]:

- Smanjuje vrijeme implementacije jer nije potrebno implementirati vlastite adaptere
- Podržava i klase i sučelja
- Omogućava adaptiranje različitih objekata npr. *Resource* ili *SlingHttpsServletRequest*
- Rade s postojećom Sling infrastrukturom
- Postoje standardne anotacije

Neke od anotacija koje se koriste u Sling modelima su sljedeće:

Antoacija	Primjer kôda	Opis
@Model	@Model(adaptables=Resource.class)	Označuje klasu kao Sling model te određuje klasu adaptiranja
@Inject	@Inject private String text;	Ime svojstva (text) bit će pretraženo unutar resursa nakon adaptiranja te će varijabla poprimiti vrijednost koja je u resursu. Ako svojstvo nije pronađeno vraća se *null*
@Default	@Inject @Default(values="AEM") private String text;	Ukoliko resurs nema svojstvo i vrati *null*, ova anotacija sadrži vrijednost koje će biti dodijeljeno varijabli kako bi se izbjegla *null* vrijednost.
@Named	@Inject @Named(value="articleText") private String text;	Služi za dohvaćanje svojstva kada ime varijable ne odgovara imenu svojstva. U ovom slučaju traži se svojstvo koje je zadano unutar @Named anotacije (articleText)
@PostConstruct	@PostConstruct public void postConstruct() { log.info("Hello World"); }	Omogućava izvršavanje metode odmah nakon što je instancirana klasa i izvršeno adaptiranje

Tablica 5. Anotacije Sling modela (prema [19])

Postoji više anotacija kojima se injektiraju određeni elementi. Anotacije za injektiranje prikazane su u sljedećoj tablici:

Antoacija	Podržani elementi	Injektira
@ValueMapValue	optional, name, via	Pojedino svojstvo resursa
@ChildResource	optional, name, via	Resurs dijete trenutnog resursa
@RequestAttribute	optional,name,via	Atribut iz zahtjeva
@OSGiService	optional, filter	OSGi servis
@SlingObject	optional	Sling objekt primjerice *ResourceResolver*

Tablica 6. Anotacije za injektiranje (prema [19])

Sling modeli također podržavaju izvoz u JSON formatu. Ovo svojstvo koristi se kako bi se omogućio pristup podacima, a naziva se *Sling Model Exporter*. Za ovakav izvoz modela

postoji anotacija *@Exporter* koja definira u kako i u kojem formatu će se podaci izvesti. Najčešći format izvoza je JSON koji zatim prednji dio aplikacije može koristiti za dohvaćanje vrijednosti iz modela. Takvo dohvaćanje specifično je kod SPA pristupa pa će u tom dijelu biti detaljno objašnjeno.

5.3.1.4. Klijentske datoteke

Od svih tehnologija još nedostaju klijentske datoteke, tj. JavaScript i CSS koje se koriste za razvoj prednjeg frontalnog dijela aplikacije. Ove datoteke spremaju se u *clientlib* direktorij koji se nalazi unutar direktorija same komponente. Svojstva ovog direktorija su sljedeća:

- categories – identificira u koju kategoriju JS i CSS datoteke pripadaju. Svojstvo može poprimiti više vrijednosti što omogućuje da jedan direktorij može biti dio jedne ili više kategorija.
- dependencies – pruža listu drugih ili vanjskih kategorija o kojima ovaj direktorij ovisi
- embed – služi kako bi se ugradio kôd iz nekih drugih direktorija. Ako čvor je čvor G ugrađen u čvor H, HTML će se sastojati od kôda čvorova G i H.
- allowProxy – ako je direktorij smješten unutar /apps direktorija, ovo svojstvo omogućava pristup preko proxy servleta
- js.txt/css.txt – datoteke koje pomažu identifcirati koje JS i CSS datoteke treba uključiti

Na sljedećoj slici prikazana je struktura *clientlib* direktorija:

Slika 27. Struktura *clientlib* direktorija (autorski rad)

Najvažniji je sadržaj .content.xml datoteke:

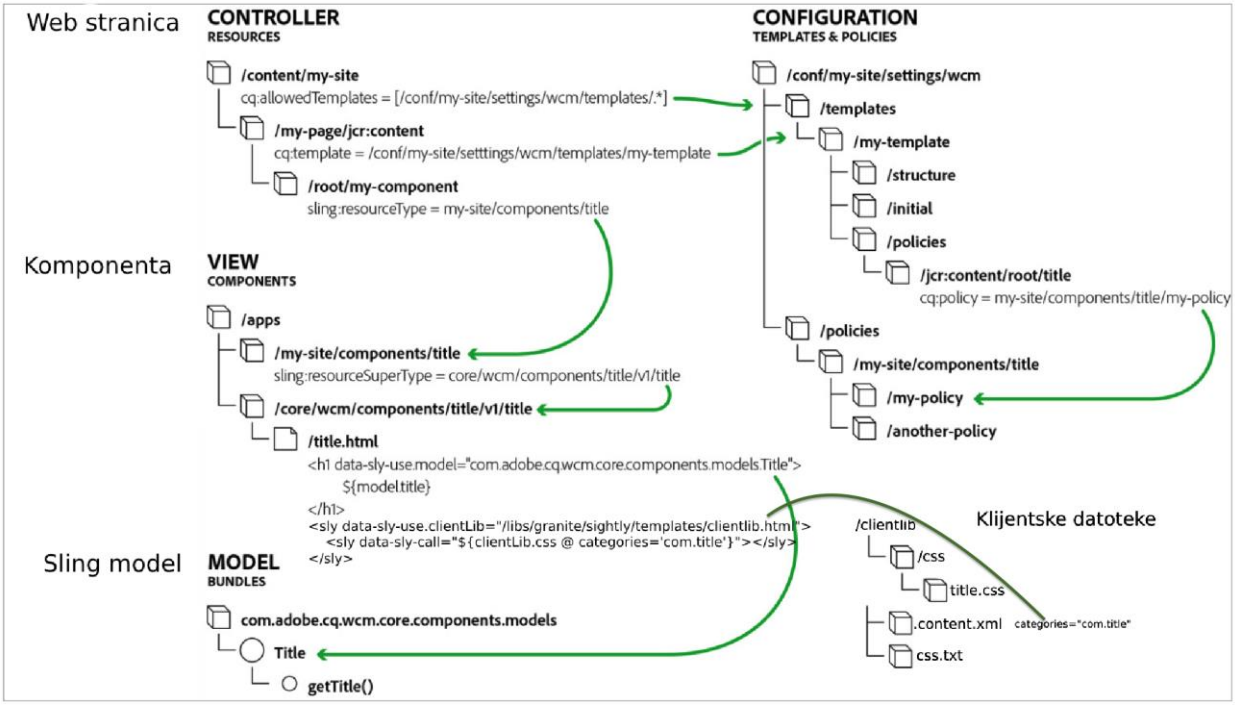
```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0"
xmlns:jcr="http://www.jcp.org/jcr/1.0"
          jcr:primaryType="cq:ClientLibraryFolder"
          allowProxy="{Boolean}true"
          categories="[f1app.component.articleText]" />
```

Možemo primijetiti da je zadana kategorija „f1app.component.articleText“. Kategorija je ključna jer se preko nje referencira ovaj direktorij unutar HTL predloška:

```
<sly data-sly-
use.clientLib="/libs/granite/sightly/templates/clientlib.html">
      <sly data-sly-call="${clientLib.css @ categories=
      'f1app.component.articleText'}"> </sly>
</sly>
```

5.3.1.5. Način rada

Sada su objašnjene sve tehnologije koje se koriste kod ovog pristupa pa ih je moguće smjestiti na jedan dijagram:



Slika 28. Standardni pristup (prema [19])

Prilikom kreiranja stranice moguće je odabrati jedan od ponuđenih predložaka. Predložak određuju strukturu stranice, tj. raspored kontejnera u kojima se dodavaju komponente. Svaka komponenta ima zadanu grupu kojoj pripada, čime se ograničava

korištenje komponente u pojedinom kontejneru. Nakon postavljanja komponente na stranicu u HTL-u se koristi Sling model koji sadrži poslovnu logiku komponente. Korištenjem anotacija za injektiranje dohvaćaju se svojstva komponente koja određuje kakav zadatak i nad kojim podacima poslovna logika treba odraditi. Nakon izvršenja poslovne u HTL-u su dostupne varijable te se na strani poslužitelja generira finalni HTML dokument koji se šalje u korisnički preglednik koji prikazuje sadržaj.

5.3.2.SPA pristup

Već je u drugom poglavlju objašnjeno da kod SPA aplikacija postoji samo jedna stranica na kojoj se izmjenjuje web sadržaj. Na taj način izbjegnuto je osvježavanje cijele stranice pa tako i preuzimanje čitave HTML datoteke.

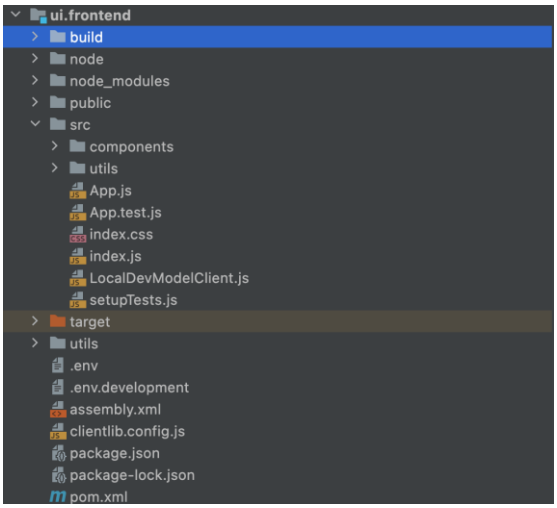
Kod SPA pristupa uočljivo je razdvajanje pozadinskog i frontalnog razvoja jer postoji potpuno odvojen modul *ui.frontend* u kojem se nudi snažan JavaScript razvojni okvir poput Reacta, Angulara ili Vue. Korištenjem ovih razvojnih okvira stvara se izuzetno bogato grafičko sučelje koje nudi visoku razinu interakcije sa brzim tranzicijama i animacijama.

5.3.2.1. Postavljanje projekta

Kod postavljanja projekta u SPA pristupu važno je postaviti opciju *frontendModule*. Moguće opcije su *React* ili *Angular* ovisno o razvojnom okviru koji će se koristiti unutar modula. Primer naredbe za izradu SPA projekta sa *React* razvojnim okvirom je sljedeći:

```
mvn -B org.apache.maven.plugins:maven-archetype-plugin:3.2.1:generate \
 -D archetypeGroupId=com.adobe.aem \
 -D archetypeArtifactId=aem-project-archetype \
 -D archetypeVersion=43 \
 -D appTitle="F1 App SPA React" \
 -D appId="f1app-spa-react" \
 -D artifactId="f1app.spa.react" \
 -D groupId="foi.ahip20.f1app.spa.react" \
 -D frontendModule="react" \
 -D aemVersion="cloud"
```

Nakon izvršenja Maven naredbe kreirati će se projekt koji ima uključen *ui.frontend* modul.



Slika 29. Struktura *ui.frontend* modula (autorski rad)

Na slici 29. vidimo samu strukturu *ui.frontend* modula. Unutar *src* direktorija vidimo direktorij *components* u kojem se zapravo nalaze Javascript datoteke za implementaciju frontalnog dijela komponenti. Također primjećujemo *index.js* datoteku koja je zapravo „ulazna točka“ SPA aplikacije. Sadržaj *index.js* datoteke prikazan je u nastavku:

```
const renderApp = () => {
    ModelManager.initialize(modelManagerOptions).then(pageModel => {
        const history = createBrowserHistory();
        render(
            <Router history={history}>
                <App
                    history={history}
                    cqChildren={pageModel[Constants.CHILDREN_PROP]}
                    cqItems={pageModel[Constants.ITEMS_PROP]}

                    cqItemsOrder={pageModel[Constants.ITEMS_ORDER_PRO
                    P]}
                    cqPath={pageModel[Constants.PATH_PROP]}
                    locationPathname={window.location.pathname}
                />
            </Router>,
            document.getElementById('spa-root')
        );
    });
};
```

U ovom dijelu kôda možemo primijetiti da se inicijalizira *Model Manager* koji zatim renderira početnu i jedinu stranicu na kojoj će se prikazivati sve ostale stranice ovisno o zahtjevima korisnika.

5.3.2.2. React

React je JavaScript biblioteka razvijena od strane Facebook-a 2013. godine. React služi za renderiranje korisničkog sučelja izrađenog od više manjih dijelova koji se nazivaju komponentama. Cilj primjene komponenti je ponovna iskoristivost. Komponente se također mogu ugnijezditi, tj. svaka komponenta može sadržavati druge komponente koje predstavljaju djecu toj komponenti. Najveća prednost React-a je mogućnost izrade SPA stranica jer omogućava izradu komponenti koje dinamički mijenjaju svoj sadržaj bez potrebe za osvježavanjem cijele stranice. To stvara prednost u odnosu na druge pristupe implementacije jer se ističe brzinom, jednostavnošću te oku ugodnim animacijama [28].

Navedene prednosti te optimizacija performansi postiže se korištenjem virtualnog DOM-a (eng. Document Object Model). DOM predstavlja dokument s logičkim stablom koje čine čvorovi elementi pa tako određuje strukturu i sadržaj web stranice. Korištenjem virtualnog DOM-a React neće direktno ažurirati DOM nego će promjene primijeniti na virtualnom DOM-u. Nakon toga slijedi usporedba DOM-a i virtualnog DOM-a te se DOM ažurira samo s potrebnim promjenama.

React koristi JSX (JavaScript XML) sintaksu koja omogućava pisanje kôda koji nalikuje na HTML unutar JavaScripta. Prilikom izgradnje taj kôd se prevodi u čisti JavaScript.

Još jedna prednost Reacta je korištenje *State* i *Props* tipova podataka. *State* predstavlja unutarnje stanje komponente te se ne može prosljeđivati, dok *Props* predstavlja tip podatka koji se može prosljeđivati između komponenti. *Props* tip podatka također služi prosljeđivanju varijabli iz poslovne logike.

U nastavku je primjer jednostavne React komponente:

```
class Text extends React.Component {
  render() {
    return <h2>Tekst koji se renderira!</h2>;
  }
}
```

Glavna metoda unutar React komponente je *render()* koja zapravo generira sadržaj koji će se prikazati u pregledniku.

5.3.2.3. Komponente

Kod ovog pristupa definicija komponente i dijalog ostaju smješteni unutar *ui.apps* modula, u *components* direktoriju, međutim direktorij komponente više ne sadrži HTML, CSS i JS datoteke za renderiranje komponenti. Razvoj frontalnog dijela komponente sada se premješta u *ui.frontend* gdje je na raspolaganju *React* razvojni okvir. Sama implementacija

komponente smješta se u *ui.frontend/src/components* direktorij gdje se kreira direktorij sa nazivom komponente u koji se smješta JavaScript datoteka za renderiranje komponente.

Slika 30. Struktura komponenti unutar *ui.frontend* modula (autorski rad)

Na slici 30. možemo primijetiti *ArticleText.js* datoteku u kojoj se nalazi implementacija frontalnog dijela aplikacije. Sadržaj je prikazan u nastavku:

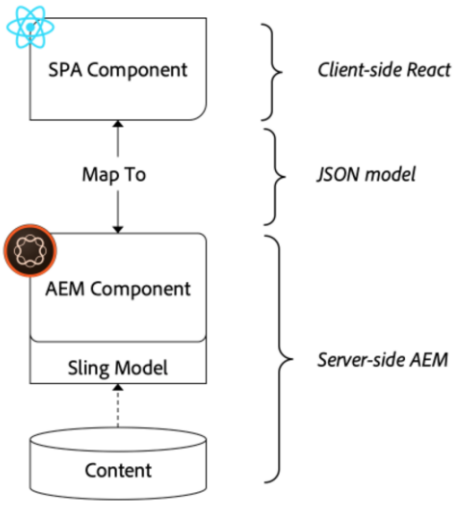
```
import React, {Component} from 'react';
import {MapTo} from '@adobe/aem-react-editable-components';
require('./ArticleText.css')
export const ArticleTextEditConfig = {
    emptyLabel: 'Article Text',
    isEmpty: function(props) {
        return !props || !props.text || props.text.trim().length < 1;
    }
};
export default class ArticleText extends Component {
    render() {
        if(ArticleTextEditConfig.isEmpty(this.props)) {
            return (null);
        }
        return (
            <div>
                <p className="articleText">{this.props.text} </p>
            </div>
        );
    }
}

MapTo('f1app-spa-react/components/articleText')(ArticleText,
ArticleTextEditConfig);
```

Pri samom početku potrebno je uključiti *React, Component* te *MapTo* ovisnosti. U trećoj liniji koda uključuje se CSS komponente koji se nalazi u istom direktoriju. U nastavku slijedi *ArticleTextEditConfig* koji je zaslužan za prikazivanje rezerviranog mjesta s nazivom komponente. Možemo primijetiti funkciju *isEmpty* koja prima *props*. *Props* su podaci, tj.

varijable koje dolaze u JSON formatu Sling modela. Spomenuti JSON model nastaje izvozom Sling modela, što će biti prikazano kasnije. Ovim principom povezuje se poslovna logika Sling modela s *React* komponentom. U ovom slučaju možemo primijetiti da će se rezervirano mjesto prikazati ukoliko ne postoji vrijednost varijable *text*.

Sami prikaz komponente implementiran je u klasi *ArticleText* koja nasljeđuje *Component*. U *render* metodi možemo primijetiti da se prvo provjerava rezultat funkcije *isEmpty*. Ako je vraćeno *true*, znači da varijabla *text* ne postoji ili nema vrijednost. U tom slučaju prikazat će se rezervirano mjesto komponente, renderiranje nije potrebno pa se vraća *null* vrijednost. Ako postoji varijabla *text* i ima neku vrijednost, renderirat će se element sa <p> elementom koji prikazuje tekst spremljen u varijabli *text* koja dolazi iz Sling modela.

Na kraju je potrebno povezati komponentu sa Sling modelom. Povezivanje se postiže koristeći *MapTo* funkcije. *MapTo* sadrži putanju komponente iz *ui.apps* koja predstavlja tip resursa (*sling:resourceType)*. Svaki Sling model u svom JSON izvozu dodaje varijablu tipa Sling resursa. Komponenta traži JSON izvoz koji sadrži tip resursa koji odgovara vrijednosti unutar *MapTo* metode, a nakon toga se sve ostale varijable koje sadrži tražena JSON datoteka parsiraju kao *props*.



Slika 31. Povezivanje Sling modela i *React* komponente (preuzeto s [29])

Kako bi bili u mogućnosti koristiti implementiranu komponentu, istu je potrebno uključiti u *import-components.js* datoteci:

```
import './ArticleText/ArticleText'
```

5.3.2.4. Sling modeli

Sama implementacija poslovne logike unutar Sling modela uopće se ne mijenja, ali je potrebno omogućiti izvoz modela u JSON formatu.

Kako bi Sling model mogao pružati više implementacija preporuka je koristiti sučelje koje treba proširivati (eng. Extend) *ComponentExporter* što će rezultirati implementacijom metode *getExportedType*. Kod same implementacije modela potrebno je dodati antoaciju *@Exporter* te navesti naziv *exportera* i ekstenziju (JSON ili XML). Injektiranje vrijednosti svojstava komponente se ne mijenja, jedino je potrebno implementirati *getExportedType* metodu koja će vraćati tip resursa komponente (sling:resourceType) koji će odrediti koja komponenta će se odabrati za renderiranje. Slijedi prikaz sučelja Sling modela *ArticleText* komponente:

```
public interface ArticleText extends ComponentExporter{
     String getText();
}
```

U nastavku je prikazana implementacija sučelja, tj. Sling model:

```
@Model(adaptables = SlingHttpServletRequest.class,
        adapters = { ArticleText.class, ComponentExporter.class },
        resourceType = ArticleTextImpl.RESOURCE_TYPE,
        defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
@Exporter(name = ExporterConstants.SLING_MODEL_EXPORTER_NAME,
        extensions = ExporterConstants.SLING_MODEL_EXTENSION)
public class ArticleTextImpl implements ArticleText {
        static final String RESOURCE_TYPE = "f1app-spa-
react/components/articleText";
          @ValueMapValue
          @Default(values = StringUtils.EMPTY)
          private String text;
          @Override
          public String getText() {
              return this.text;
          }
          @Override
          public String getExportedType() {
              return ArticleTextImpl.RESOURCE_TYPE;
          }
}
```

Možemo primijetiti korištenje anotacije @Exporter kojom naznačujemo korištenje *Jackson Exportera* za izvoz Sling modela u JSON formatu. Ukoliko sada pristupimo komponenti sa selektorom *model* i ekstenzijom *.json* dobivamo izvoz Sling modela u JSON formatu:

Slika 32. Prikaz izvoza Sling modela *ArticleText* komponente (autorski rad)

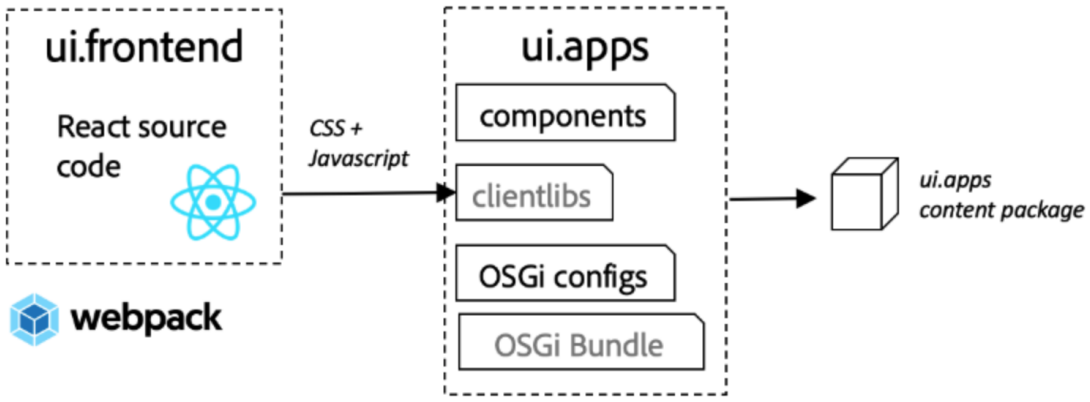
Na slici vidimo da JSON datoteka sadrži dva podatka:

- sling:resourceType – tip resursa kojim se Sling model povezuje sa *React* komponentom. Možemo primijetiti da tip resursa odgovara zadanom u *MapTo* metodi komponente.
- text – varijabla koja će biti dostupna u *React* komponenti korištenjem *props. (props.text)*

5.3.2.5. Klijentske datoteke

Klijentske datoteke pojedine komponente dodaju se u sam direktorij komponente te se zatim uključuju u JavaScript datoteku. S obzirom da se u ovom pristupu radi o renderiranju na jednoj stranici moguće je zadati CSS pravila globalno, tj. na razini stranice. U tom slučaju CSS se sprema unutar *src* direktorija u *index.css* datoteku.

