LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

VIENU LAPU LIETOTŅU IZSTRĀDES IETVARU SALĪDZINĀJUMS

BAKALAURA DARBS

Autors: **Ēriks Žeibe**
Studenta apliecības Nr.: ez18039
Darba vadītājs: prof., Dr.sc.comp. Jānis Zuters

RĪGA 2022

ANOTĀCIJA

Vienu lapu lietotnes tiek plaši lietotas gan kā publiskas vietnes, gan kā uzņēmumu iekšējās lietotnes. Tās sniedz ātru, dinamisku lietotāja pieredzi, un ir pielīdzināmas darbvirsmu lietotnēm. To izstrādei ir pieejami vairāki plaši izmantoti JavaScript ietvari. Bakalaura darba ietvaros tika apskatīti un savā starpā salīdzināti 3 populārākie no tiem - Angular, React, Vue.

Teorētiskajā daļā tika izpētīta vienu lapu lietotņu izstrādes principi, kā arī pētītas un salīdzinātas to izstrādes ietvaru piedāvātās iespējas.

Praktiskajā daļā tika salīdzināti ietvari, ar katru no tiem izstrādājot vienu lapu lietotni. Tie tika salīdzināti pēc vairākiem iepriekš noteiktiem kritērijiem, liekot uzsvaru uz šo ietvaru ātrdarbību.

Atslēgvārdi: Vienu lapu lietotne, Angular, React, Vue, ietvars

ABSTRACT

COMPARISON OF SINGLE PAGE APPLICATION DEVELOPMENT FRAMEWORKS

Single page applications are widely used both as public websites and as internal enterprise applications. They provide a fast, dynamic user experience and are comparable to desktop applications. Several widely used JavaScript frameworks are available for their development. In this bachelor's thesis, the 3 most popular of them were examined and compared with each other – Angular, React, Vue.

In the theoretical part, the principles of single page application development were studied, and the possibilities offered by their development frameworks were studied and compared.

In the practical part, the frameworks were compared, with each of them developing a single page application. They were compared against several pre-defined criteria, with an emphasis on the performance of these frameworks.

Keywords: Single page application, Angular, React, Vue, framework

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

API *(Application Programming Interface)* – programmatūras starpslānis, kas ļauj divām lietotnēm veikt savstarpēju komunikāciju.

CSS *(Cascading Style Sheets)* – valoda, kas apraksta HTML vai XML dokumenta attēlojumu.

DOM *(Document Object Model)* – programmēšanas API priekš tīmekļa dokumentiem, kas definē to struktūru un veidu, kā to var iegūt, manipulēt.

HTML *(Hypertext Markup Language)* – valoda, lai aprakstītu tīmekļa lapu struktūru.

HTTP *(Hypertext Transfer Protocol)* – lietojuma slāņa tīkla protokols, kurš nodrošina tīmekļa dokumentu pārraidi, piemēram, HTML.

JavaScript – programmēšanas valoda, kas tiek plaši lietota tīmekļa izstrādē.

JSON *(JavaScript Object Notation)* – tekstisks datu apmaiņas formāts, kas izmanto JavaScript programmēšanas valodas objektu notāciju.

JSX *(JavaScript XML)* – JavaScript programmēšanas valodas sintaktisks paplašinājums.

MPA *(Multi Page Application)* – tīmekļa vietnes veids, kam ir raksturīga vairāku lapu servēšana no servera.

PWA *(Progressive Web Application)* – tīmekļa lietotnes, kas izmanto vairākas PWA raksturīgas pieejas, lai tās varētu darbināt uz vairākām platformām.

SEO *(Search Engine Optimization)* – tīmekļa vietnes pielāgošanas process ar mērķi ierindot vietni pēc iespējas augstāk meklēšanas dzinējos.

SPA *(Single Page Application)* – tīmekļa vietnes veids, kam ir raksturīga tikai vienas lapas pilna servēšana no tīmekļa servera.

TypeScript – uz JavaScript pamata veidota programmēšanas valoda, kurā ir stingra tipu sistēma.

XSS *(Cross Site Scripting)* – kiberuzbrukuma veids, kurā uzbrucēji tīmekļa vietnēs injicē kaitīgus skriptus, kas var izpildīties uz upuru ierīcēm.

IEVADS

Tīmekļu vietnes ir ikdienas dzīves sastāvdaļa. Caur tām cilvēki iegūst zināšanas, apmainās ar viedokļiem un spēj saņemt dažāda veida pakalpojumus. Tīmekļu vietņu izstrādes pieejām attīstoties, mūsdienās liela daļa vietņu ir pārgājušas no tradicionālās daudzu lapu lietotņu (MPA) pieejas uz vienu lapu lietotņu pieeju (SPA). To var novērot tādās labi zināmās vietnēs kā YouTube, Facebook, Twitter, Google Maps, GMail u.c. Tipiski kā viens no galvenajiem iemesliem šādai izvēlei ir tas, ka SPA spēj nodrošināt darbvirsmu lietotnēm līdzīgu lietotāja pieredzi - dinamisku, gludu un stabilu.