Prilikom izgradnje projekta *npm* modul *aem-clientlib-generator* zadužen je za generiranje *clientlib* datoteka (minificirani JS i CSS) iz svih komponenti i klijentskih datoteka koje se smještaju unutar *ui.apps* modula koji je zatim isporučen u AEM aplikaciju kao paket sadržaja.



Slika 33. Generiranje i isporuka *React* izvornog kôda (preuzeto s [29])

5.3.3. Pristup korištenjem web komponenti

Izrada AEM aplikacije korištenjem web komponenti nije toliko popularna koliko su prethodna dva pristupa. Ovaj pristup svodi se na izradu ponovno iskoristivih, vlastitih i prilagođenih (eng. Custom) elemenata zvanih web komponente. Prije opisa postavljanja projekta za izradu aplikacije u ovom pristupu valja objasniti same web komponente.

5.3.3.1. Web komponente

Web komponente su tehnologija koja omogućava kreiranje vlastitih prilagođenih elemenata koji imaju svojstvo enkapsulacije, tj. učahurivanja čime se „sakrivaju“ i postaju izolirane unutar HTML dokumenta. Ideja za pojavu ove tehnologije bila je ponovna upotrebljivost, tj. otkrivanje tehnologije kojom će se pojedine komponente moći koristiti u više različitih projekata, na više platforma, a da njezini stilovi ne utječu na projekt, ili da stilovi projekta ne utječu na komponentu [31]. Jednostavnije rečeno cilj je kreirati potpuno izoliran element sa svojim skriptama i stilovima.

Rad web komponenti oslanja se na tri glavna dijela, a to su prilagođeni elementi, HTML predlošci te *Shadow* DOM.

Prilagođeni elementi su HTML elementi poput , <img> ali ih sami možemo imenovati. Npr. <ljestvica-vozača>. Takvi elementi proširuju klasične HTML elemente, a omogućavaju da se komponenta doda u HTML korištenjem samo tog jednog elementa koji sadrži izoliranu web komponentu. Web komponente mogu biti implementirane koristeći različite HTML elemente, međutim sadržaj web komponente sakriva se u *shadow DOM*.

Shadow DOM je tehnologija koja omogućava izoliranje dijela stabla DOM-a u posebni i izdvojeni (*shadow*) DOM u kojemu djeluje potpuno izoliran stil (CSS) i JavaScript. Omogućava nam izoliranje tj. enkapsulaciju komponente u prilagođeni element. Korištenjem ove tehnologije sakriva sadržaj web komponente, sprječavaju se moguće kolizije stilova jer *shadow DOM* koristi stil komponente i na njega ne može utjecati nijedan drugi stil. Ova tehnologija oduvijek postoji u internetskim preglednicima pa tako za primjer možemo uzeti video.

```
<video width="320" height="240" controls="">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

U primjeru iznad možemo vidjeti HTML kôd video elementa. Svaki video ima opcije poput kontrole razine zvuka, start/stop gumb i sl. U kôdu iznad ne pronalazimo takve elemente. Možemo zaključiti da je video sadrži *shadow DOM*. Ukoliko u postavkama preglednika uključimo

prikaz *shadow DOM-a* možemo vidjeti kako su navedeni elementi izolirani, tj. sakriveni. Radi jednostavnosti pregleda, kôd sa *shadow DOM-om* prikazan je na sljedećoj slici:

Slika 34: Video element sa *shadow DOM* (autorski rad)

Aktiviranjem prikaza *shadow DOM-a* vidimo pravu implementaciju video elementa od kojih poneki također sadrže svoj *shadow DOM.*

HTML predlošci su korisnički definirani predlošci za prikaz sadržaja koji se ne renderiraju sve dok nisu pozvani, a omogućuju prosljeđivanje podataka korištenjem elementa <slot>.

```
const template = document.createElement('template');

template.innerHTML = `
  <div>
    <h3></h3>
  </div>
`;
class WebKomponenta extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: 'open'});
    this.shadowRoot.appendChild(template.content.cloneNode(true));
    this.shadowRoot.querySelector('h3').innerText =
this.getAttribute("svojstvo");
  }
}
window.customElements.define('web-komponenta', WebKomponenta);
```

U primjeru iznad sadržaj je *web-komponenta.js* datoteke koja sadrži implementaciju web komponente. Na vrhu vidimo predložak koji određuje kako će se komponenta renderirati. Klasa *WebKomponenta* proširuje *HTMLElement* i odgovorna je za renderiranje komponente. Možemo primijetiti *attachShadow* metodu koja omoguće korištenje *shadow DOM-a*. Mod *open*

omogućava pristupanje *shadow DOM-u* korištenjem *this.shadowRoot*. Nakon toga u *shadow DOM* se dodaje sadržaj predloška, dohvaća se <h3> element te mu se dodaje sadržaj atributa s imenom 'svojstvo'. U zadnjoj liniji definira se prilagođeni element koji definira da će se ova web komponenta renderirati unutar elementa s oznakom <web-komponenta>.

Nakon implementacije moguće je komponentu dodati u HTML:

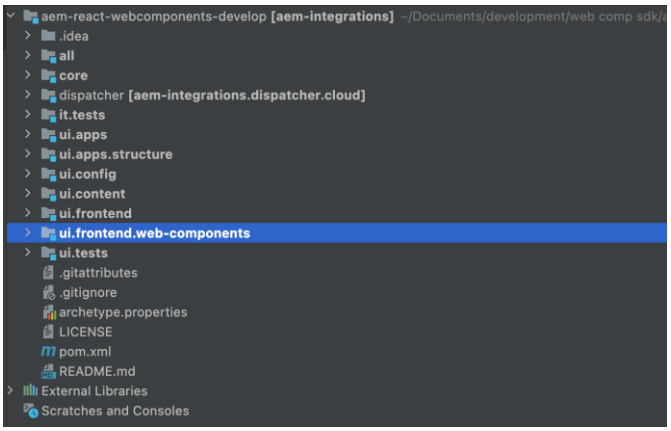
```
<web-komponenta svojstvo="Web komponenta"></web-komponenta>
```

Slika 35. Prikaz web komponente i njezin HTML sadržaj (autorski rad)

Na slici 35. vidimo da je prikazan samo element <web-komponenta> dok je njezin sadržaj izoliran unutar *shadow DOM-a*.

5.3.3.2. Postavljanje projekta

Za postavljanje projekta prilikom ovog pristupa također trebamo postaviti opciju *frontendModule* kako bi se generirao posebno odvojen *ui.frontend* modul. U prijašnjem SPA pristupu objašnjena je važnost ovog modula za izvedbu SPA aplikacije i samih *React* komponenata. Kako bi izbjegli konflikte i dodatno odvojili web komponente preporuka je kreirati još jedan *frontend* modul za web komponente [30]. Najlakši način za kreiranje dodatnog modula je kopiranje postojećeg *ui.frontend* modula te preimenovanje istog. Na sljedećoj slici možemo vidjeti strukturu projekta:



Slika 36. Prikaz strukture projekta (autorski rad)

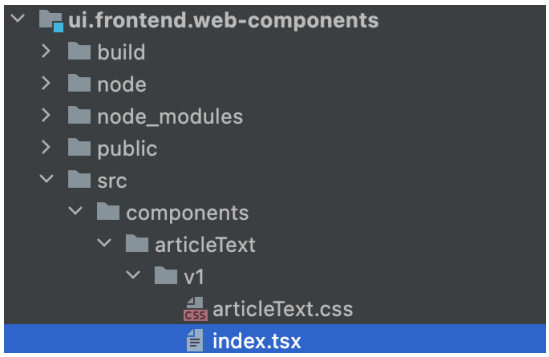
Nakon kopiranja modula posebno treba obratiti pažnju na *pom.xml* datoteku novokreiranog modula gdje treba promijeniti informacije o roditeljskom projektu, a u roditeljskom projektu u istoimenoj datoteci treba uključiti novonastali direktorij kao modul.

Sljedeći korak je podešavanje postavki već spomenutog *AEM clientlib generatora.* Unutar *clientlib.config.js* potrebno je promijeniti ime i kategoriju *clientlib* direktorija koji će se generirati unutar *ui.apps*, a sadrži minificirane JS i CSS datoteke. Korištenjem generiranih *clientlibova* omogućeno je dodavanje elemenata web komponenti iz *ui.frontend.web-components* modula. Kako prednosti SPA pristupa ne bi utjecale na ovaj pristup, maknute su ovisnosti za SPA renderiranje.

5.3.3.3. Komponente

Kod pristupa korištenjem web komponenti definicija komponente, dijalog i HTML datoteka smješteni su unutar *ui.apps* modula. U ovom slučaju zadaća HTML datoteke je korištenje Sling modela, prikazivanje rezerviranog mjesta ukoliko je potrebno te korištenje prilagođenog elementa određene web komponente sa slanjem varijabli po potrebi.

Same web komponente implementirane su unutar *ui.frontend.web-components* modula koji je kopiran prilikom postavljanja projekta. Komponente se smještaju unutar *src/components* direktorija. Nakon kreiranja direktorija za web komponentu potrebno je kreirati *index.tsx* datoteku. Radi se o *TypeScript* datoteci u kojoj se web komponente pišu koristeći JavaScript XML (JSX). Na slici 37. vidimo strukturu web komponenti.



Slika 37. Struktura web komponenti (autorski rad)

Sadržaj *index.tsx* datoteke prikazan je u nastavku.

```
import React, {Component} from 'react';
// @ts-ignore
import {byAttrVal, createCustomElement, DOMModel} from "@adobe/react-webcomponent";
import MetaUtils from '../../../utils/MetaUtils';
require('./articleText.css')

class ArticleTextModel extends DOMModel {
  @byAttrVal() text: string = '';
  isInEditor = MetaUtils.isInEditor();
  hidePlaceHolder = false;
}
class ArticleTextComponent extends Component<ArticleTextModel> {
  render() {
    return (
        <div>
          <h2 className="article-text">{this.props.text}</h2>
        </div>
    )
  }
}
const ArticleTextElement = createCustomElement(ArticleTextComponent, ArticleTextModel, 'element');
// @ts-ignore
window.customElements.define('article-text', ArticleTextElement);
```

Nakon uključivanja potrebnih biblioteka primjećujemo *ArticleTextModel* klasu koja proširuje *DOMModel.* Tah model definira kako će *DOM* biti parsiran u *React* svojstva, tj. *props.* U ovom slučaju anotacija @byAttrVal() parsira *text* varijablu tipa String koja je unutar

prilagođenog elementa poslana kao atribut. Osim *byAttrVal* postoje anotacije postoje još i byBooleanAttrVal, byJsonAttrVal, byContentVal, byContent, byModel, byChildContentVal…

Nakon toga implementirana je klasa *ArticleTextComponent* koja proširuje *Component*. To je zapravo *React* web komponenta koja renderira prilagođeni element. Unutar *render* metode možemo vidjeti da se pomoću *props* pristupa *text* varijabli. Na kraju se metodom *createCustomElement* kreira prilagođeni element, a *define* metodom registrira se web komponenta na naziv prilagođenog elementa.

Nakon implementacije web komponente istu je potrebno uključiti unutar *index.ts* datoteke:

```
import './components/articleText/v1';
```

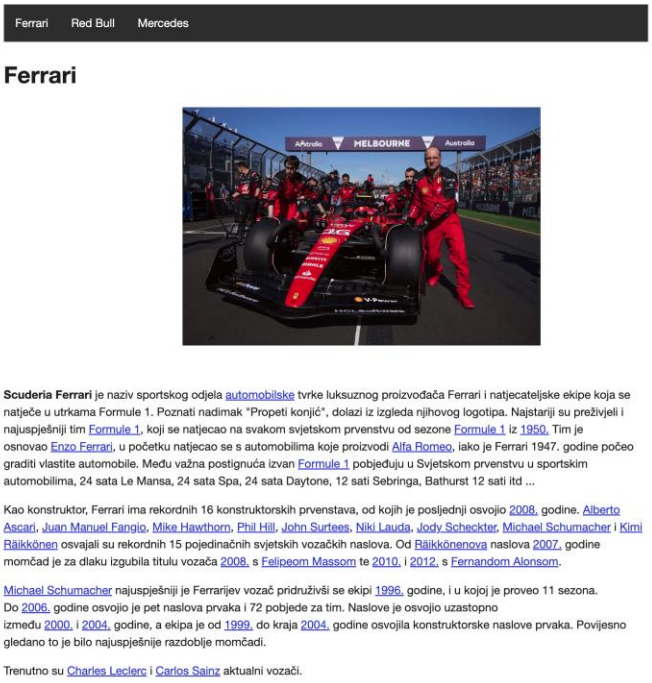
Prilikom izgradnje projekta situacija je jednaka kao i kod SPA pristupa. Iz *ui.frontent.web-components* modula generirat će se *clientlibs* koji se nalazi unutar *ui.apps* modula koji se zatim isporučuje kao paket sadržaja.

5.3.3.4. Sling modeli

Kod Sling modela nema promjena u odnosu na standardni pristup.

6. Usporedba pristupa za izradu web aplikacija

U prijašnjem poglavlju objašnjeni su pristupi za izradu aplikacije u AEM-u. Kako bi usporedili pristupe kreirani su projekti, tj. aplikacije u sva tri pristupa. S obzirom da se uspoređuje sami SPA pristup potrebno je kreirati više stranica kako bi usporedili prelazak s jedne stranice na drugu. Korištene su jednostavne komponente jer se ovaj dio rada ne svodi na implementaciju komponenti već na samu usporedbu pristupa. Komponente koje su koriste u primjeru aplikacija za usporedbu su: navigacija, naslov, slika, tekst članka.



Slika 38. Primjer stranice kreirane za usporedbu pristupa (autorski rad)

Svaki pristup ima svojih prednosti i nedostataka. U nastavku bit će objašnjene prednosti i nedostaci svakog pristupa.

6.1. Standardni pristup

Prednosti standardnog pristupa su sljedeće:

- Poznat je svakom AEM razvojnom inženjeru – prilikom učenja AEM-a koristi se ovaj pristup te ima puno službene dokumentacije
- Odgovara za stranice za puno različitog sadržaja – s obzirom da se ovdje radi o MPA aplikaciji, pogodan je za izradu više različitih stranica te je svaka stranica nova prilika.
- Renderiranje na strani poslužitelja – koristeći HTL koji se prevodi u HTML na stani poslužitelja, HTML datoteka koja se isporučuje jednaka je onoj koja će se prikazati u pregledniku
- Prilagođen za SEO – tražilice obično prolaze svaku stranicu i analiziraju sadržaj kako bi istu rangirale. Ovdje svaka stranica ima HTML koji će biti renderiran što poboljšava vidljivost stranice.
- Stabilan i pouzdan – ovaj pristup je prvi pristup koji se koristio za aplikacije u AEM-u stoga je podržan u svim preglednicima i na svim uređajima

Nedostaci su sljedeći:

- Performanse – prelaskom na sljedeću stranicu preuzima se cijela HTML datoteka sa svim sadržajem iako su promjene možda minimalne. Na taj način obrada zahtjeva troši dosta vremena, tj. dugo se osvježava, što je danas jedan od najvažnijih kriterija web aplikacije.
- Smanjena interaktivnost – moderne web stranice sve više podsjećaju na mobilne aplikacije gdje se sve odvija na jednoj stranici bez osvježavanja. Svako osvježavanje stranice narušava zadovoljstvo korisnika.
- Teži razvoj kompleksnih sučelja – razvojnim inženjerima nisu dostupne moderne tehnologije za razvoj bogatih sučelja poput *Reacta, Angulara* ili *Vue*.

6.2. SPA pristup

Prednosti SPA pristupa su sljedeće:

- Odvajanje frontalnog i pozadinskog razvoja – u ovom pristupu značajno je odvojen frontalni i pozadinski razvoj. Sling modeli izvoze se u JSON formatu čime se princip svodi na API (eng. Application Programming Interface).
- Performanse – kod SPA pristupa sve se odvija na jednoj stranici. Prilikom prvog učitavanja stranice dohvaćen je JSON model s informacijama o stranicama i komponentama te sve potrebne klijentske datoteke. Prilikom prelaska na sljedeću stranicu nema osvježavanja ni preuzimanja podataka.
- Bogata korisnička sučelja – razvojnim inženjerima su dostupni razvojni okviri za implementaciju kompleksnih sučelja (*React, Angular, Vue*)
- Dostupnost *React* komponenti – uključivanjem određenih biblioteka moguće je koristiti već gotove *React* komponente

Nedostaci su sljedeći:

- Inicijalno učitavanje – inicijalno učitavanje je sporije jer se preuzima model sa svim stranicama
- Nije optimizirano za SEO – tražilice imaju problema s indeksiranjem dinamički renderiranog sadržaja
- Kompleksnost i održavanje – rastom aplikacije raste i sama kompleksnost implementacije
- Kompatibilnost s preglednikom – kod ovog pristupa sadržaj se renderira na strani klijenta. Stariji preglednici možda nemaju funkcionalnost koja je potrebna za renderiranje naprednih komponenti.

- Renderiranje nije instantno – sadržaj se renderira procesiranjem JSON modela što traje nekoliko milisekundi

6.3. Pristup korištenjem web komponenti

Prednosti ovog pristupa su sljedeće:

- Enkapsulacija – sadržaj web komponenti izdvojen je u posebni *DOM* i potpuno je neovisan
- Interoperabilnost – web komponente funkcioniraju odvojeno bez korištenja razvojnih okvira
- Ponovna upotrebljivost – web komponente su u potpunosti namijenjene ponovnoj upotrebi. Korištenje je vrlo jednostavno. Dodaje se prilagođeni element komponente u HTML.
- Odvojenost frontalnog i pozadinskog razvoja – Sling modeli koriste se kako bi se postavili atributi web komponente ali razvoj samih web komponenata je odvojen od pozadinskog dijela aplikacije
- Dostupnost gotovih web komponenata – gotove komponente dodaju se korištenjem prilagođenog elementa web komponente. Npr. Coral Spectrum web komponente.

Nedostaci su sljedeći:

- Renderiranje nije instantno – web komponente zahtijevaju nekoliko milisekundi kako bi se učitale, procesirale, registrirale te renderirale. U tom trenutku javlja se tzv. problem bijelog zaslona.
- Nije optimizirano za SEO – tražilice imaju problema s indeksiranjem dinamički renderiranog sadržaja
- Enkapsulacija može biti nedostatak – nije moguće dijeliti stilove između komponenti jer svaka komponenta zahtijeva svoj stil u odvojenom *DOM*-u
- Jako malo dostupne dokumentacije o arhitekturi AEM projekta za web komponente
- Kompatibilnost s preglednikom – kod ovog pristupa sadržaj se renderira na strani klijenta. Stariji preglednici možda nemaju funkcionalnost koja je potrebna za renderiranje naprednih komponenti.

6.4. Kriteriji usporedbe

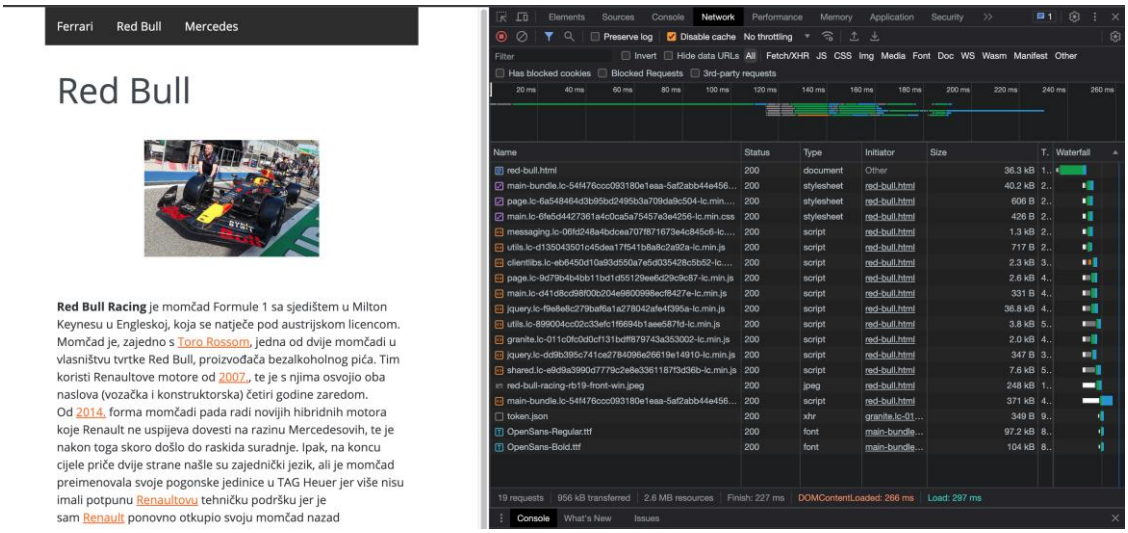
U ovom poglavlju direktno će se usporediti pristupi izrade aplikacije po određenim kriterijima. Kriteriji su: performanse, renderiranje, arhitektura, kompleksnost i brzina implementacije, ponovna upotrebljivost komponenti, odvojenost komponenti i dostupnost dokumentacije i primjera.

6.4.1. Performanse

Kod ovog kriterija promatra se vrijeme osvježavanja stranica. Kako bi što objektivnije prikazali rezultate, keširanje je isključeno. Usporedba se provodi promatrajući i analizirajući *Network* karticu unutar razvojnih alata preglednika.

Kod **standardnog pristupa** prolaskom kroz različite stranice dolazi do osvježivanja te se preuzima cijela HTML datoteka s web sadržajem.

Učitavanje svake stranice kod ovog pristupa traje 300 do 400 milisekundi:



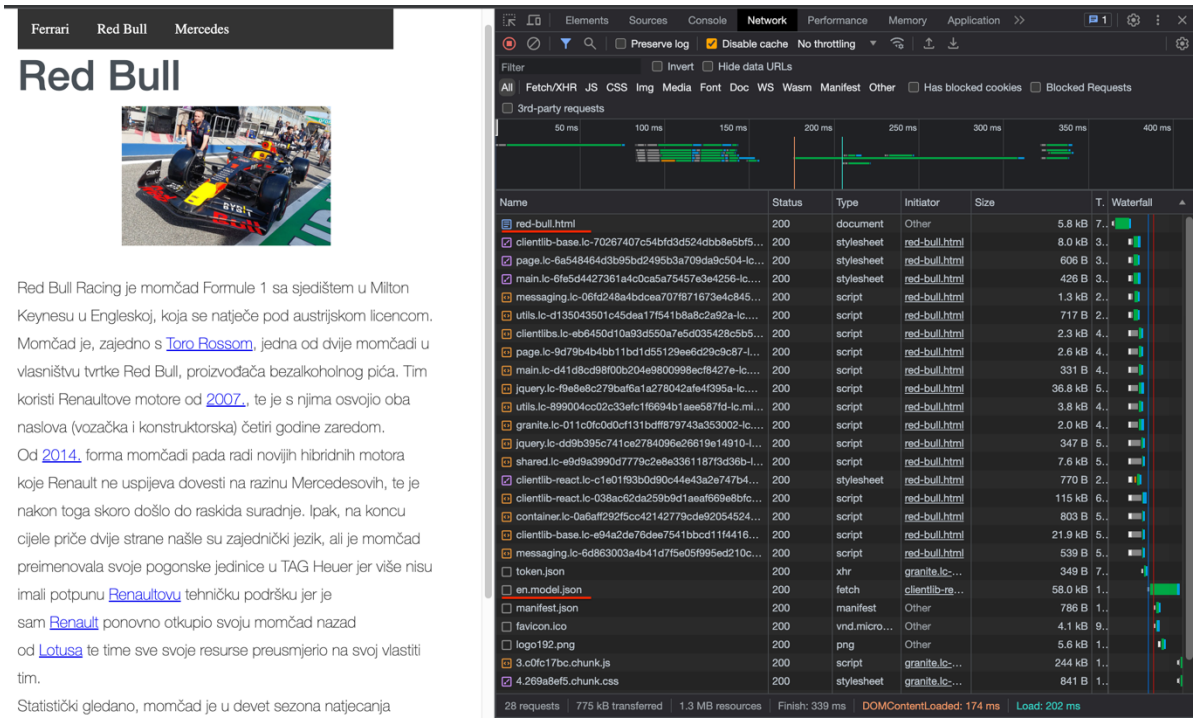
Slika 39. Osvježavanje stranice kod standardnog pristupa (autorski rad)

Kod **SPA pristupa** prilikom prvog učitavanja učita se HTML datoteka, sve klijentske datoteke i *en.model.json* koji sadrži podatke o sadržaju svih dostupnih stranica. Ovi podaci preuzimaju se samo jednom pa je inicijalno učitavanje jedino koje se provodi i traje oko 200 milisekundi.

HTML sadrži element s identifikatorom „spa-root“ i u njemu se prikazuje i izmjenjuje sadržaj svih dostupnih stranica. Korištenjem spomenute JSON datoteke nije više potrebno preuzimati sadržaj jer ona sadrži sve što je potrebno za prikaz dostupnih stranica.

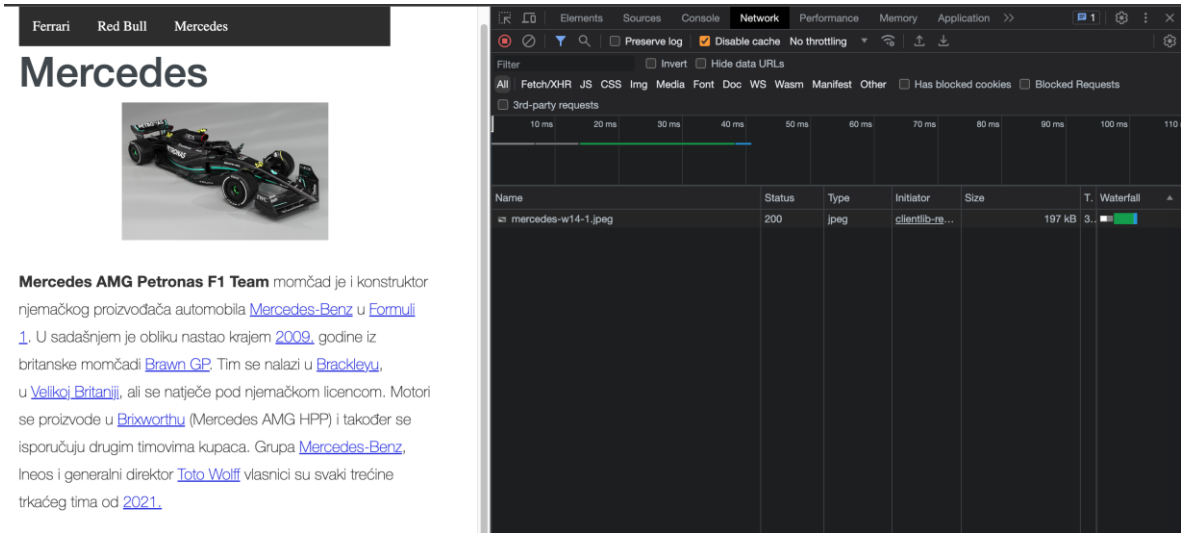
Prelaskom na sljedeću stranicu nema osvježavanja, sadržaj se mijenja po potrebi na trenutnoj stranici.

Analizom *Network* kartice vidljivo je da zahtjev za stranicom traje 30 milisekundi zbog preuzimanja slike. Ukoliko se uključi keširanje ne postoji nijedan zahtjev prilikom posjete ostalih stranica. Na sljedećoj slici prikazano je inicijalno učitavanje stranice.



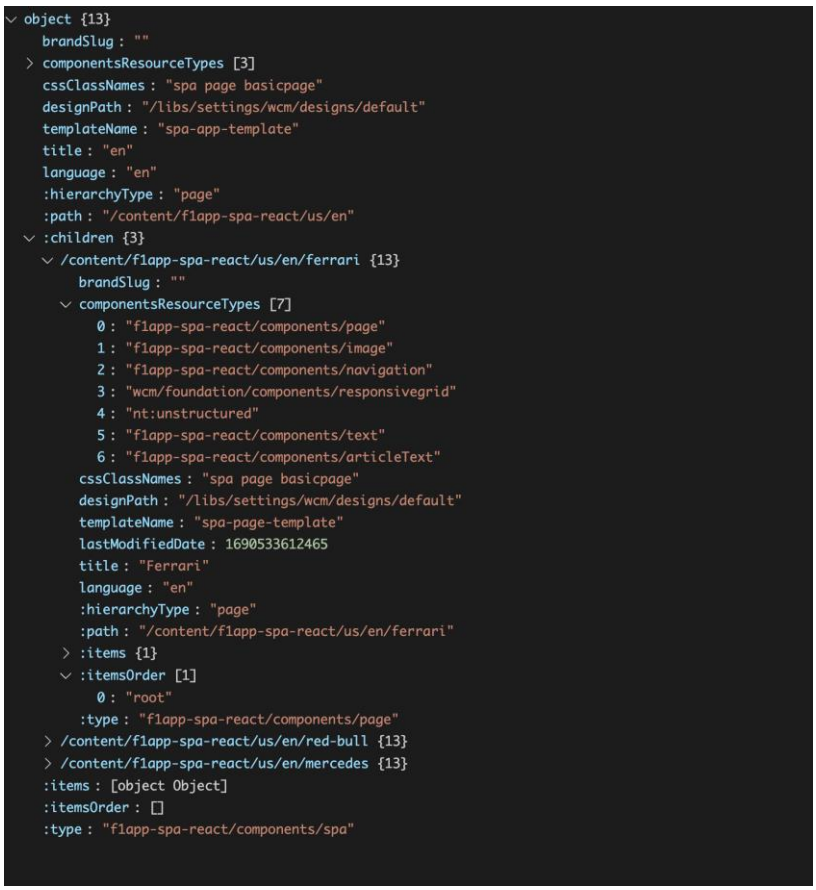
Slika 40. Inicijalno učitavanje kod SPA pristupa (autorski rad)

Na sljedećoj slici prikazano je učitavanje druge stranice:



Slika 41. Učitavanje druge stranice kod SPA pristupa (autorski rad)

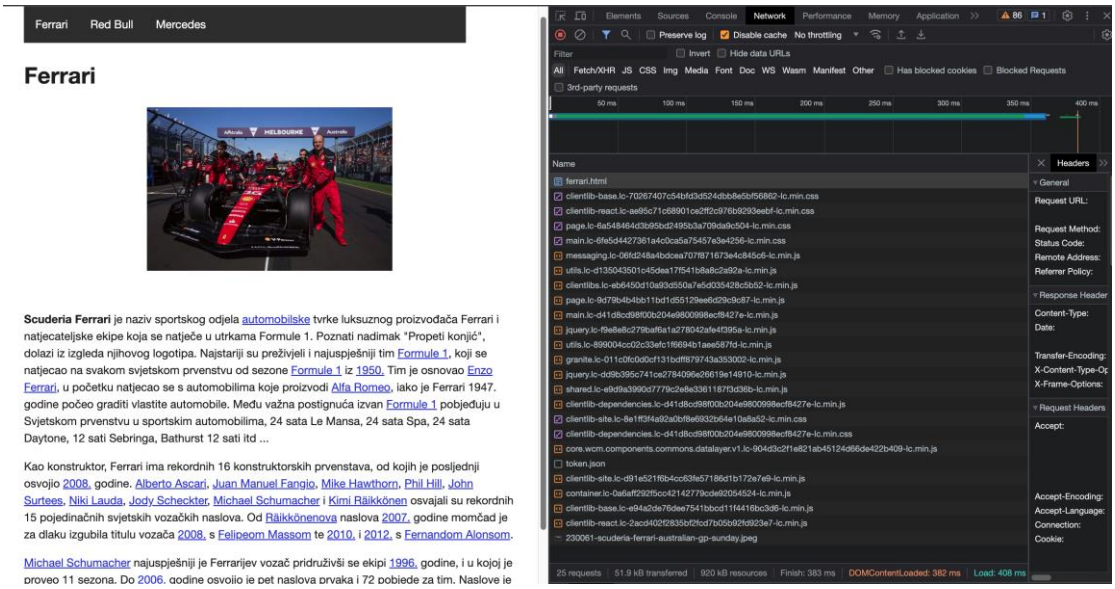
Radi jednostavnosti prikaza sadržaj *en.model.json* datoteke prikazan je na sljedećoj slici:



Slika 42. Sadržaj *en.model.json* datoteke (autorski rad)

Možemo primijetiti da se stranice nalaze unutar :*children* objekta. U postavkama predloška moguće je podesiti razinu stranica koje će biti dostupne u prvom modelu. Ukoliko imamo 3 razine navigacije, a u trećoj razini nalazi se veliki broj stranica, preporuka je uključiti samo prve dvije razine u početni model kako inicijalno učitavanje ne bi predugo trajalo. Dolaskom na stranicu treće razine preuzet će se nova *en.model.json* datoteka koja sadrži informacije stranice treće razine.

Kod **pristupa korištenjem web komponenti** možemo primijetiti najlošije performanse zbog vremena koje je potrebno da se web komponente učitale, procesirale, registrirale i renderirale. Kod ovog pristupa učitavanje pojedine stranice uvelike varira, a traje 300 do 450 milisekundi:



Slika 43. Učitavanje stranice s web komponentama (autorski rad)

6.4.2. Renderiranje

Kod **standardnog pristupa** renderiranje se provodi na strani poslužitelja pa je sadržaj HTML datoteke jednak na poslužitelju i u web pregledniku klijenta. To omogućuje instantno renderiranje.

Kod **SPA pristupa** renderiranje se provodi na strani korisnika pa sadržaj HTML datoteka na poslužitelju ne odgovara sadržaju koji se prikazuje u pregledniku. JavaScript treba procesirati *React* komponente kako bi se one renderirale.

Kod **pristupa korištenjem web komponenti** renderiranje se također provodi na strani korisnika te je potreban JavaScript koji procesira, registrira i renderira web komponente.

Za ovaj kriterij teško je odlučiti rezultat jer obje vrste renderiranja imaju svoje prednosti i nedostatke. Renderiranje na strani poslužitelja može se pohvaliti instantnim renderiranjem, međutim to otežava i troši resurse poslužitelja. S druge strane renderiranje na strani korisnika je brzo i ne opterećuje poslužitelj, ali je potreban JavaScript i ostale funkcionalnosti web preglednika te postoji opasnost problema bijelog ekrana, tj. moguće je da komponenta neće biti na stranici nekoliko milisekundi ili neće imati stilove.

6.4.3. Arhitektura

Kod standardnog pristupa pozadinski razvoj implementira se unutar *ui.apps* modula. Komponente se nalaze unutar *ui.apps* modula gdje se smještaju definicija, dijalog, HTML i

klijentske datoteke (CSS, JS). Komponente su sa Sling modelima povezane koristeći *data-sly-use* čime možemo pristupati varijablama Sling modela.

Kod SPA pristupa primjećuje se veliki napredak u smislu arhitekture. Frontalni razvoj komponenata odvojen je u poseban *ui.frontend* modul gdje se koriste napredni razvojni okviri. Sling modeli anotirani su kao *Exporteri*, tj. izvoze svoje podatke u JSON obliku poput API-ja. Komponente traže JSON sa svojim tipom resursa te preuzimaju potrebne podatke iz JSON-a.

Kod pristupa web komponenata nema definirane standardne arhitekture te ne postoji arhetip za generiranje projekta koji bi služio kao početna točka razvoja aplikacije u ovom pristupu. Web komponente implementiraju se u odvojenom modulu. Tamo se i definiraju prilagođeni elementi koji se zatim pozivaju u HTML datoteci smještenoj unutar *ui.apps* modula. Sling modeli povezuju se na isti način kao kod standardnog pristupa.

6.4.4. Kompleksnost i brzina implementacije

Standardni pristup poznat je gotovo svim razvojnim inženjerima te je prvi pristup koji se koristio za izradu aplikacija u AEM-u. Prilikom generiranja projekta postoji arhetip s primjerima komponenti što uvelike olakšava učenje AEM tehnologija.

Za SPA pristup također postoji arhetip s primjerom komponenti. Razvojni inženjeri frontalnog dijela aplikacije preferiraju ovaj pristup jer im omogućava brzi razvoj robusnih i bogatih sučelja.

Pristup korištenjem web komponenti nije baš zastupljen u svijetu AEM-a. Potrebno je izvesti modifikacije na projektu opisane u prethodnom poglavlju.

6.4.5. Ponovna upotrebljivost komponenti

Komponente kod standardnog pristupa mogu se ponovo upotrijebiti u drugom projektu, međutim potrebno je dosta modifikacije. Postoji opasnost od kolizije stilova i međusobnog narušavanja.

Komponente implementirane u SPA pristupu imaju veću mogućnost ponovne upotrebe. Postoje razne biblioteke koje sadrže već gotove komponente.

Web komponente imaju najviše prednosti s obzirom na ovaj kriterij jer je upravo to svrha web komponente. Web komponente dolaze sa svojim stilovima i skriptama te su potpuno izolirane i neovisne.

6.4.6. Odvojenost komponenti

Kod standardnog pristupa CSS komponenti generira se u minificirani CSS te je dostupan globalno. Samim time postoji velika mogućnost kolizije stilova, tj. stil komponente može narušavati stilove projekta ili obrnuto. Komponente su renderirane u globalnom *DOM*-u

Kod SPA pristupa situacija je slična kao kod standardnog, tj. postoji mogućnost kolizije stilova, a renderiranjem na strani korisnika HTML komponente je također u globalnom *DOM*-u.

Kod pristupa korištenjem web komponenti komponente su u potpunosti odvojene, tj. izolirane u *shadow DOM*-u. Samim time svaka komponenta ima svoj stil i nastanak kolizije je nemoguć.

6.4.7. Dostupnost dokumentacije i primjera

Kod standardnog pristupa dostupno je pregršt dokumentacije, primjera, videa, foruma i slično.

Za SPA pristup količina dokumentacije je vidno smanjena, međutim Adobe nudi primjer implementacije osnovne stranice i komponente što olakšava savladavanje tehnologija koje se koriste kod ovoga pristupa.

Za pristup korištenjem web komponenata gotovo da i ne postoji službena dokumentacija, pa čak niti arhetip.

6.5. Rezultati usporedbe

S obzirom da su istražene i objašnjene implementacije koristeći tri pristupa, moguće je iste direktno usporediti. Za svaki kriterij bit će određeno koji pristup je najbolji, srednji i najgori. Dodjeljivanjem 3 boda za „najbolji“, 2 za „srednji“ te 1 za „najgori“ bit ćemo u mogućnosti izračunati zbroj bodova te tako odrediti koji pristup ima najviše prednosti.

Valja naglasiti da ovim rezultatima ne možemo odrediti koji pristup je najbolji jer odabir pristupa ovisi od projekta do projekta. Ukoliko se radi na implementaciji jednostavnog projekta sa više stranica, bez previše interakcija i zahtjevnih sučelja, nije potrebno implementirati aplikaciju SPA pristupom.

Kriterij	Standardni pristup	SPA pristup	Pristup korištenjem web komponenti
Performanse	Dobre	Odlične	Dobre
Renderiranje	Na strani poslužitelja	Na strani korisnika	Na strani korisnika
Arhitektura	Ne postoji odvojenost pozadinskog i frontalnog dijela aplikacije	Dobro odvojeni pozadinski i frontalni dio	Slaba odvojenost pozadinskog i frontalnog dijela
Kompleksnost i brzina razvoja	Jednostavno i brzo	Jednostavno i brzo	Kompleksnije postavljanje projekta, brz razvoj
Ponovna upotrebljivost komponenti	Slaba	Umjerena	Odlična
Odvojenost komponenti	Slaba	Umjerena	Odlična
Dostupnost dokumentacije	Postoji puno dokumentacije i primjera	Postoji dovoljno dokumentacije i primjera	Gotovo da ne postoji dokumentacije i primjera

Tablica 7. Usporedba pristupa (autorski rad)

Prije spomenutim bodovanjem standardni pristup osvojio bi 13 bodova, SPA pristup 17 bodova, a pristup korištenjem web komponenti 15 bodova.

7. Praktični dio – F1 App

U ovom poglavlju opisat ću razvoj praktičnog dijela rada, tj. aplikacije u AEM-u. Imajući na umu da je AEM CMS sustav, opisat ću aplikaciju iz dvije perspektive: autorske i korisničke. S autorske strane objasnit ću kako autori mogu dodavati, uređivati i konfigurirati komponente, dok ću s korisničke strane objasniti kako se aplikacija koristi i koje su njezine mogućnosti.

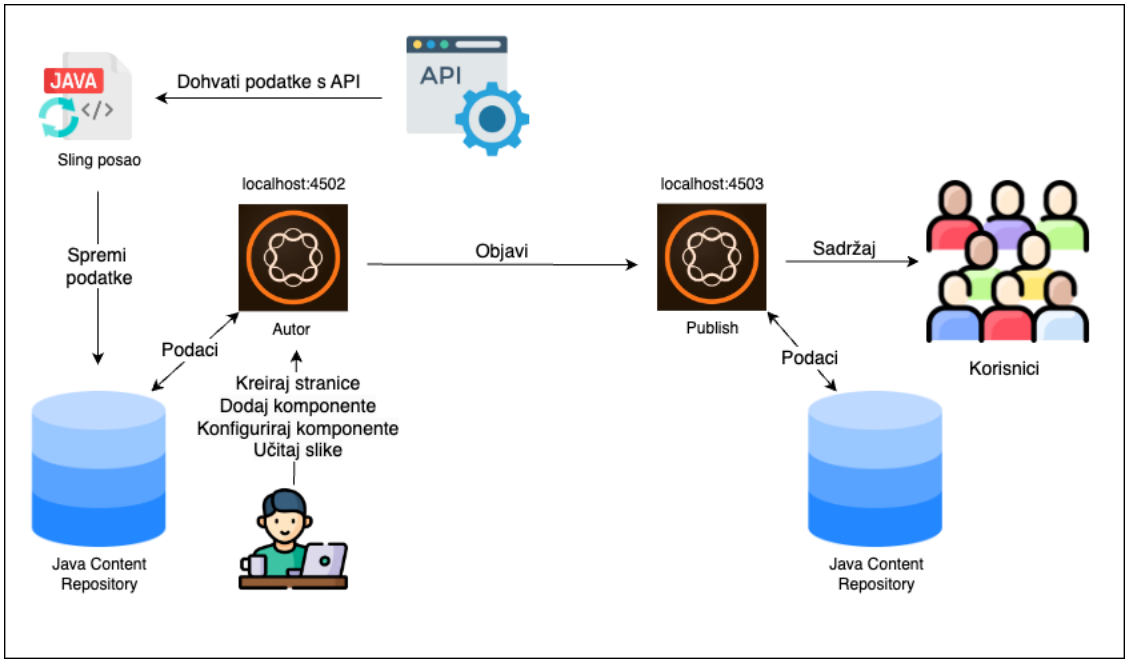
7.1. Opis aplikacije

F1 App je aplikacija namijenjena svim ljubiteljima Formule 1. Koristeći ovu aplikaciju korisnici mogu saznati sve novosti vezane uz navedeni sport, saznati rezultate utrka ili trenutačni poredak, provjeriti raspored nadolazećih utrka i slično. Glavne funkcionalnosti su sljedeće:

- Prijava i registracija – svaki korisnik može kreirati korisnički račun kako bi komentirao vijesti s novostima o vozačima i momčadima ili kako bi spremio omiljene vozače ili timove u favorite
- Pregled novosti – svi korisnici mogu pregledavati novosti iz svijeta Formule 1 te iste filtrirati po vozaču ili momčadi
- Komentiranje – neregistrirani korisnici mogu vidjeti komentare drugih korisnika ispod članaka, registrirani korisnici imaju mogućnost dodavanja komentara, brisanja ili uređivanja vlastitih komentara, dok administrator ima mogućnost brisanja tuđih komentara
- Dodavanje omiljenog vozača i/ili momčadi – korisnik može odabrati vozača ili momčad i dodati ga u favorite. Informacije o spremljenim vozačima spremaju se u memoriju preglednika kako bi se zapamtio odabir.
- Popis preporučenih članaka – na temelju spremljenih favorita u memoriji preglednika korisniku će se na naslovnici prikazati nekoliko novosti povezanih s vozačima tj. momčadima koji su odabrani kao favoriti. Autor ima mogućnost odabira maksimalnog broja novosti koje će biti prikazane.
- Prikaz rasporeda utrka – autor može birati koliko utrka unazad i koliko utrka unaprijed će biti prikazano. Također ima mogućnost prikazivanja termina slobodnih treninga. Korisnici mogu preuzeti iCalendar (.ics) datoteku kako bi utrku direktno dodali u vlastiti kalendar.
- Prikaz broja bodova pojedinog vozača ili momčadi
- Prikaz rezultata utrke

Podaci će se preuzimati koristeći API za Formulu 1 [32]. Kako bi se smanjio broj poziva prema API i tako ubrzao rad aplikacije, podaci će biti spremljeni u JCR repozitorij te će se implementirati servis koji će periodički provjeravati jesu li podaci ažurni.

Za rad aplikacije koriste se već spomenute osnovne dvije instance AEM-a: autor i publish. Na autor instanci autori kreiraju sadržaj te ga objavljuju na publish koji isti sadržaj servira krajnjim korisnicima. Svi podaci spremaju se u JCR.



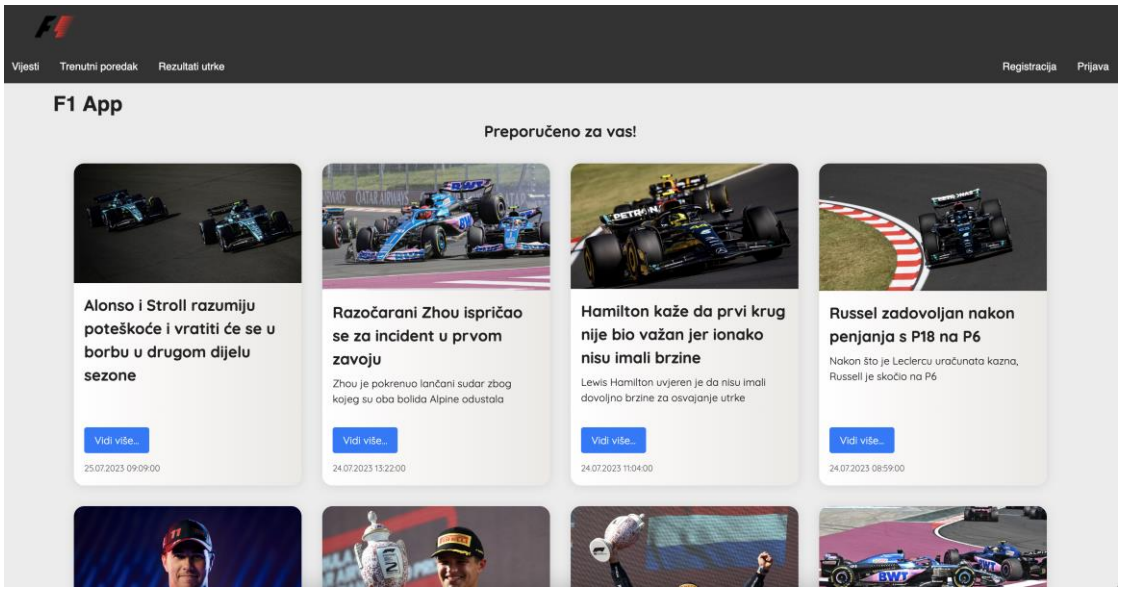
Slika 44. Arhitektura aplikacije s najviše razine (autorski rad)

7.2. Funkcionalnosti aplikacije

U ovom poglavlju bit će objašnjena svaka funkcionalnosti. Za svaku funkcionalnost prikazat ću zaslon, tj. kako izgleda korisnički dio aplikacije, objasniti pozadinsku logiku koja omogućuje funkcionalnost te izdvojiti bitnije dijelove kôda.

7.2.1. Osnovni izgled aplikacije

Na slici 45. prikazana je naslovnica F1 App aplikacije. Možemo primijetiti da je aplikacija građena koristeći komponente. Strukturalne komponente koje se koriste u aplikaciji su sljedeće: zaglavlje, logo navigacija i podnožje.



Slika 45. Naslovnica F1 App (autorski rad)

Jedina strukturalna komponenta koja sadrži pozadinsku logiku je navigacija. Kako bi kreirali linkove za navigaciju, dohvaća se početni resurs, a to je naslovna stranica aplikacije. Da bi dohvatili stranice niže razine koje su potrebne u navigaciji potrebno je resurs adaptirati na *Page*. Kako bi dobili listu svih stranica niže razine koristi se metoda *listChildren*. Za kreiranje objekta navigacije korištena je klasa *NavigationItem* koja sadrži listu navigacije druge razine ispod trenutne stranice te ime i url trenutne stranice.

U nastavku su prikazani važni dijelovi klase *NavigationModel* koja je zapravo Sling model komponente *custom-navigation*.