SPA popularitātes rezultātā ir izveidoti un attīstījušies vairāki rīki, ar kuriem var izstrādāt šīs vietnes. Mūsdienās īpašu popularitāti ir ieguvuši tādi SPA ietvari kā Angular, React un Vue. Ar visiem trim var izstrādāt kvalitatīvas SPA, tomēr starp to lietošanu var būt vairākas atšķirības. Tā kā izstrādājot jaunu vietni ir jāpieņem lēmums, kādu ietvaru izmantot, šajā darbā tiks izpētītas katra ietvara iespējas un starp tiem tiks veikts salīdzinājums.

Bakalaura darba mērķis ir izpētīt populārākos JavaScript ietvarus, ar kuriem var izstrādāt SPA, kā arī veikt to savstarpēju salīdzinājumu. Lai to paveiktu tika izvirzīti sekojoši uzdevumi:

- Izpētīt SPA pamatprincipus, priekšrocības un trūkumus.
- Izpētīt un salīdzināt populārāko SPA ietvaru iespējas.
- Veikt praktisku SPA ietvaru ātrdarbības salīdzinājumu pēc vairākiem kritērijiem.
- Salīdzināt iegūtos rezultātus ar publiski pieejamiem salīdzinājumiem.

Darbs sastāv no trīs nodaļām. Pirmajā nodaļā tiek apskatīti SPA darbības principi, kā arī apkopotas to lietošanas priekšrocības un trūkumi. Otrajā nodaļā tiek aprakstītas SPA ietvaru piedāvātās iespējas un tiek veikts savstarpējs salīdzinājums. Trešajā nodaļā tiek veikts praktisks ietvaru salīdzinājums, liekot uzsvaru uz ātrdarbību.

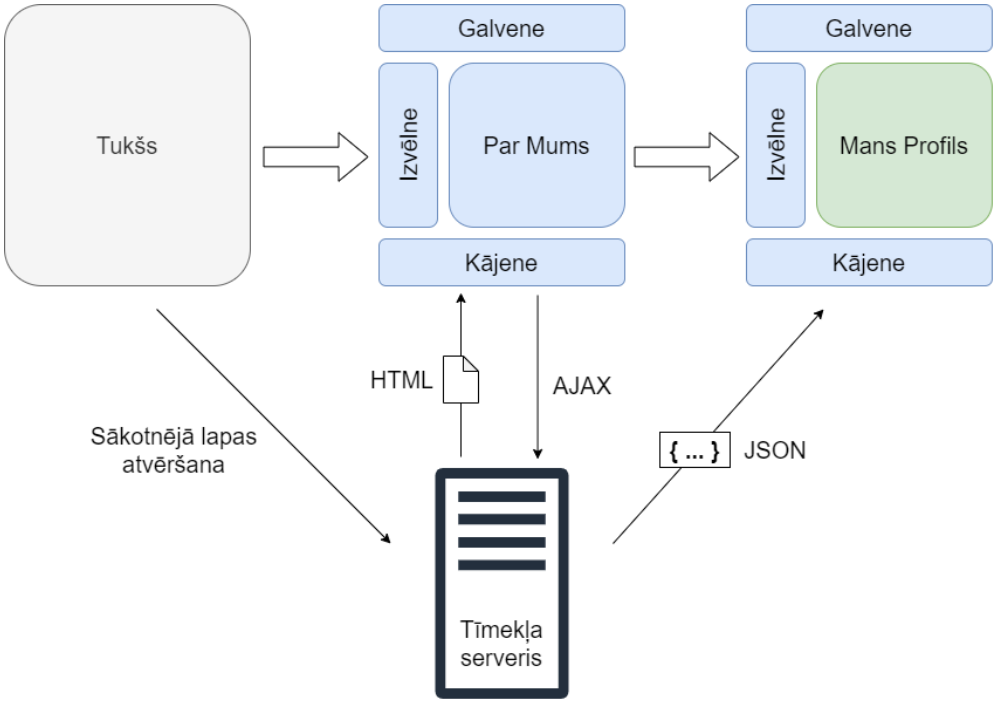
1. VIENU LAPU LIETOTNE

Vienu lapu lietotne ir tīmekļa vietne, kas tās lietošanas gaitā neveic lapas pārlādi atšķirībā no tipiskajām daudzu lapu lietotnēm. Kā visas lietotnes, tās mērķis ir palīdzēt lietotājam veikt kādu uzdevumu, piemēram, uzrakstīt dokumentu, pārvaldīt tīmekļa serveri. SPA var iedomāties kā smagnēju klientu, kas vienu reizi tiek lejupielādēts no tīmekļa servera [21].