```
@Slf4j
@Model(adaptables = SlingHttpServletRequest.class, adapters =
Navigation.class,
  resourceType = NavigationModel.RESOURCE_TYPE, defaultInjectionStrategy =
DefaultInjectionStrategy.OPTIONAL)
public class NavigationModel implements Navigation {


    static final String RESOURCE_TYPE = "f1app/components/custom-
navigation/";
    static final String NAVIGATION_ROOT = "/content/f1app/us/en/home";
    static final String ANONYMOUS_USER = "anonymous";


    @SlingObject
    private ResourceResolver resourceResolver;


    @SlingObject
```

```
    private SlingHttpServletRequest request;

    @ScriptVariable
    private Page currentPage;

    @OSGiService
    private LinkService linkService;
```

U klasi Sling modela možemo primijetiti anotacije koje se koriste za injektiranje atributa resursa u varijable. Antotacija *@SlingObject* injektira objekte koji se mogu dohvatiti iz *SlingHttpServletRequesta, ResourceResolvera* ili *Resource. @ScriptVariable* također injektira objekte, a najčešći su trenutna stranica, trenutni dizajn, trenutni čvor, kontekst i slično. Anotacijom *@OSGiService* injektira se određeni OSGi servis.

```
    @PostConstruct
    public void init() {
        try {
            final Resource rootResource =
this.resourceResolver.getResource(NAVIGATION_ROOT);
            this.homePage = rootResource != null ?
rootResource.adaptTo(Page.class)
                                :
this.resourceResolver.getResource("/content/f1app").adaptTo(Page.class);
            this.navigationItems = this.createNavigation(this.homePage);
            this.userName = this.findUserName();
            this.loggedIn = !StringUtils.equals(ANONYMOUS_USER,
this.userName);
        } catch (final RuntimeException e) {
            log.error("Exception in post construct", e);
        }
    }
```

Anotacija *PostConstruct* označava da će se metoda izvršiti odmah prilikom inicijalizacije Sling modela. Unutar metode *init* kreiraju se već spomenuti elementi navigacije:

```
private List<NavigationItem> createNavigation(final Page homePage) {
        return IteratorUtils.toList(this.homePage.listChildren())
                            .stream()
                            .filter(this::excludeSpecificPages)
                            .map(this::createNavigationItem)
                            .collect(Collectors.toList());
    }
```

U nastavku je prikaz klase *NavigationItem* koja je anotirana sa *@Data* čime se korištenjem *Lombok* biblioteke dobivaju svi potrebni Getteri i Setteri te *@Builder* kako bi iskoristili istoimeni uzorak dizajna za mogućnost kreiranja objekta uz samo određene atribute.

```
@Data
@Builder
public class NavigationItem {

    @Builder.Default
    private List<NavigationItem> children = new ArrayList<>();
    private String name;
    private String url;
}
```

Ostala funkcionalnost navigacije vezana je za prijavu i registraciju pa će biti objašnjena u sljedećem dijelu.

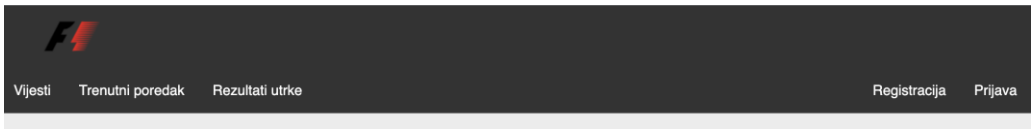
7.2.2. Prijava i registracija

Desnu stranu navigacije potrebno je mijenjati ovisno trenutnom korisniku. U modelu navigacije metodom *findUsername* provjerava se korisničko ime kako bi saznali da li je korisnik prijavljen. U AEM-u svaka sesija ima prijavljenog korisnika. Ako korisnik zapravo nije prijavljen, unutar sesije se vodi kao „Anonymous“. Radi se o predefiniranom korisničkom profilu koji ima samo osnovna prava čitanja nužnih podataka web stranice kreirane u AEM-u.

Kako bi dohvatili korisničko ime potrebno je *resourceResolver* adaptirati na *Session*. Postojanjem sesije moguće je iskoristiti metodu *getUserID* koja vraća korisničko ime korisnika.

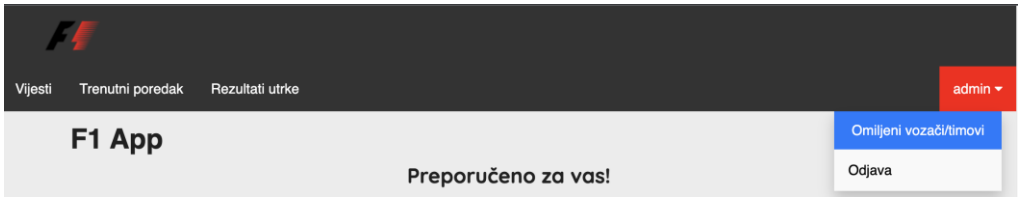
```
private String findUserName() {
    final Session session = this.resourceResolver.adaptTo(Session.class);
    if (session != null) {
        return session.getUserID();
    }
    return StringUtils.EMPTY;
}
```

Ako je traženo korisničko ime „Anonymous“ znači da korisnik nije prijavljen. U tom slučaju s desne strane navigacije dobiva linkove za registraciju, odnosno prijavu:



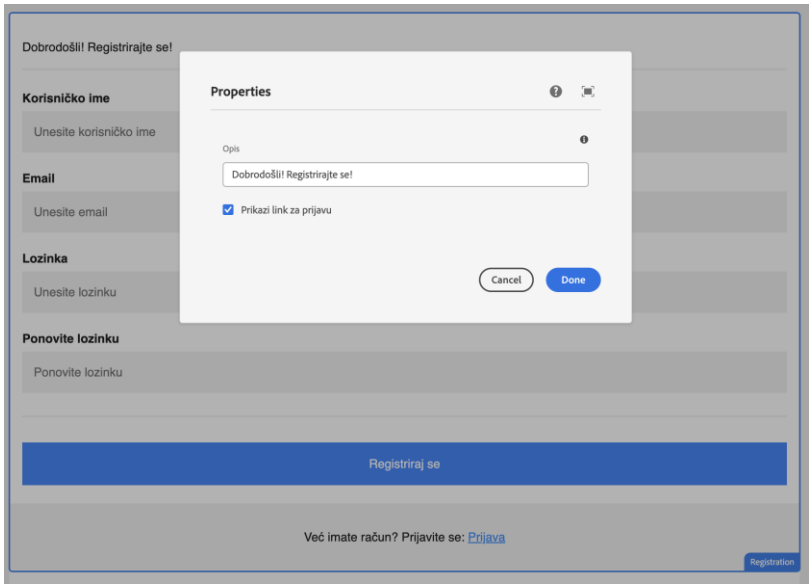
Slika 46. Navigacija kod neregistriranog korisnika (autorski rad)

Ukoliko je dohvaćeno korisničko ime iz sesije i to ime nije „Anonymous“ znači da je korisnik prijavljen. U tom slučaju s desne strane se prikazuje korisničko ime koje otvara izbornik s dodatnim mogućnostima: odabir omiljenih vozača/timova te odjava:



Slika 47. Navigacija kod registriranog korisnika (autorski rad)

Registracija je moguća korištenjem komponente *Registration*. Autori imaju na raspolaganju upisati tekst koji će biti prikazan iznad same forme te mogućnost prikazivanja direktnog linka za prijavu ispod forme:



Slika 48. Konfiguracija komponente *Registration* (autorski rad)

Forma za registraciju sastoji se od polja za unos korisničkog imena, emaila, lozinke i ponovljene lozinke. Validacija svakog polja vrši se na strani korisnika koristeći JavaScript. Nakon što je forma uspješno ispunjena okida se *RegistrationServlet* koji iz zahtjeva dohvaća

parametre u kojima su spremljeni podaci potrebni za registraciju korisnika. Servlet koristi *UserService* servis koji sadrži metodu *createUser*. U nastavku je prikaz *doPost* metode *RegistrationServleta* koji proširuje *SlingAllMethodsServlet*:

```
      @Override
          protected void doPost(final SlingHttpServletRequest request, final
SlingHttpServletResponse response) {
              try {
                  final String username = request.getParameter("username");
                  final String email = request.getParameter("email");
                  final String password = request.getParameter("psw");
                  response.setStatus(this.userService.createUser(username,
email, password));
              } catch (final RuntimeException e) {

response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
                  log.error("Error while creating user", e);
              }
          }
```

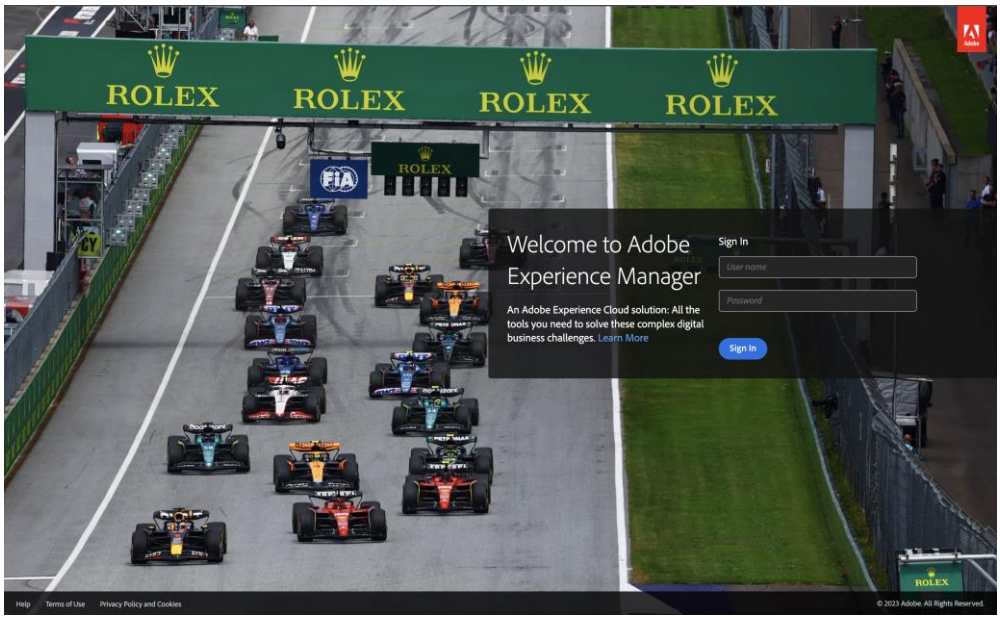
Servis *UserService* sadrži implementaciju metode *createUser* koja zapravo kreira korisnika s podacima iz registracijske forme:

```
    public int createUser(final String username, final String email, final
String password) {
              try (final ResourceResolver resourceResolver =
this.getResourceResolverFromServiceUser()) {
         final UserManager userManager = resourceResolver
                                          .adaptTo(UserManager.class);
         final Session session = resourceResolver.adaptTo(Session.class);
                  if (userManager != null && session != null) {
                      final User user = userManager.createUser(username,
                                                               password);
                      ValueFactory valueFactory = session
                                                  .getValueFactory();
                      final Value emailValue = valueFactory
                                  .createValue(email, PropertyType.STRING);
                      user.setProperty("./profile/email", emailValue);
                      session.save();
```

```
                final Group group = (Group) userManager
                                                .getAuthorizable(GROUP_ID);
                final Authorizable auth = userManager
                                                .getAuthorizable(username);
                if (group != null && auth != null) {
                    group.addMember(auth);
                }
                session.save();
                return 200;
            }
        } catch (final LoginException e) {
            log.error("Error while fetching system user", e);
       }
        return 500;
    }
```

Prvo je potrebno dohvatiti *ResourceResolver* korištenjem sistemskog korisnika koji ima potrebne ovlasti za kreiranje korisničkih računa. Adaptiranjem *ResourceResolvera* dobivamo sesiju i *userManager* koji sadrži ključnu metodu *createUser* kojoj se kao argumenti šalju korisničko ime i lozinka. Korištenjem *valueFactory* možemo dodati dodatne atribute poput emaila. Nakon kreiranja korisnika potrebno ga je dodati u određenu grupu kako bi imao potrebna prava čitanja i pisanja.

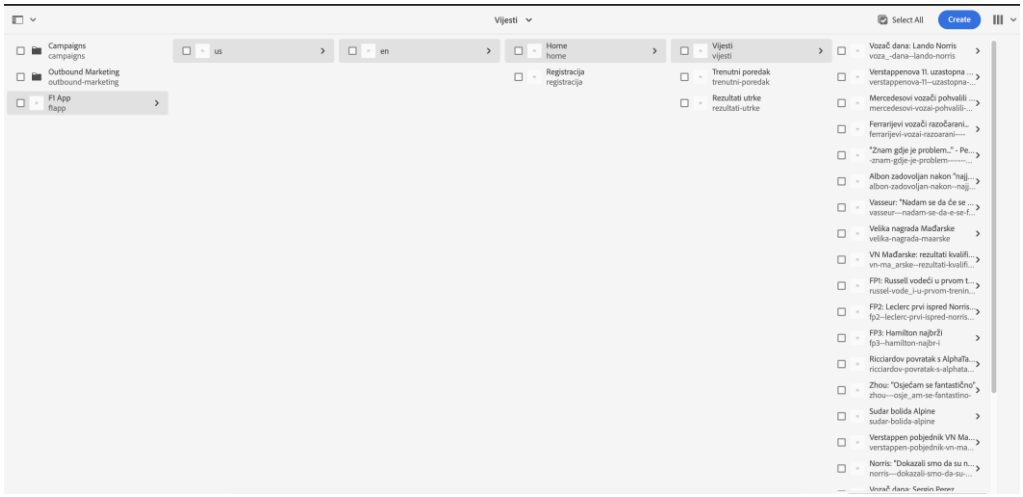
Nakon uspješne registracije korisnik je automatski prebačen na prijavu:



Slika 49. Prijava (autorski rad)

7.2.3. Kreiranje članaka

Autori imaju mogućnost kreiranja članka s novostima. U ovom slučaju stranice sa člancima spremljene su ispod stranice *Vijesti*:

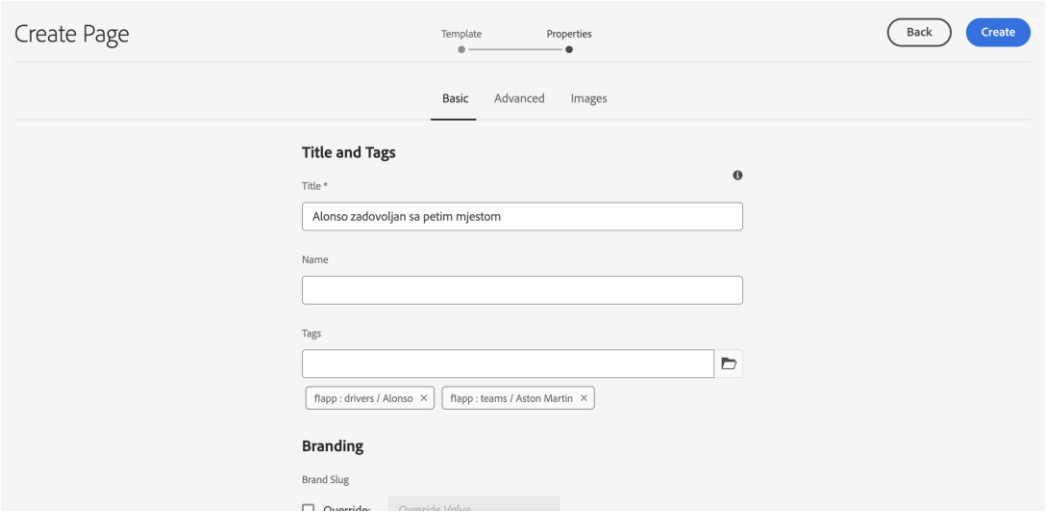


Slika 50. Struktura stranica aplikacije (autorski rad)

Kako bi se kreirala nova stranica potrebno je kliknuti na stranicu *Vijesti* te pritisnuti *Create -> Page*. Otvara se prozor u kojem je potrebno odabrati predložak stranice. Predložak je definicija web stranice u XML formatu, a određuje strukturu stranice i inicijalne podatke komponente. Kreirana su dva predloška, a to su *ArticlePage* i *ContentPage*. *ArticlePage* je predložak namijenjen člancima pa tako na vrhu ima naslov i podnaslov, prikaz vremena

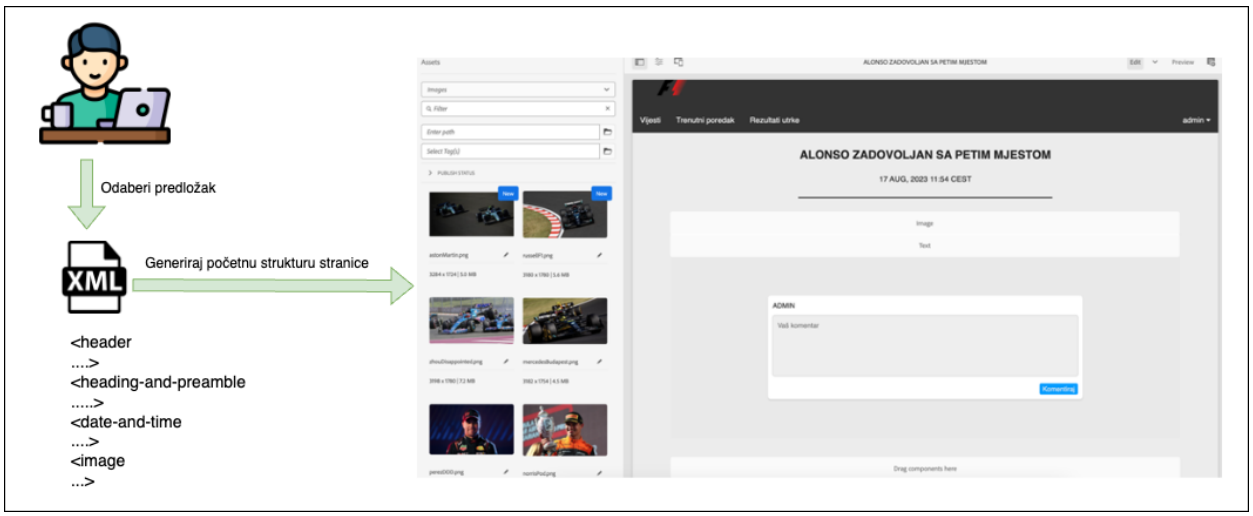
kreiranja, sliku, tekst i komentare. *ContentPage* je općeniti predložak korišten za kreiranje stranica sa sadržajem poput registracije, naslovnice i sl.

Nakon odabira predloška potrebno je unijeti glavne informacije o stranici. Obavezan podatak je naslov, a u ovom slučaju autori trebaju i odabrati oznake. Oznake određuju o kojem se vozaču ili timu radi u članku. Taj podatak je važan kako bi se kasnije stranice mogle filtrirati. Oznake su kreirane automatski prilikom dohvaćanja podataka o vozačima pa će kasnije biti opisano.



Slika 51. Kreiranje stranice s člankom (autorski rad)

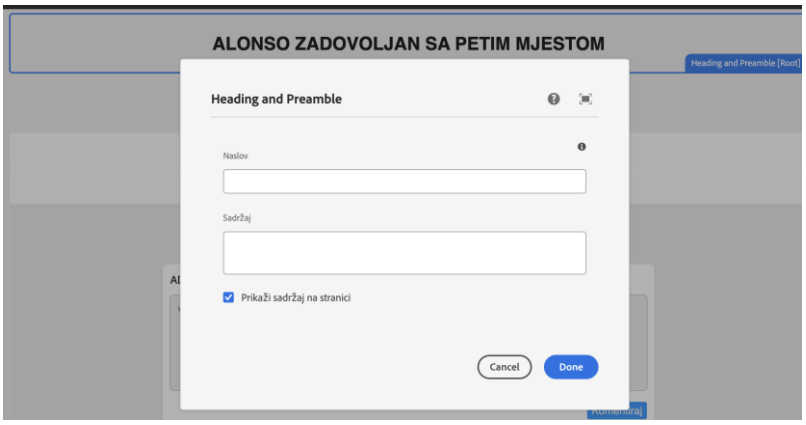
Nakon kreiranja komponente su već dodane na stranicu pravim redoslijedom:



Slika 52. Prazna stranica s člankom (autorski rad)

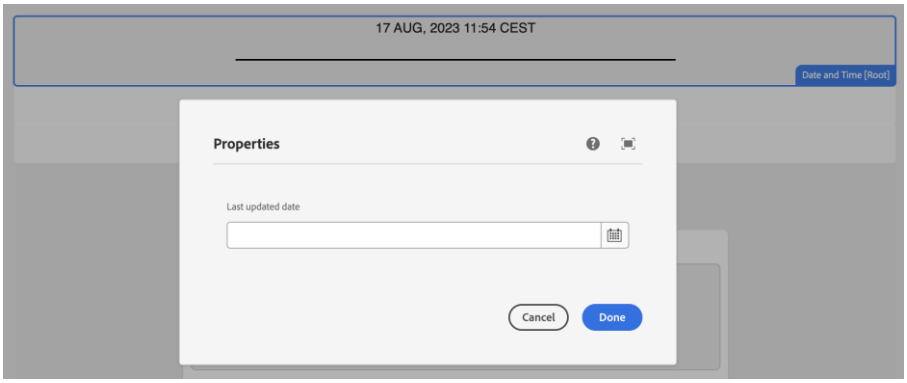
Komponente koje se koriste kod kreiranja stranice s člancima su sljedeće:

- *Heading and preamble* – koristi se kako bi se prikazao naslov i podnaslov (sadržaj) članka. Ukoliko se naslov ne upiše, uzima se naslov stranice. Sadržaj stranice unosi se kako bi se prikazao ispod kartice s vijestima što će biti prikazano kasnije, a može se i prikazati na stranici.



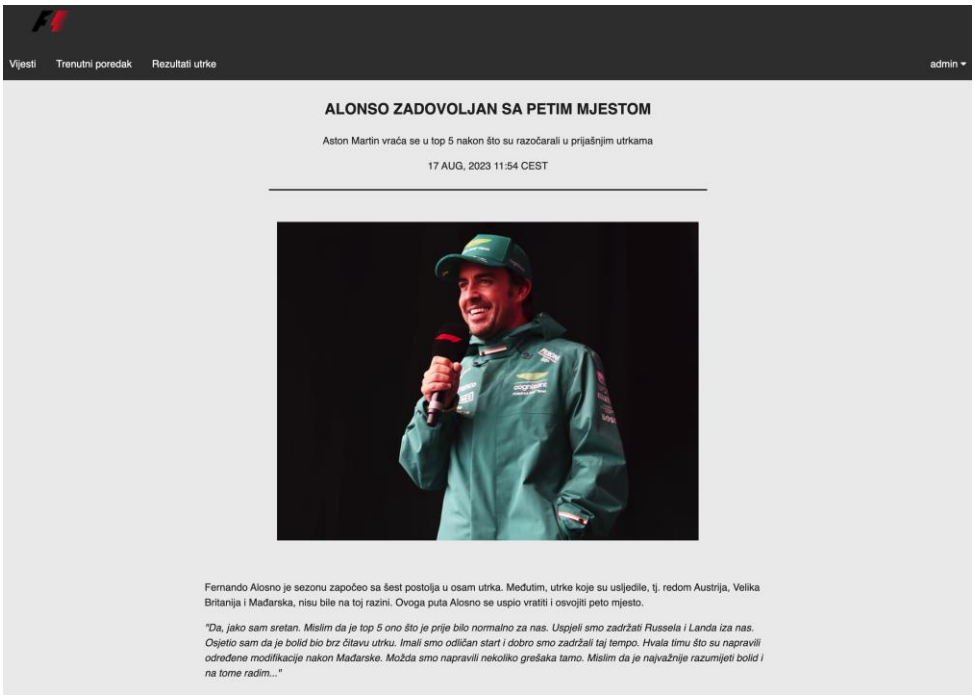
Slika 53. Dijalog komponente *Heading and Preamble* (autorski rad)

- *Date and Time* – koristi se za prikaz vremena objave članka. Ukoliko se ne konfigurira, uzima se vrijeme kreiranja stranice. Podatak ove komponente koristi se za sortiranje vijesti.



Slika 54. Dijalog komponente *Date and Time* (autorski rad)

- *Image* – služi za odabir i prikaz slike
- *Text* – služi za unos teksta članka
- *Comments* – prikazuje komentare, bit će detaljnije objašnjena kod funkcionalnosti komentiranja

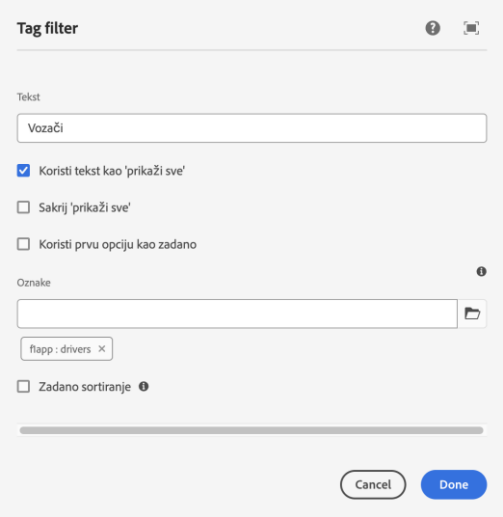


Slika 55. Kreirana stranica sa sadržajem (autorski rad)

7.2.4. Pregled članaka

Kako bi autori konfigurirali stranicu za pregled članaka koriste se sljedeće komponente:

- *Filter container* – strukturalna komponenta koja ne sadrži pozadinsku logiku već samo služi kao kontejner u koji će se stavljati *Tag filter* komponente
- *Tag filter* – komponenta koja omogućuje filtriranje članaka. Autori imaju na raspolaganju dodavati više filtera koje je potrebno konfigurirati.



Slika 56. Dijalog komponente *Tag filter* (autorski rad)

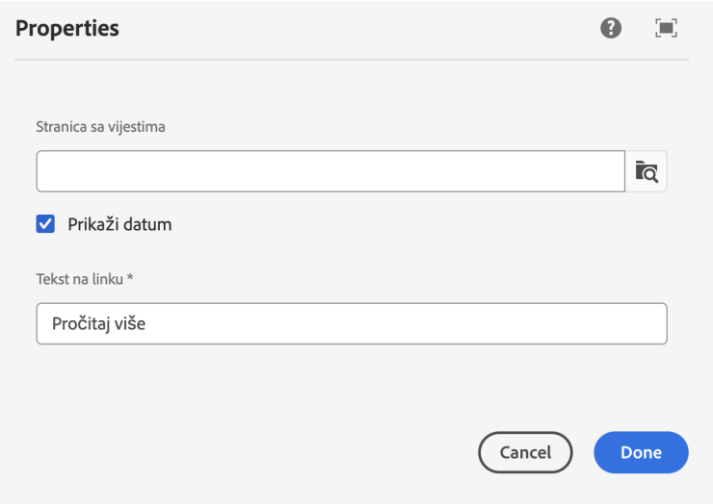
Tag filter dohvaća oznake koje autor odabere u *Oznake* polju.

Za dohvaćanje oznaka zaslužne su dvije metode unutar *TagFilterModel* klase:

```
    private List<Tag> getTagOptions() {
        final TagManager tagManager = this.tagManagerFactory
.getTagManager(this.resourceResolver);
        return this.tags.stream()
                        .map(tagManager::resolve)
                        .flatMap(tag->
this.getTagCollection(tag).stream())
                        .sorted(new TagComparator(this.order))
                        .collect(Collectors.toList());
    }
    private List<Tag> getTagCollection(final Tag tag) {
        return tag.listChildren().hasNext()
                ? IteratorUtils.toList(tag.listChildren())
                : List.of(tag);
    }
```

Ukoliko odabrana oznaka sadrži djecu, dohvaćaju se i sva djeca oznake. Tako u ovom slučaju autor odabire oznaku *drivers*. Prilikom dohvaćanja oznaka metodom *getTagCollection* provjerava se ima li odabrana oznaka djecu. Ako ima, dohvaćaju se sva djeca oznake pa tako dobivamo oznake za pojedine vozače.

- *News list* – prikazuje kartice s člancima. Svaka kartica sastoji se od naslova, slike, sadržaja, datuma objave te gumba koji vodi na stranicu s člankom.



Slika 57. Dijalog komponente *News list* (autorski rad)

Autorima je omogućeno biranje roditeljske stranice s vijestima. Ukoliko se ostavi prazno, podrazumijeva se da je trenutna stranica roditelj svih stranica s vijestima odnosno

člancima. Autori također mogu odlučiti hoće li se prikazivati datum objave te zadati tekst na gumbu koji otvara stranicu s člankom.

```
    private void init() {
            try {
                this.rootPath = StringUtils.isEmpty(this.rootPath) ?
    this.currentPage.getPath() : this.rootPath;
                this.tagsForFiltering=this.request
.getParameterValues("tags[]") == null ? new ArrayList<>()
       : List.of(this.request.getParameterValues("tags[]")).stream()
                                .filter(StringUtils::isNotEmpty)
                                .collect(Collectors.toList());
                final Page rootPage = this.pageManagerFactory
.getPageManager(this.resourceResolver).getPage(this.rootPath);
                if (rootPage != null) {
                    this.pathList = IteratorUtils
                            .toList(rootPage.listChildren())
                            .stream()
                            .filter(this::pageContainsTags)
                            .map(Page::getPath)
                            .collect(Collectors.toList());
                    this.pageListItems = this.pathList.stream()
                    .map(path -> this.resourceResolver.getResource(path))
                    .filter(Objects::nonNull)
                    .map(this::createPageListItem)
                    .filter(Objects::nonNull)
                    .sorted(new PageListItemComparator())
                    .collect(Collectors.toList());
                }
            } catch (final RuntimeException e) {
                log.error("Exception in post construct", e);
            }
        }
```

U prvoj liniji određuje se roditeljska stranica koja sadrži stranice s vijestima. Ukoliko autor nije zadao taj podatak, uzima se trenutna stranica. Nakon toga provjeravaju se parametri zahtjeva. Ako zahtjev ne sadrži *tags* parametar, znači da korisnik nije odabrao filter. Ako zahtjev ipak sadrži parametar *tags*, vrijednosti parametra spremaju se u listu *tagsForFiltering*.

Nakon što se dohvati roditeljska stranica metodom *listChildern* dobivaju se stranice sa člancima. Iste je potrebno filtrirati metodom *pageContainsTags:*

```
    private boolean pageContainsTags(final Page page) {
          return     CollectionUtils.isEmpty(this.tagsForFiltering)     ||
                        Stream.of(page.getTags()
```

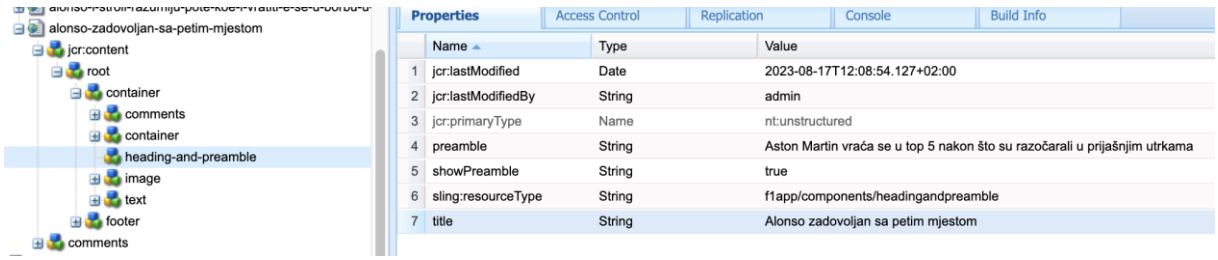
```
                        .map(Tag::getTagID)
                        .anyMatch(this.tagsForFiltering::contains);
    }
```

Metoda vraća *true* ako je lista tagova prazna, ili ako stranica sadrži jedan od tagova iz liste.

Nakon što je definirana lista stranica koje je potrebno prikazati, potrebno je kreirati objekt *PageListItem* koji sadrži sve podatke s karticama. Dohvaćanje svakog podatka prati istu logiku pa će u primjeru koda biti prikazano samo dohvaćanje naslova i kreiranje spomenutog objekta:

```
private PageListItem createPageListItem(final Resource resource) {
        final Resource contentResource =
this.resourceResolver.getResource(StringUtils.join(resource.getPath(),
PATH_TO_MAIN_PARSYS));
        String title = Optional.ofNullable(contentResource)
                .map(res -> res.getChild("heading-and-preamble"))
                .map(Resource::getValueMap)
                .map(valueMap->valueMap.get("title", StringUtils.EMPTY))
                .orElse(StringUtils.EMPTY);
        return PageListItem.builder()
                        .link(StringUtils.join(resource.getPath(), ".html"))
                        .image(image)
                        .title(title)
                        .preamble(preamble)
                        .date(lastUpdatedDateString)
                        .build();
    }
```

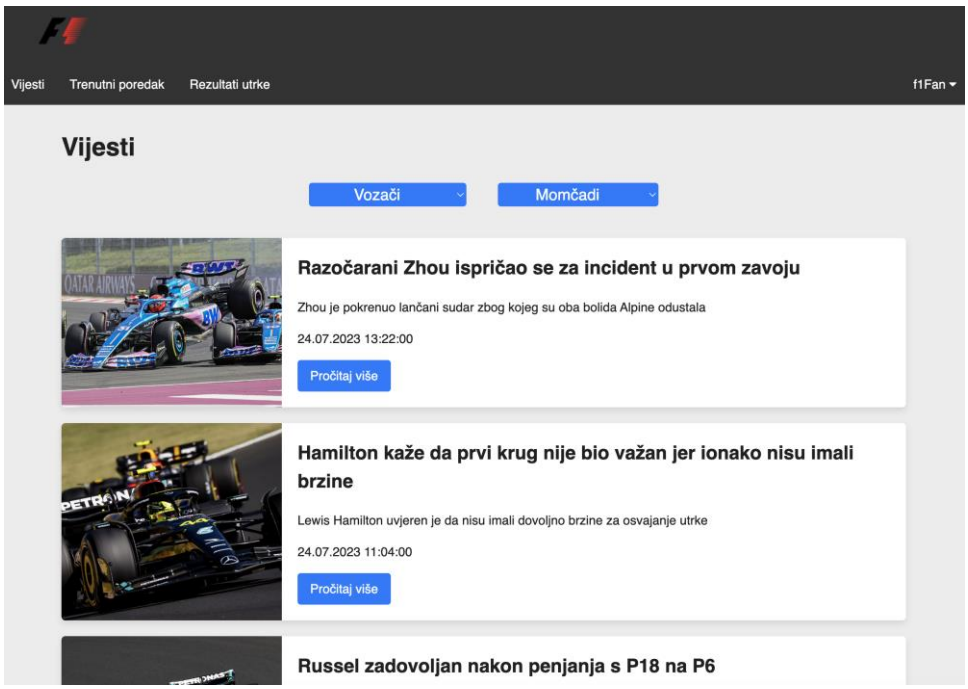
Kako bi bolje razumjeli dohvaćanje podataka iz komponente, na sljedećoj slici je prikazana struktura stranice:



Slika 58. Struktura komponenti na stranici (autorski rad)

contentResource je zapravo resurs čvora *container*. Metodom *getChild* i nazivom komponente dolazimo do resursa komponente. Metodom *getValueMap* s argumentom „title“ dohvaćamo istoimeni atribut spremljen u komponenti. Na isti način dohvaćaju se vrijednosti sadržaja, slike, linka, i datuma. Korištenjem *buildera* kreira se *PageListItem* objekt koji se

dohvaća u prednjem dijelu aplikacije te se kreiraju kartice. *Filter container, Tag filter* i *News List* komponente prikazane su na sljedećoj slici:



Slika 59. Stranica s člancima (autorski rad)

7.2.5. Komentiranje članaka

Ispod svakog članka nalazi se komponenta za komentiranje. Neregistrirani korisnici mogu samo pregledavati komentare, registrirani korisnici imaju mogućnost komentiranja te brisanja i uređivanja vlastitih komentara, dok administrator može komentirati i uređivati vlastite komentare, ali također ima ovlasti brisati svoje i tuđe komentare.

Komentiranje se svodi na slanje AJAX zahtjeva koji okida *CommentServlet*. Kao parametri šalju se „action“, „comment“ te „id“. „Action“ je radnja koju je potrebno obaviti, a može biti „add“, „delete“ ili „edit“. „Id“ je identifikator komentara koji treba urediti ili izbrisati, a „comment“ je sami tekst komentara koji treba dodati ili urediti. U nastavku je prikazan sadržaj prije spomenutog servleta.