Tehniski SPA ir tīmekļa vietne, kas mijiedarbojas ar lietotāju dinamiski pārrakstot tās saturu, izmantojot JavaScript, tā vietā, lai veiktu jaunas lapas ielādi no servera. Šāda pieeja novērš lietotāja darbības pārtraukumus, kas notiek lapu pārlādes brīžos tādā veidā panākot darbvirsmu lietotnēm līdzīgāku pieredzi [22].

1.1. Darbības princips

Atverot SPA vietni, pirmais solis ir līdzīgs kā jebkurai citai vietnei – tīmekļa serverim tiek pieprasīts *hypertext markup language* (HTML) dokuments, kuru pārlūks saņem un attēlo. Atšķirība rodas, lietotājam darbojoties ar šo vietni. Veicot tajā jebkādas darbības – apmeklējot citas vietnes sadaļas, filtrējot preces, rediģējot savu profilu, SPA vietne vairs nepieprasa jaunu HTML dokumentu, bet gan tikai nepieciešamos *JavaScript object notation* (JSON) datus, kurus SPA vietnes JavaScript spēj pats apstrādāt un atjaunot nepieciešamās HTML dokumenta sekcijas (skat. 1.1. att.). Tādā veidā HTML dokumenta atjaunošana nonāk pārlūka ziņā un tas vairs netiek pārlādēts kopš vietnes atvēršanas.



1.1. att. **SPA darbības principa shēma**

1.2. Priekšrocības

Salīdzinājumā ar tipiskajām MPA vietnēm, SPA sniedz sekojošas priekšrocības:

- **Laba veiktspēja un gluda lietotāja pieredze** – lai izpildītu lietotāja darbību, SPA ielādē mazu JSON failu, tā vietā, lai pārlādētu pilnu lapu. Tas uzlabo vietnes veiktspēju sniedzot lietotājam darbvirsmas lietotnei līdzīgu lietotāja pieredzi.
- **Laba Stabilitāte** – Lietotnes pamatfunkcijas tipiski tiek pilnībā ielādētas tās atvēršanas brīdī. Tas ļauj veikt vairums darbību arī slikta interneta savienojuma gadījumā. Situācijā, ja savienojums ir zudis, lietotnei ir iespēja lietotājam draudzīgā veidā to paziņot un ļaut veikt citas funkcijas, tā vietā, lai neveiksmīgi veiktu lapas pārlādi.
- **Iespēja veikt paralēlu izstrādi** – Izstrādājot SPA, ir iespējams sadalīt priekšgala un aizmugurgala sistēmu izstrādi 2 atsevišķām izstrādātāju grupām. Tā kā aizmugurgala sistēma atgriež tikai JSON datus, tad pēc vienošanās par savstarpējo *application programming interface* (API), iespējams neatkarīgi klienta pusē izstrādāt SPA vietni, tās attēlojumu un neatkarīgi servera pusē izstrādāt datu glabāšanu, apstrādi un atgriešanu JSON formātā.
- **Atbalsts mobilai izstrādei** – SPA vietnes aizmugursistēma var tikt izmantota, lai izstrādātu mobilās lietotnes versiju šai vietnei. Tas ir tādēļ, ka aizmugursistēma atgriež tikai JSON datus, nevis tīmekļa vietnēm specifisku HTML saturu [23].

1.3. Trūkumi

SPA vietnēm ir arī savi trūkumi, ar kuriem jārēķinās, plānojot izstrādāt jaunu lietotni:

- **Slikta vietnes meklētājdzinēja optimizācija (SEO)** – Viena no metrikām, ko meklēšanas dzinēji izmanto kārtojot vietnes ir lapu skaits vietnē. Tā kā SPA satur tikai 1 lapu, tas var negatīvi ietekmēt vietnes attēlošanu meklēšanas dzinēju rezultātos [24]. Lapas analīzi arī traucē fakts, ka, lai iegūtu SPA vietnes saturu, vispirms ir jāizpilda tajā esošais JavaScript. Šobrīd to spēj paveikt tikai Google lapu analizētājs *(web crawler)*. Ir veidi, kā SPA vietnēm var uzlabot SEO, piemēram, ar servera puses renderēšanu, izomorfu JavaScript, pirms-renderēšanu, tomēr tas prasa papildus laiku un zināšanas [25].
- **Liels pārlūka resursu patēriņš** – Tā kā SPA vietnes dinamiskumu nodrošina JavaScript, tad ir lielāka slodze uz izmantoto pārlūku. SPA bieži ir nepieciešamas jaunākās pārlūku versijas, kā arī vietne var strādāt lēnāk uz vecākām mobilajām ierīcēm, kurām būtu grūtības apstrādāt iekļauto JavaScript.