```
protected void doPost(final SlingHttpServletRequest request, final
SlingHttpServletResponse response) throws ServletException, IOException {
      try {
            final ResourceResolver resourceResolver = request
                                                .getResourceResolver();
            final String action = this.getParameter(request, "action");
            final String currentPage = this.getCurrentPage(request);
            if (StringUtils.equals(action, "add")) {
                final String comment = this.getParameter(request,"comment");
```

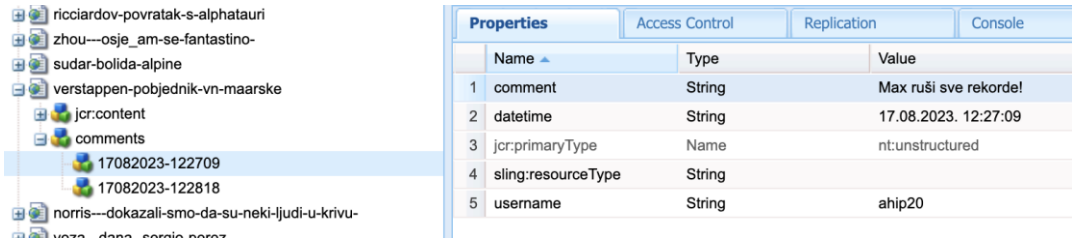
```
                this.commentService.addComment(currentPage,
                                    resourceResolver.getUserID(), comment);
                  }
              if (StringUtils.equals(action, "delete")) {
                  final String id = this.getParameter(request,"id");
                  this.commentService.deleteComment(currentPage, id);
                  }
              if (StringUtils.equals(action, "edit")) {
                   final String id = this.getParameter(request, "id");
                   final   String   comment   =   this.getParameter(request,
                                                                  "comment");
                   this.commentService.editComment(currentPage, id, comment);
                   }
             } catch (final RuntimeException e) {

response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
                   log.error("Error while handling comment", e);
              }
          }
```

U kôdu iznad možemo primijetiti da je prvi zadatak servleta provjeriti vrijednost parametra „action“. Ovisno o akciji zahtijevaju se i ostali parametri koji se šalju kao argumenti. Za pozadinsku logiku ove funkcionalnosti zadužen je *CommentService* servis koji ima implementirane metode za dodavanje, brisanje i uređivanje komentara. Metode su prilično slične, svodi se na dohvaćanje čvora koji sadrži komentar, a onda se ovisno o akciji manipulira čvorovima s komentarima. Ako se dodaje komentar, kreira se novi čvor. Prilikom kreiranja čvora kreira se i jedinstveno ime u obliku datuma i vremena dodavanja te se u čvor spremaju korisničko ime, vrijeme komentiranja i komentar. Kod brisanja šalje se identifikator komentara koji je potrebno obrisati, dok se kod uređivanja prosljeđuje identifikator i novi komentar. Svaka metoda prima i *currentPage* jer su komentari spremljeni unutar stranice samog članka. Zbog jednostavnosti bit će prikazana samo metoda za dodavanje komentara:

```
  public void addComment(final String pagePath, final String userID, final
String comment) {
        try      (final      ResourceResolver      resourceResolver      =
this.getResourceResolverFromServiceUser()) {
          final Instant instant = Instant.now();
          final String name = DateTimeFormatter.ofPattern("ddMMyyyy-HHmmss")
                          .withZone(ZoneId.systemDefault()).format(instant);
          final   Resource   newCom   =   ResourceUtil.getOrCreateResource
(resourceResolver,      StringUtils.join(pagePath,"/comments/",      name),
StringUtils.EMPTY, StringUtils.EMPTY, true);
            final    ModifiableValueMap    modifiableValueMap    =    newCom
                                      .adaptTo(ModifiableValueMap.class);
```

```
            modifiableValueMap.put("username", userID);
            modifiableValueMap.put("comment", comment);
            modifiableValueMap.put("datetime",
                        DateTimeFormatter.ofPattern("dd.MM.yyyy. HH:mm:ss")
                        .withZone(ZoneId.systemDefault()).format(instant));
            newCom.getResourceResolver().commit();
        } catch (LoginException | PersistenceException e) {
            log.error("Error while adding comment", e);         }
    }
```

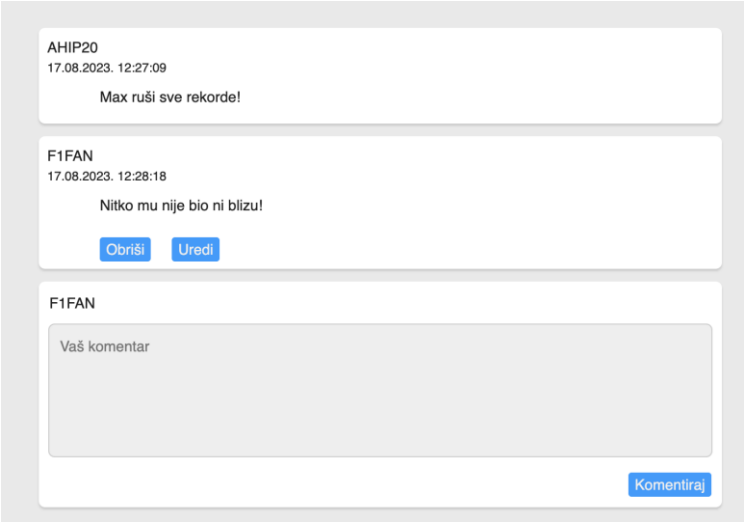
Na slici 60. možemo vidjeti da se kreira čvor „comments“ kao čvor dijete same stranice, a sadrži čvorove s komentarima.



Slika 60. Struktura komentara (autorski rad)

Metode za brisanje i uređivanje komentara zahtijevaju identifikator po kojemu pronalaze istoimeni čvor. U slučaju brisanja čvor se jednostavno izbriše, dok se kod uređivanja dohvaća *ModifiableValueMap* resursa pomoću koje se uređuju atributi, konkretno atribut „comment“.

Prikaz komentara implementiran je u samom Sling modelu komponente. Potrebno je dohvatiti svu djecu čvora „comments“, adaptirati svaki čvor na *CommentItem* te kreirati kartice sa potrebnim podacima. Na slici 61. prikazana je komponenta *Comments*.



Slika 61. Komponenta *Comments* (autorski rad)

7.2.6. Dodavanje favorita i prikaz preporučenih članaka

Prijavljeni korisnici imaju mogućnost dodavanja omiljenih vozača ili timova u favorite. Favoriti se spremaju u memoriju preglednika kako bi se zapamtio njihov odabir. Ideja ove funkcionalnosti je da korisnik odabere omiljene vozače i/ili timove kako bi mu se na naslovnici prikazali preporučeni članci.

Ova funkcionalnost sastoji se od dvije komponente, a to su *Favourites* i *Favourite news. Favourites* je komponenta koja se prikazuje u skočnom prozoru kada korisnik klikne na „Omiljeni vozači/timovi“. Komponenta se sastoji od dvije tzv. kartice (eng. Tab) na kojima su prikazani vozači odnosno timovi. Otvaranjem kartice prikazuju se vozači ili timovi s osnovnim informacijama: ime, prezime, slika, tim. Ispod svakog vozača nalazi se gumb kojim je moguće dodati ili ukloniti vozača/tim iz favorita. Dohvaćanje informacija o vozačima implementirano je unutar Sling modela komponente, odnosno *FavouritesModel* klase. Logika kod dohvaćanja vozača je sljedeća: prvo se pokušaju dohvatiti vozači iz JCR repozitorija koristeći metodu *getDriversFromDatabase* čiji je kôd prikazan u nastavku:

```
private List<DriverModel> getDriversFromDatabase(){
      List<DriverModel> driversList = new ArrayList<>();
      final    Resource    driversRootResource    =    this.resourceResolver
                              .getResource(F1AppConstants.DRIVERS_RESOURCE);
       if (driversRootResource != null) {
            final    List<Resource>    driverResources    =    IteratorUtils
                                .toList(driversRootResource.listChildren());
            if (CollectionUtils.isNotEmpty(driverResources)) {
                  driversList     =     driverResources.stream().map(d     ->
d.adaptTo(DriverModel.class)).collect(Collectors.toList());
                  }
              }
              return driversList;
          }
```

Za početak dohvaća se čvor koji je predefiniran za spremanje informacija o vozačima. Ako taj čvor ne postoji, znači da nema spremljenih informacija o vozačima te se vraća prazna lista. Ako čvor postoji, dohvaćaju se njegova djeca te se svaki čvor dijete adaptira na *DriverModel* kako bi se injektirale vrijednosti atributa čvora.

Ukoliko je vraćena prazna lista, potrebno je preuzeti podatke koristeći API. Sve metode koje su vezane uz API nalaze se unutar servisa *ApiService* pa tako i metoda *APIDriversCall.*

```
private void APIDriversCall() {
  List<DriverDTO> drivers = new ArrayList<>();
     try {
        // … (inicijalizacija HttpClienta, HttpUriRequesta…)
        final HttpResponse response = httpClient.execute(getApiRequest);
        if (response.getStatusLine().getStatusCode() == 200) {
        final DriverStandingsResponse driverStandingsResponse= objectMapper
            .readValue(EntityUtils.toString(response.getEntity()),
                                        DriverStandingsResponse.class);
        drivers = driverStandingsResponse
                        .getMrData()
                        .getStandingsTable()
                        .getStandingsLists()
                        .get(0)
                        .getDriverStandings()
                        .stream()
                        .map(this::getDriverDTO)
                        .filter(Objects::nonNull)
                        .collect(Collectors.toList());

           if (CollectionUtils.isNotEmpty(drivers)) {
             this.dataService.saveDrivers(drivers);
                   }
             }
            //catch
          }
```

U kôdu iznad prikazano je dohvaćanje informacija o vozačima koristeći API. Nakon šte se inicijalizira HttpClient i UriRequest koristeći krajnju točku za dohvaćanje vozača, izvršava se zahtjev. Nakon toga provjerava se statusni kod odgovora. Ukoliko je statusni kod 200, što znači da je zahtjev obrađen bez problema, dobivene vrijednosti mapiraju se na *DriverStandingsResponse* klasu koja sadrži sve ostale objekte tj. entitete. Sljedeći bitan dio koda je mapiranje podataka na *DriverDTO*, tj. klasu koja služi kao prenositelj podataka (eng. Data Transfer Object), a sadrži sve potrebne informacije o vozaču: ime, prezime, kod, tim. Nakon dohvaćanja podataka za sve vozače, potrebno je iste i upisati u JCR bazu. Za rad s JCR zadužen je servis *DataService*, a za unos vozača konkretno metoda *saveDrivers*:

```
public void saveDrivers(final List<DriverDTO> drivers) {
  try        (final        ResourceResolver        resourceResolver        =
this.getResourceResolverFromServiceUser())                                  {

                        ...
```

```
    for (final DriverDTO driver : drivers) {
        this.createDriverTagIfNotExisting(resourceResolver, driver);
        final Resource driverResource = ResourceUtil.getOrCreateResource
          (resourceResolver, StringUtils.join(DRIVERS_RESOURCE, "/",
                  driver.getCode()), JcrConstants.NT_UNSTRUCTURED,
                      StringUtils.EMPTY, true);
        final ModifiableValueMap modifiableValueMap = driverResource
                                    .adaptTo(ModifiableValueMap.class);
        if (modifiableValueMap != null) {
           modifiableValueMap.put("firstName", driver.getName());
           modifiableValueMap.put("lastName", driver.getLastName());
           modifiableValueMap.put("code", driver.getCode());
           modifiableValueMap.put("team", driver.getTeam());
                   }
               }
               driversResource.getResourceResolver().commit();
           }
```

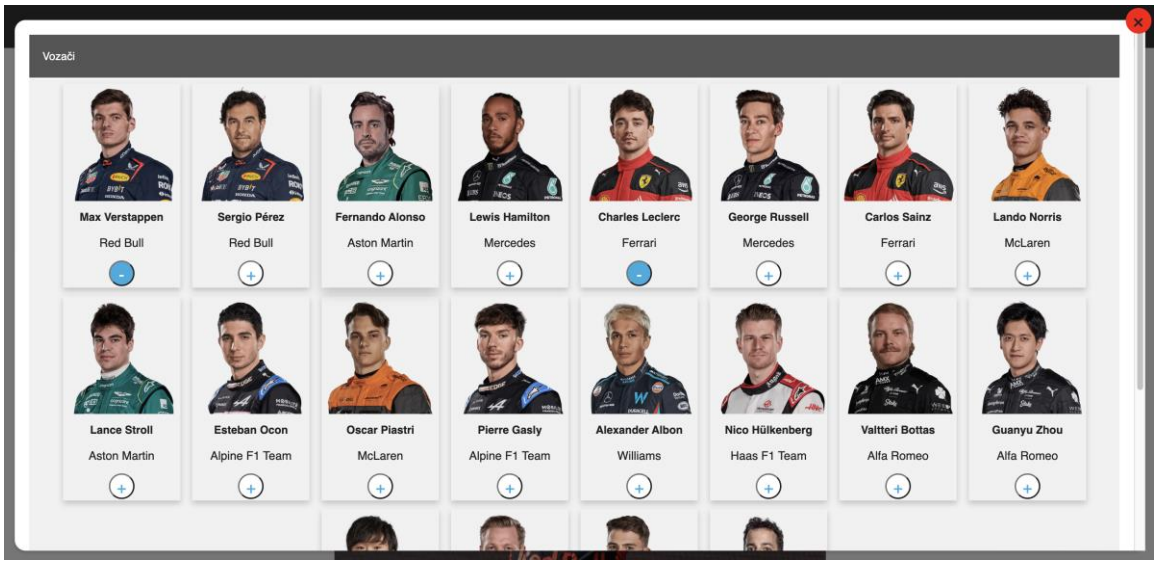
Za svakog vozača kreira se novi čvor s atributima koji sadrže potrebne podatke o vozaču. Ime čvora određeno je kodom vozača (npr. „ver“ za Maxa Verstappena). Za svakog vozača poziva se metoda *createTagIfNotExisting* koja kreira oznaku vozača ako već ne postoji. Oznake su već spomenute kod funkcionalnosti pregledavanja članaka. Njima se označuje pojedini članak, a služe za filtriranje po vozaču odnosno timu. Autori imaju mogućnost kreiranja oznaka pomoću grafičkog sučelja samog AEM-a, ali u ovom slučaju oznake se generiraju preko kôda već spomenutom metodom.

```
private void createDriverTagIfNotExisting(final ResourceResolver
resourceResolver, final DriverDTO driver) {
  final TagManager tagManager = this.tagManagerFactory
                      .getTagManager(resourceResolver);
  final String driverTagPath=StringUtils.join(F1AppConstants.DRIVERS_TAGS,
                                                "/", driver.getCode());
  final Tag driverTag = tagManager.resolve(driverTagPath);
  if (driverTag == null) {
    try {
       tagManager.createTag(driverTagPath, driver.getLastName(),
                                                StringUtils.EMPTY);
      } catch (InvalidTagFormatException e) {
           log.error("Error while creating tag.", e);
      }
    }
}
```

Za manipulaciju oznakama potreban je *TagManager* koji se dobiva adaptiranjem *ResourceResolvera.* Metodom *resolve* dohvaća se oznaka s putanjom koja se šalje kao jedini argument. Ako postoji oznaka na zadanoj putanji metoda vraća samu oznaku, a ako ne postoji vraća se *null* vrijednost. U ovom slučaju zanima nas slučaj kada se vraća *null* jer to znači da ne postoji oznaka za vozača. Korištenjem metode *createTag* kreira se oznaka na određenoj putanji. Metoda prima tri argumenta: putanja, ime oznake, opis oznake.

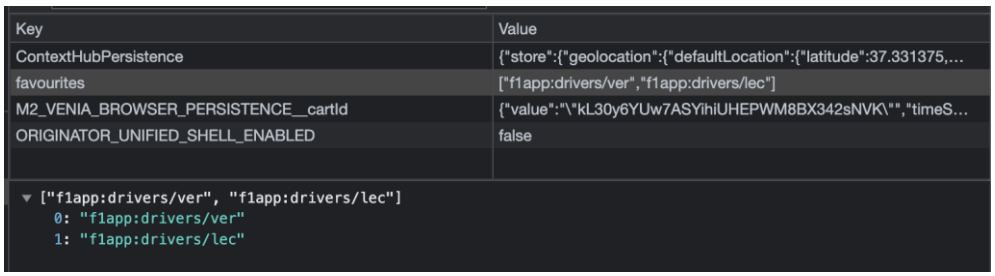
Nakon što se pohrane vozači, Sling model ih pokušava opet prvo preuzeti iz repozitorija. Nakon što se vozači uspješno preuzmu, kreiraju se kartice sa slikom, imenom, timom i gumbom za dodavanje/brisanje iz favorita.

Valja napomenuti da je isti slučaj s preuzimanjem timova. Na sljedećoj slici prikazan je skočni okvir s *Favourites* komponentom kojom korisnici upravljaju favoritima:



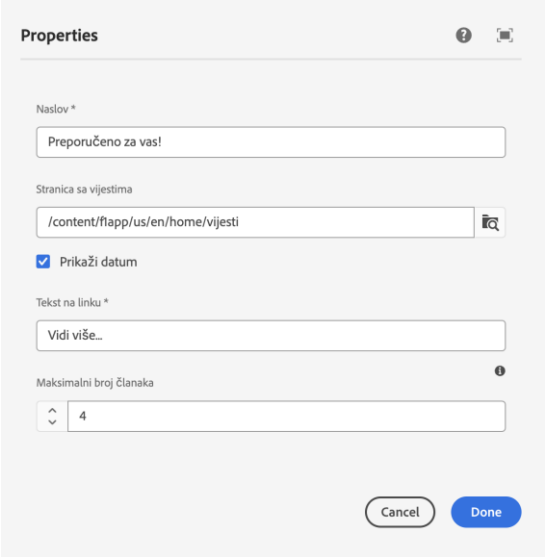
Slika 62. Skočni prozor za upravljanje favoritima (autorski rad)

Na slici 62. možemo vidjeti kartice sa svim vozačima na kojima se nalazi i gumb za dodavanje, odnosno brisanje vozača iz favorita. U konkretnom slučaju u favoritima se nalaze Max Verstappen te Charles Leclerc. Pregledom memorije preglednika možemo vidjeti da se u favorite spremaju oznake odabranih vozača:



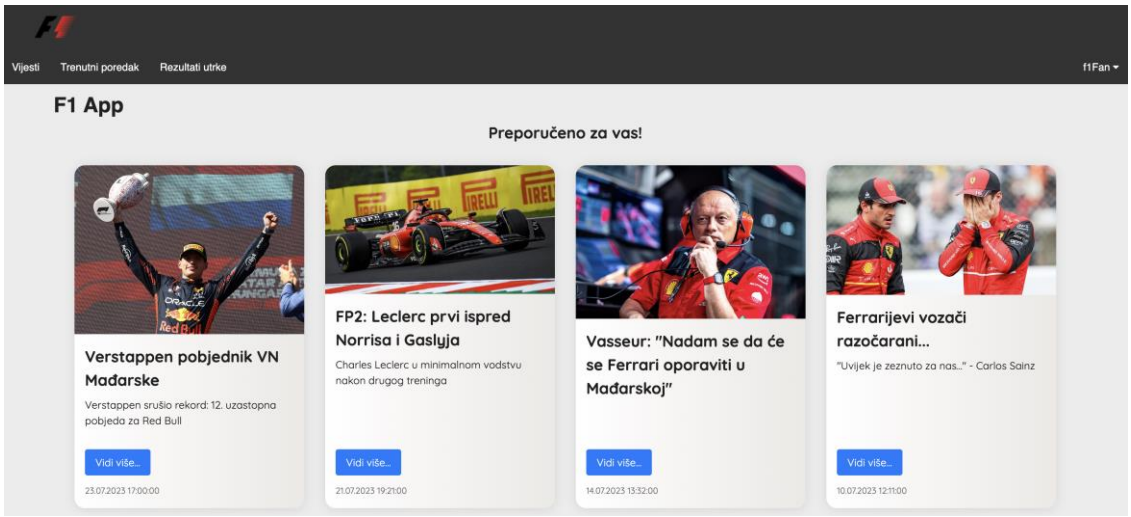
Slika 63. Memorija preglednika (autorski rad)

Ove podatke koristi *Favourite news* komponenta koja ima istu logiku kao i *News list* koja je objašnjena kod funkcionalnosti pregleda članaka. Razlika je u autorskom dijelu jer kod ove komponente autori imaju više mogućnosti. Na raspolaganju im je postavljanje naslova koji će biti prikazan na vrhu komponente, definiranje roditeljske stranice koja sadrži stranice s člancima, prikazivanje datuma, unos teksta koji će biti prikazan na gumbu s linkom, a mogu i zadati maksimalan broj članaka koji će se prikazivati na naslovnici. Na sljedećoj slici prikazan je dijalog komponente.



Slika 64. Dijalog *Favourite news* komponente (autorski rad)

Na sljedećoj slici prikazana je i sama komponenta. Možemo primijetiti da su prikazani članci povezani s vozačima koji su odabrani kao favoriti.



Slika 65. Prikaz *Favourite news* komponente (autorski rad)

7.2.7. Poredak vozača / timova

Prikaz trenutnog poretka vozača ili timova omogućava *Standings* komponenta. Kako bi se spriječilo masovno pozivanje API-ja podaci su spremljeni u JCR. Razlika između podataka o trenutnom poretku i podataka o vozačima je da se trenutni poredak mijenja. Zbog toga je potrebno na neki način provjeravati jesu li trenutni podaci validni.

Kako bi se riješio ovaj problem implementiran je Sling raspoređivač (eng. Scheduler) koji periodički, ovisno o Cron ekspresiji, pokreće Sling posao (eng. Job). Unutar *StandingsImportScheduler* klase najvažnije metode su *activate* i *deactivate*. *Activate* metoda anotirana je istoimenom anotacijom i izvršava se prilikom registriranja tj. aktiviranja samog raspoređivača.

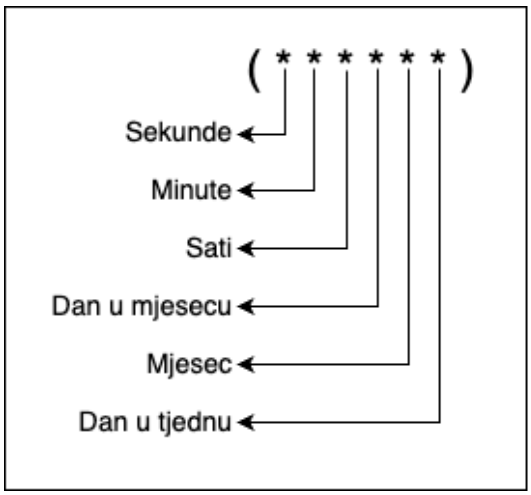
```
@Activate
@Modified
private void activate(final StandingsImportSchedulerConfig config) {
        this.scheduleExpression = config.scheduleExpression();
        this.isEnabled = config.enabled();
        this.unScheduleJob();
        this.scheduleJob();
    }
```

Možemo primijetiti da metoda *activate* dohvaća varijable konfiguracije te prvo uklanja poslove, a zatim ih zakazuje. Metoda za zakazivanje poslova *scheduleJob* prikazana je u nastavku:

```
    private void scheduleJob() {
        try {
            if (this.isEnabled) {
                final JobBuilder.ScheduleBuilder scheduleBuilder =
this.jobManager.createJob(JOB_TOPIC).schedule();
                scheduleBuilder.cron(this.scheduleExpression);
                scheduleBuilder.add();
            }
        } catch (final RuntimeException e) {
            log.error("Unable to schedule a job", e);
        }
    }
```

Dobra praksa svakog raspoređivača je postojanje boolean varijable koja omogućuje da zaustavimo ili pokrenemo zakazivanje posla koji se periodički obavlja. U gornjoj metodi vidljivo je da će se Sling posao zakazati samo ako je varijabla *isEnabled* postavljena na *true*. Zatim

se zadaje Cron ekspresija. Cron ekspresija određuje kada će se posao periodički izvršavati. Cron ekspresija zadaje se u String formatu s pet mjesta: (* * * * *) od kojih svako ima svoje značenje:



Slika 66. Cron ekspresija (autorski rad)

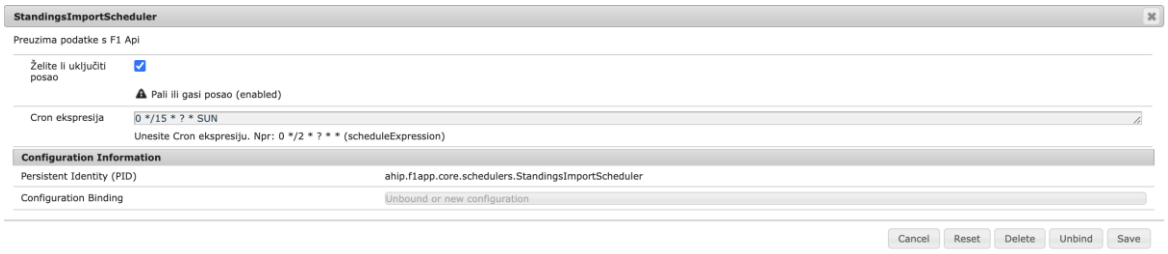
Na slici 66. možemo vidjeti objašnjenje svakog mjesta unutar Cron ekspresije. Simboli kojima se određuju ekspresije su sljedeći:

- * – bilo koja vrijednost
- , – separator vrijednosti
- - – označava raspon
- ? – ne specifična vrijednost – koristi se kod dana u mjesecu ili dana u tjednu
- L – zadnji dan mjeseca ili tjedna

Kako bi bolje razumjeli ove ekspresije, slijedi nekoliko primjera:

- 0 0 * * * – pokreće se u ponoć svakog dana
- 0 5 * * 1-5 – pokreće se u 5:00 radnim danima
- 0 */2 * ? * * – pokreće se svake dvije minute
- 0 0 12 ? * SUN – pokreće se svake nedjelje u podne

Kako bi autori mogli podesiti Cron ekspresiju ili isključiti posao, na raspolaganju im je konfiguracija u AEM konzoli koja se nalazi na: http://localhost:4502/system/console/configMgr, a vidljiva je na slici ispod.



Slika 67. Konfiguracija *StandingImportScheduler* (autorski rad)

Imajući na umu da se utrke Formule 1 voze nedjeljom, jedna od mogućnosti za Cron ekspresiju vidljiva je na primjeru iznad. U ovom slučaju raspoređivač će pokretati posao za ažuriranje podataka svakih 15 minuta u nedjelju.

Klasa *StandingsImportJob* implementira *JobConsumer* i nadjačava metodu *process* u koju se zapravo smješta logika koja će se periodički izvršavati.

```
@Override
    public JobResult process(final Job job) {
        try {
            this.apiService.getCurrentTeamsStandings();
            this.apiService.getCurrentDriverStandings();
            return JobResult.OK;
        } catch (final RuntimeException e) {
            log.error("Exception in job consumer", e);
            return JobResult.FAILED;
        }
    }
```

U primjeru iznad vidimo da se u metodi *process* poziva *ApiService* koji će preuzeti trenutni poredak vozača i timova.

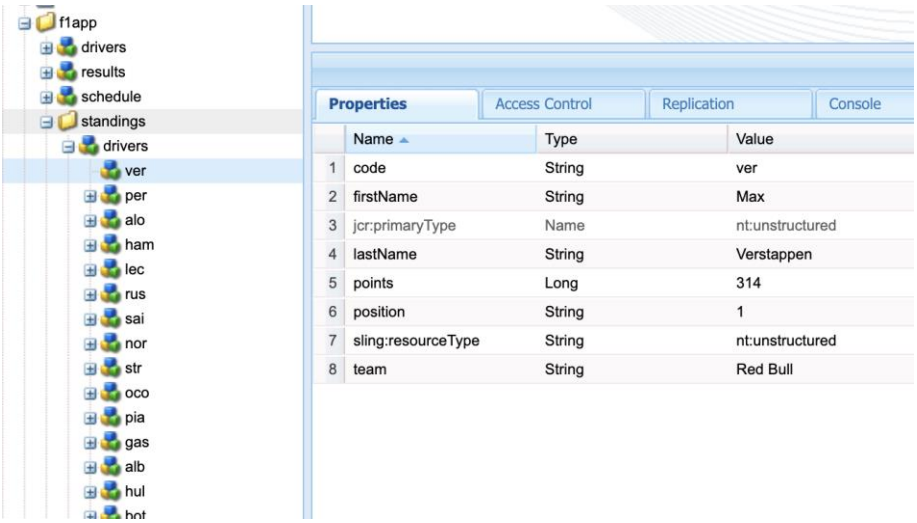
Periodičko pozivanje API-ja potrebno je za validaciju podataka. Usporedba postojećih i validnih podataka jedini je način da saznamo jesu li podaci iz JCR aktualni. Kako bi se proces zapisa i usporedbe maksimalno smanjio, prilikom preuzimanja poretka dohvaća se JSON odgovor. JSON se sprema u roditeljski čvor čija djeca sadrže trenutni poredak s brojem bodovima i pozicijama pojedinih vozača. Ovaj pristup uvelike olakšava validaciju jer je samo potrebno usporediti JSON koji se nalazi u repozitoriju s JSON koji je vratio API. Ako su odgovori u JSON formatu jednaki, znači da nije bilo promjena te nema potrebe za daljnjim akcijama jer su podaci aktualni. Ako JSON odgovor API-ja ipak ne odgovara JSON podatku

spremljenom u repozitoriju znači da su podaci zastarjeli i potrebno je spremiti nove podatke iz API-ja u repozitorij.

U nastavku se nalazi isječak kôda koji provodi objašnjenu logiku:

```
@Override
public void saveDriversStandings(final List<DriverDTO> drivers, final
     String standingsJson) {

   try (final ResourceResolver resourceResolver =
        this.getResourceResolverFromServiceUser()) {

      final Resource driversStandingsResource = ResourceUtil
         .getOrCreateResource(resourceResolver, DRIVERS_STANDINGS_RESOURCE,
         JcrConstants.NT_UNSTRUCTURED, StringUtils.EMPTY, true);

      final List<Resource>  driversResources = IteratorUtils
                           .toList(driversStandingsResource.listChildren());

      final String existingJson = Optional.ofNullable(resourceResolver
                     .getResource(driversStandingsResource.getPath()))
                     .map(Resource::getValueMap)
                     .map(valueMap -> valueMap.get("existingJson",
                                              StringUtils.EMPTY))
                     .orElse(StringUtils.EMPTY);

        if (CollectionUtils.isEmpty(driversResources) ||
              !StringUtils.equals(existingJson, standingsJson)) {

           resourceResolver.delete(driversStandingsResource);

           resourceResolver.commit();

           final Resource newStandingsResource = ResourceUtil
                     .getOrCreateResource(resourceResolver,
                     DRIVERS_STANDINGS_RESOURCE,
                     JcrConstants.NT_UNSTRUCTURED,
                     StringUtils.EMPTY, true);

                     . . .
```

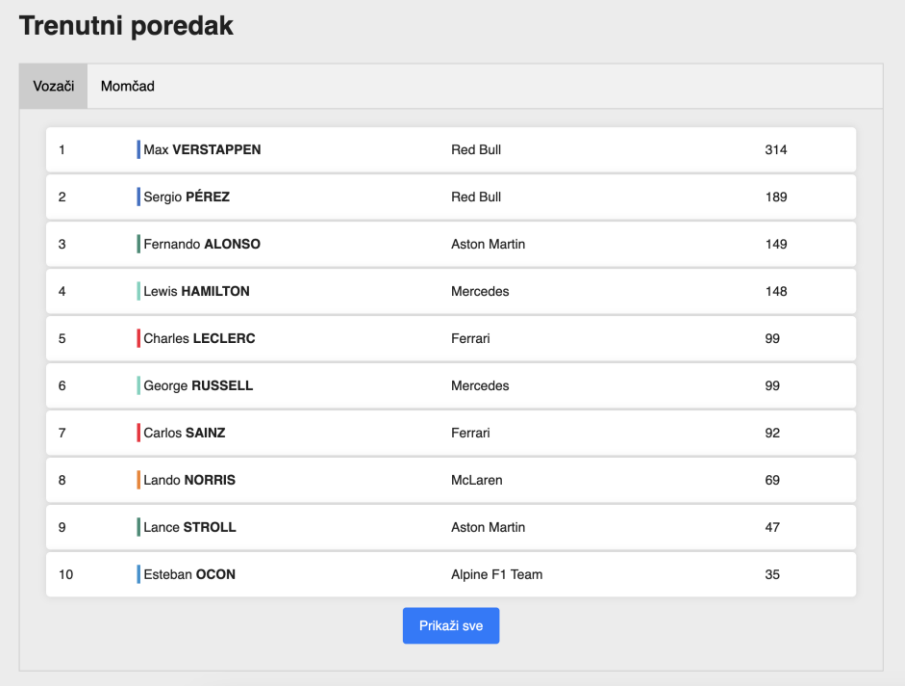
Struktura spremanja poretka u repozitoriju vidljiva je na slici 68. Čvor „drivers“ sadrži spomenuti JSON podatak koji se uspoređuje s odgovorom API-ja.



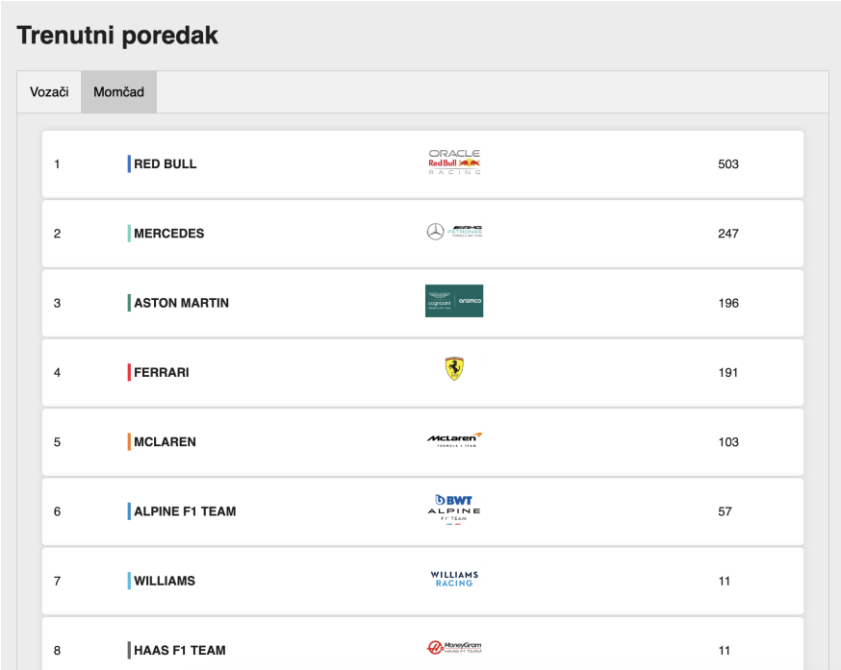
Slika 68. Struktura spremanja trenutnog poretka vozača (autorski rad)

U *Standings* komponenti autor ima nekoliko mogućnosti: Može odabrati koji poredak će se prikazati (samo vozači, samo momčad, vozači i momčad) te može limitirati poredak vozača. Ukoliko se limitira poredak, potrebno je unijeti i broj koji označava koliko vozača će biti prikazano. Npr. ako autor ograniči prikaz poretka te unese broj 10, znači da će se prikazati prvih 10 vozača i nakon tablice pojavit će se gumb „Prikaži više“ koji će proširiti tablicu do svih 20 vozača. Na dnu tablice također se mijenja gumb koji postaje „Prikaži manje“ te sakriva zadnjih 10 vozača.

Na slici 69. prikazana je funkcionalnost trenutnog poretka:



Slika 69. Trenutni poredak vozača (autorski rad)



Slika 70. Trenutni poredak momčadi (autorski rad)

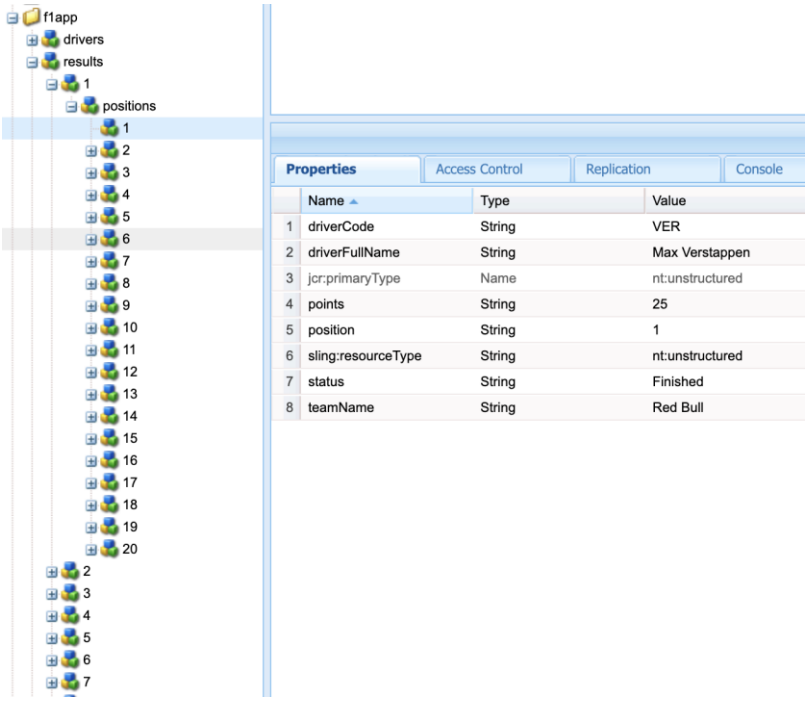
7.2.8. Rezultati utrke

Prikaz rezultata utrke omogućava *Race results* komponenta. Rezultati utrka su također podaci koji mijenjaju, tj. nakon svake utrke postoji novi set podataka s rezultatima utrke koji treba preuzeti. Zbog toga se kao i kod prethodne funkcionalnosti koristi Sling posao koji se periodički obavlja. Logika ove funkcionalnosti je sljedeća: prvo treba provjeriti zadnji broj utrke koji je spremljen u repozitoriju. Ako ne postoji zapis o utrkama, funkcija koja je zadužena za provjeravanje vratit će -1. Nakon toga potrebno je dohvatiti zadnji aktualni broj utrke koristeći API. Ako je metoda vratila -1, potrebno je preuzeti sve dosadašnje rezultate utrka aktualne sezone. Ako ipak postoji zapis u repozitoriju, provjerava se redni broj utrke u repozitoriju s rednim brojem zadnje utrke koju vraća API. Ako je redni broj utrke u repozitoriju manji od broja koji vraća API, znači da postoje novi rezultati koje je potrebno preuzeti. Kôd koji implementira navedenu logiku je sljedeći:

```
public void getRaceResults() {

final int lastExistingRound = this.dataService.getLastRaceResultRound();

final int lastActualRound = this.getLastRaceResultsRound();

if (lastExistingRound == -1) {

      this.getAllRaceResults(lastActualRound);
      }
```

```
else if (lastActualRound > lastExistingRound) {
      this.getSpecificRoundRaceResult(lastActualRound);
      }
}
```

Možemo primijetiti dvije metode koje se koriste za dohvaćanje rezultata utrke. Prva metoda je *getAllRaceResults* koja kao argument prima redni broj zadnje utrke za koje su dostupni podaci na API-ju. Jedini zadatak ove metode je unutar for petlje pozvati metodu *getSpecificRoundRaceResults* za sve utrke, tj. od prve utrke do zadnje utrke za koje postoje rezultati. Metoda *getSpecificRoundRaceResults* slična je već objašnjenim metodama koje dohvaćaju podatke s API-ja. Prvo preuzima podatke, kreira DTO objekt, a nakon toga koristi *DataService* za spremanje u repozitorij. Struktura spremanja rezultata utrke vidljiva je na sljedećoj slici:

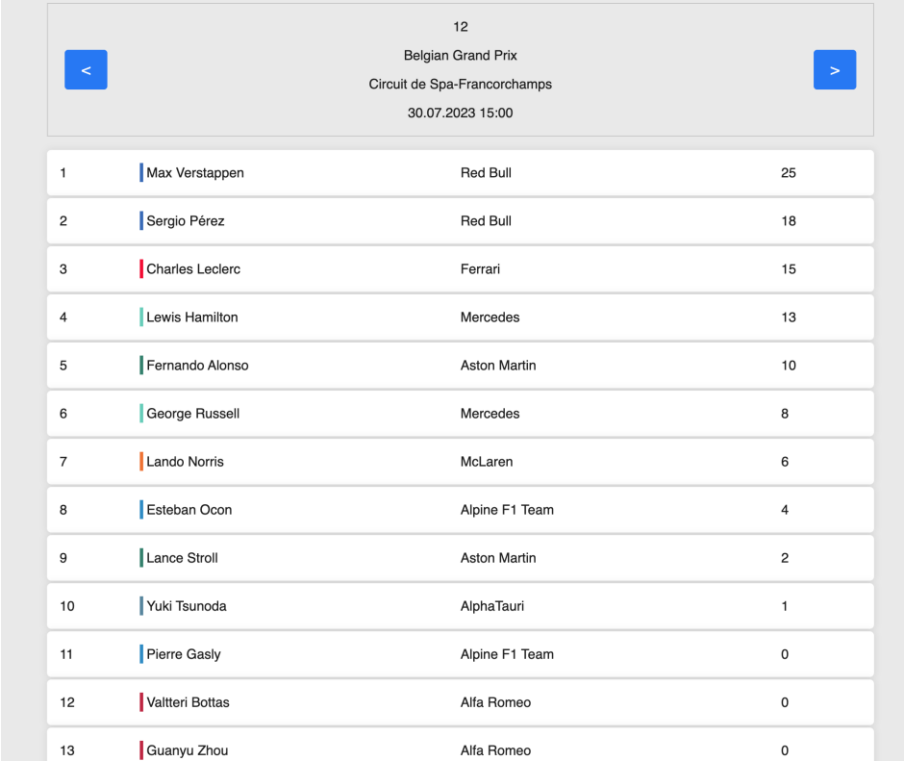


Slika 71. Struktura spremanja rezultata utrka (autorski rad)

Rezultati se spremaju u „results“ čvoru. Svaki čvor dijete predstavlja rezultate pojedine utrke, a nazvan je po rednom broju utrke. Ovaj čvor sadrži osnovne podatke o utrci poput imena utrke, imena staze, imena zemlje te datum i vrijeme utrke. Ispod čvora utrke nalazi se čvor „positions“ koji sadrži čvorove s pojedinačnim ostvarenim pozicijama. Čvor „positions“ kreiran je samo kako bi se jednostavnije injektirali čvorovi pozicija korištenjem *@ChildResource* anotacije koja služi za injektiranje čvorova djece određenog čvora. Sami

čvorovi pozicija sadrže osnovne informacije o vozaču koji je zasjeo na određenu poziciju. To su kod vozača, ime i prezime, broj ostvarenih bodova, pozicija, status i momčad.

Autori ove komponente u dijalogu imaju mogućnosti prikazati sve rezultate utrka aktualne sezone ili samo rezultate zadnje utrke. Ako se prikazuju svi rezultati, logika komponente je sljedeća: prvo se dohvaćaju rezultati zadnje utrke. Korisnici imaju na raspolaganju dva gumba kojim se kreću kroz rezultate utrka. Kada korisnici pritisnu jedan od gumbova, šalje se AJAX zahtjev s parametrom koji sadrži podatak o zahtijevanom broju utrke te se vraćaju traženi podaci. Komponenta za prikaz rezultata utrke prikazana je na sljedećoj slici.



Slika 72. Rezultati utrke (autorski rad)

7.2.9. Raspored utrka

U sklopu naslovnice pruža se još jedna funkcionalnost, a to je pregled rasporeda utrka s mogućnošću preuzimanja iCalendar datoteke kako bi se sam događaj spremio u vlastiti kalendar. Prilikom učitavanja komponente provjerava se postoje li zapisi o rasporedu utrka unutar repozitorija. Ako ne postoje, raspored se skida s API-ja i podaci se spremaju u repozitorij. Najvažniji podaci koji se spremaju su sljedeći: id staze, ime staze, država, ime utrke, redni broj utrke, datum i vrijeme utrke, vrsta utkre (normalna ili sprint), datum i vrijeme prvog

treninga, datum i vrijeme drugog i trećeg treninga ukoliko se ne radi o sprint utrci u protivnom datum i vrijeme sprint utrke te datum i vrijeme kvalifikacija.

Pozadinska logika nema puno promjena u odnosu na ostale komponente. Prilikom dohvaćanja utrka iz repozitorija bitno je pronaći koja utrka je sljedeća kako bi se prikazao raspored potrebnih utrka (koliko utrka prije, koliko utrka nakon nadolazeće). Kako bi se ovo ostvarilo, zadan je brojač izvan for petlje. Kada se for petljom dođe do prve sljedeće utrke, zaustavlja se petlja. Na temelju indeksa se dohvaća početni indeks (koliko utrka prije) te završni indeks (koliko utrka nakon). Kako bi se spriječilo prekoračenje graničnih vrijednosti (npr. 4 utrke prije prve utrke) koriste se *min* i *max* metode koje ograničavaju krajnje vrijednosti. Nakon što se izračunaju početni i završni indeksi, metodom *sublist* dobije se skraćena potrebna lista rasporeda utrka:

```
private void getRequiredRaceSchedules() {
      int index = 0;
      final DateTimeFormatter dateTimeFormatter = DateTimeFormatter
                                          .ofPattern("dd.MM.yyyy HH:mm");
      for (final ScheduleModel scheduleModel : this.schedules) {
          index ++;
          final LocalDateTime date = LocalDateTime.parse(scheduleModel
                           .getFormattedRaceDatetime(), dateTimeFormatter);
          if (date.isAfter(LocalDateTime.now())){
              break;
              }
          }
      final int upLimit = Math.min(index + this.racesAfter,
                                             this.schedules.size());
      final int downLimit = Math.max(index-1-this.racesBefore, 0);
      this.schedules = this.schedules.subList(downLimit, upLimit);
}
```

Kako bi korisnici mogli dodati utrku u raspored potrebno je kliknuti gumb „Dodaj utrku u kalendar“. Klikom na gumb okida se servlet za preuzimanje .ics datoteke. Kako bi se kreirala datoteka, potrebne su osnovne informacije o događaju: datum i vrijeme, ime utrke te ime staze.

Za kreiranje ics Stringa potrebno je obratiti pažnju na format datuma koji mora biti u obliku (yyyyMMdd'T'HHmmss).

U nastavku je prikazan sadržaj servleta za preuzimanje .ics datoteke:

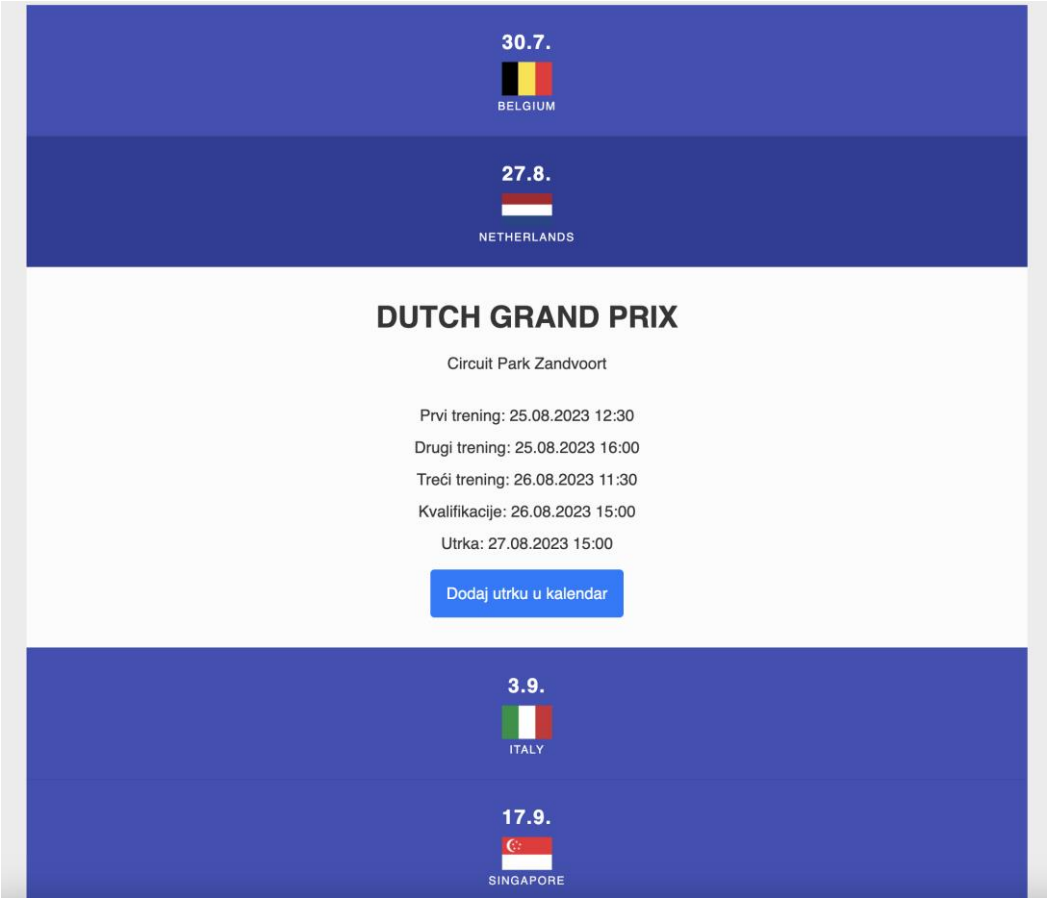
```
final String dateString = this.getParameter(request, "date");
final String raceName = this.getParameter(request, "race");
final String circuitId = this.getParameter(request, "circuit");
final LocalDateTime dateStart = this.formatStringToDate(dateString);
```

```
final LocalDateTime dateEnd = dateStart.plusHours(2);
final DateTimeFormatter dateTimeFormatter = DateTimeFormatter
                                            .ofPattern("yyyyMMdd'T'HHmmss");
final String formattedDateStart = dateStart.format(dateTimeFormatter);
final String formattedDateEnd = dateEnd.format(dateTimeFormatter);
final String ics = this.createIcs(raceName, raceName, formattedDateStart,
formattedDateEnd, circuitId);
final String fileName = StringUtils.join(raceName, ".ics");
response.setContentType("text/calendar");
response.setHeader("Content-disposition", String.format("attachment;
                                            filename=%s", fileName));
response.getWriter().print(ics);
response.getWriter().flush();
```

Prilikom konfiguracije komponente autori imaju mogućnost odabrati naslov iznad komponente, koliko termina utrka unazad te koliko termina utrka unaprijed će biti prikazano. Posljednja mogućnost je da se prikazuju ili sakriju termini slobodnih treninga.

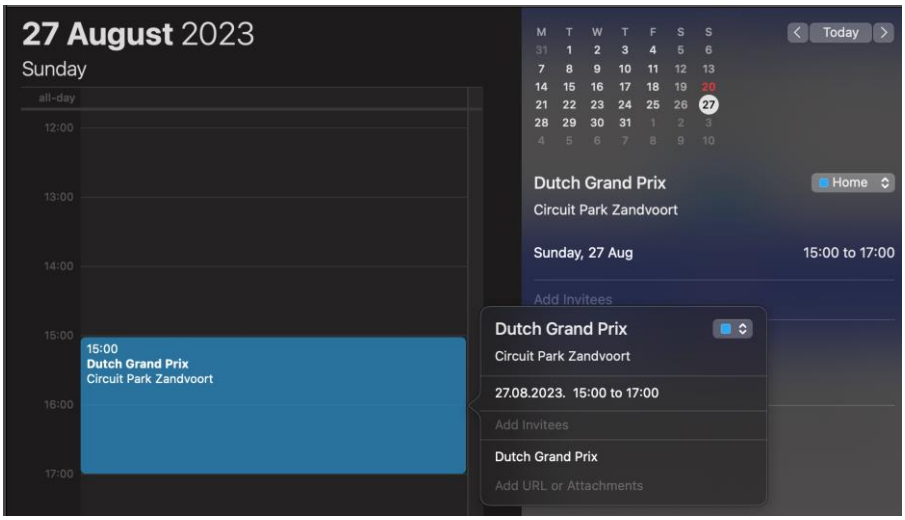
Izgled rasporeda utrka zamišljen je na principu harmonike i to tako da je sljedeća nadolazeća utrka uvijek otvorena, a klikom na svaku ostalu trenutni sadržaj se zatvara i otvaraju se rasporedi zahtijevane utrke.

Na slici 73. nalazi se *Schedule* komponenta:



Slika 73. Raspored utrka (autorski rad)

Preuzimanjem .ics datoteke događaj se dodaje u vlastiti kalendar:



Slika 74. Prikaz utrke u kalendaru (autorski rad)

8. Zaključak

U današnje vrijeme brzina kreiranja web sadržaja je ključna za organizacije kako bi privuklo što više korisnika. Korisnici Weba su sve više nestrpljivi i ne toleriraju aplikacije koje se dugo učitavaju, kasne s objavom sadržaja, nisu vizualno privlačne. Kako bi organizacije unaprijedile svoje web stranice, sve više teže implementaciji vlastitog CMS rješenja. CMS sustavi pružaju autorima brojne prednosti prilikom kreiranja i uređivanja web sadržaja, ali isto tako i smanjuje cijenu održavanja stranica.

Jedan od takvih CMS sustava je Adobe Experience Manager koji između ostalog nudi spremanje i upravljanje velikom količinom digitalne imovine, integraciju u oblaku, brzo pretraživanje, pretvaranje slika i videa. Navedenim prednostima AEM pokušava osvojiti sve više korisnika koji će se odlučiti za ovaj CMS.

Prilikom izrade aplikacije u AEM-u obično se ovisno o korištenim tehnologijama nude tri pristupa: standardni pristup, SPA pristup te pristup korištenjem web komponenata. Iako je standardni pristup najstariji, još uvijek je najstabilniji i najviše korišten. Glavni nedostatak ovog pristupa, preuzimanje čitavih HTML datoteka prilikom prelaska sa stranice na stranicu, izazvao je pojavu SPA pristupa. Korištenjem ovog pristupa kreiraju se brze, moderne, vizualno privlačne aplikacije s bogatim korisničkim sučeljima. Imajući u vidu da se web sve više čita na manjim uređajima poput pametnih mobitela i tableta, možemo očekivati da će u skorijoj budućnosti SPA aplikacije biti standard. Dokaz tome su velike aplikacije poput Facebooka koje se sve više okreću SPA tehnologiji. Glavna prednost MPA pristupa je kontrola velikog broja stranica pa se javljaju i stranice koje koriste i SPA i MPA tehnologije.

Web komponente bile su predstavljene kao revolucionarna tehnologija s brojim prednostima, međutim, dojam je da su razočarale. Sve prednosti web komponenata dobro izgledaju „na papiru“, ali u praksi implementacija je zahtjevna. Iako su osmišljene da se ponašaju kao klasični HTML blokovi, zahtijevaju puno bolje poznavanje web tehnologija za implementaciju. Zbog toga u svijetu AEM-a nisu našle svoju primjenu.

Na kraju ovog rada teško je izdvojiti koji pristup je najbolji. Prilikom planiranja izrade aplikacije potrebno je provjeriti prednosti i nedostatke svakog pristupa te odabrati onaj koji najviše odgovara potrebi i namjeni aplikacije.

Popis literature

[1] Andy Smith, „6 KEY BENEFITS OF CUSTOM WEB APPLICATIONS“, Pristupano 3.6.2023. [Na internetu]. Dostupno na: https://hatchworks.com/6-key-benefits-of-custom-web-applications/

[2] T. Berners-Lee, „Information Management: A Proposal“, *CERN*, 1989.

[3] „World Wide Web“, Pristupano 3.6.2023. [Na internetu]. Dostupno na: http://info.cern.ch/hypertext/WWW/TheProject.html

[4] Tim O'Reilly, „What Is Web 2.0“, *O'Reilly*, 2005.

[5] Pragati Verma, „Evolution of Web“, Pristupano 3.6.2023. [Na internetu]. Dostupno na: https://dev.to/pragativerma18/evolution-of-web-42eh

[6] Tim O'Reilly, John Battelle, „Web Squared: Web 2.0 Five Years On“, *O'Reilly Media*, 2009.

[7] Casimir Saternos, „Client-Server Web Apps with JavaScript and Java“, *O'Reilly Media*, 2014.

[8] Yurii Luchaninov, „Web Application Architecture in 2023: Moving in the Right Direction“, Pristupano 4.6.2023. [Na internetu]. Dostupno na: https://mobidev.biz/blog/web-application-architecture-types

[9] Jaydeep Paradiya, „SPA vs MPA: A Comprehensive Guide“, Pristupano 7.6.2023. [Na internetu]. Dostupno na: https://radixweb.com/blog/single-page-application-vs-multi-page-application

[10] V.Solovei, O.Olshevska, Y.Bortsova, „THE DIFFERENCE BETWEEN DEVELOPING SINGLE PAGE APPLICATION AND TRADICIONAL WEB APPLICATION BASED ON MECHATRONICS ROBOT LABORATORY ONAFT APPLICATION“, *ATBP*, 2018.

[11] Fernando Monteiro „Learning Single-page Web Application Development“, *Packt Publishing*, december 2014.

[12] Emmit A. Scott Jr., „SPA Design and Architecture: Understanding single-page web applications“, *Manning Publications*, November 2015.

[13] Deane Barker, „Web Content Management“, *O'Reilly Media*, march 2016.

[14] Gaurav Kathuria, „Web Content Management with Documentum“, *Packt Publishing*, june 2006.

[15] Aigars Silkalns, „WordPress Statistics: How Many Websites Use WordPress in 2023?“, Pristupano 29.6.2023. [Na internetu]. Dostupno na: https://colorlib.com/wp/wordpress-statistics/

[16] Sagar Jadhav, „Drupal Vs. WordPress Vs. Adobe Experience Manager: Which CMS Should You Choose?“, Pristupano 29.6.2023. [Na internetu]. Dostupno na: https://www.amuratech.com/blog/cms-comparison-drupal-vs-wordPress-vs-adobe-experience-manager

[17] Adobe Systems, „Develop Websites and Components in Adobe Experience Manager“, 2017

[18] Adobe Experience League, „Introduction to the AEM Architecture Stack“, Pristupano 29.6.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-learn/cloud-service/underlying-technology/introduction-architecture.html?lang=en

[19] Adobe Systems, „Extend and Customize Adobe Experience Manager“, 2017

[20] Adobe Experience League, „Introduction to the Java Content Repository (JCR)“, Pristupano 29.6.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-learn/cloud-service/underlying-technology/introduction-jcr.html?lang=en

[21] The Apache Software Foundation, „Apache Sling - Bringing Back the Fun!“, Pristupano 29.6.2023. [Na internetu]. Dostupno na: https://sling.apache.org/

[22] Adobe Experience League, „Introduction to Sling“, Pristupano 29.6.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-learn/cloud-service/underlying-technology/introduction-sling.html?lang=en
[23] Adobe Experience League, „Introduction to Author and Publish Tier“, Pristupano 11.7.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-learn/cloud-service/underlying-technology/introduction-author-publish.html?lang=en
[24] Adobe Experience League, „Replication“, Pristupano 11.7.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-65/deploying/configuring/replication.html?lang=en
[25] Adobe Experience League, „Dispatcher Overview“, Pristupano 11.7.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-dispatcher/using/dispatcher.html?lang=en
[26] Adobe Experience League, „AEM Project Archetype", Pristupano 20.7.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-core-components/using/developing/archetype/overview.html?lang=en
[27] Adobe Experience League, „Getting Started with HTL", Pristupano 20.7.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-htl/content/getting-started.html?lang=en
[28] Reactjs.org, „React“, Pristupano 22.7.2023. [Na internetu]. Dostupno na: https://react.dev/
[29] Adobe Experience League, „Map SPA components to AEM components", Pristupano 22.7.2023. [Na internetu]. Dostupno na: https://experienceleague.adobe.com/docs/experience-manager-learn/getting-started-with-aem-headless/spa-editor/react/map-components.html?lang=en
[30] Matija Kovaček, „Sample AEM project with React Web Components", GitHub repozitorij, Pristupano 22.7.2023. [Na internetu]. Dostupno na: https://github.com/mkovacek/aem-react-webcomponents
[31] Mozilla, „Web Components", Pristupano 22.7.2023. [Na internetu]. Dostupno na: https://developer.mozilla.org/en-US/docs/Web/API/Web_components
[32] „Ergast Developer API", Pristupano 24.7.2023. [Na internetu]. Dostupno na: http://ergast.com/mrd/

Popis slika

Popis slika treba biti izrađen po uzoru na indeksirani sadržaj, te upućivati na broj stranice na kojoj se slika može pronaći.

Popis tablica

Popis tablica treba biti izrađen po uzoru na indeksirani sadržaj, te upućivati na broj stranice na kojoj se tablica može pronaći.