- **Drošības riski** – Salīdzinājumā ar MPA, SPA ir lielāks mērķis starplapu skriptēšanas uzbrukumiem (XSS), kas ļauj lietotnē injicēt kaitīgu pirmkodu. Šo draudu ir iespējams mazināt, izmantojot kādu no SPA ietvariem, kuros ir iebūvētas aizsardzība pret šo uzbrukumu.

1.4. Klienta bāzēta maršrutēšana

Neskatoties uz to, ka SPA pēc definīcijas satur tikai 1 lapu, ir veids, kā panākt, lai lietotājs saņemtu vairāku lapu lietotnes pieredzi – klientu bāzēta maršrutēšana.

Tīmekļa vietnes sastāv no vairākām lapām, starp kurām lietotājs var pārslēgties. Maršrutēšana ir mehānisms, kas nodrošina atbilstošas lapas ielādi atkarībā no apmeklētās saites. Tipiski tiek izmantota servera bāzēta maršrutēšana. Ar tās palīdzību, lietotājam noklikšķinot uz saites vai ievadot to manuāli pārlūkā, tiek pieprasīta un ielādēta jauna lapa no servera. Šāda pieeja ir pretrunā ar SPA galveno ideju – pilna lapas ielāde notiek tikai vienu reizi. Tādēļ SPA lapās tiek izmantota klienta bāzēta maršrutēšana.

Klienta bāzētas maršrutēšanas pamatideja ir, ka par lapu maršrutēšanu parūpējas klienta puse jeb lietotāja pārlūkprogramma. Piemēram, lietotājam apmeklējot mājaslapas sadaļu “Par mums”, tā vietā, lai ielādētu jaunu lapu no servera, tiek izpildīts JavaScript kods, kurš asinhroni iegūst un atjauno lapas saturu. Rezultātā, tā vietā, lai pārlādētu visu vietni, tiek ielādēts un atjaunots tikai “Par mums” saistītais saturs. Lai lietotājam vienkāršotu uztveri, caur JavaScript tiek mākslīgi nomainīta saite, kas rādās pārlūkā, izmantojot pārlūkos iebūvēto History API [1, 2].

2. VIENU LAPU LIETOTŅU IZSTRĀDES IETVARI

2.1. Angular

Angular ir atvērta koda TypeScript bāzēta tīmekļa lietotņu izstrādes platforma, kuras izstrādi vada Google kompānijas Angular komanda kopā ar komūnu ar atsevišķiem indivīdiem un kompānijām. Projekta pirmkods ir pieejams GitHub un tā lejupielādi iespējams veikt no *node package manager* (NPM) repozitorija. Ietvara dokumentācija un projekta informācija ir pieejama Angular oficiālajā mājaslapā angular.io, kur ir arī pieejamas noderīgas norādes uz saistītajām bibliotēkām un mācību resursiem.

Angular pirmo versiju AngularJS izstrādāja 2010. gadā Google darbinieks Miško Heverijs. Tas ir uz Javascript bāzēts ietvars, kas stipri atšķiras no nākamajām Angular 2+ versijām. Kā dažas no atšķirībām pieminamas, ka AngularJS izmanto *model view controller* (MVC) nevis komponenšu un direktīvu arhitektūru, neatbalsta mobilās ierīces, kā arī ir mazāk advancēta atkarību injicēšana [8]. Tīmekļa izstrādei attīstoties, AngularJS radās grūtības tam pielāgoties, tādēļ 2016. gadā tas tika pārrakstīts uz jaunu versiju Angular 2, uz kura pamata tika attīstītas pārējās Angular 2+ versijas. Šajā darbā tiks apskatītas tieši Angular 2+ versijas.

Angular platformā ietilpst:

- Komponenšu bāzēts ietvars priekš mērogojamu tīmekļa lietotņu izstrādes.
- Integrētas bibliotēkas, kas sniedz papildus iespējas, piemēram, maršrutēšanu, formu pārvaldību u.c.
- Izstrādes rīki, kas palīdz izstrādāt, kompilēt, testēt un mainīt pirmkodu [7].

2.1.1. Ietvara struktūra

Izstrādājot Angular lietotni, tai ir iespējams izstrādāt vairākas atsevišķas daļas, kas ir savā starpā saistītas, veidojot vienotu lietotni (skat. 2.1. att.).

Būtiskākā Angular arhitektūras sastāvdaļa ir komponente. Izstrādātās komponentes ir iespējams grupēt kokveida struktūrā veidojot Angular lietotni. Katra komponente ir cieši saistīta ar veidni.

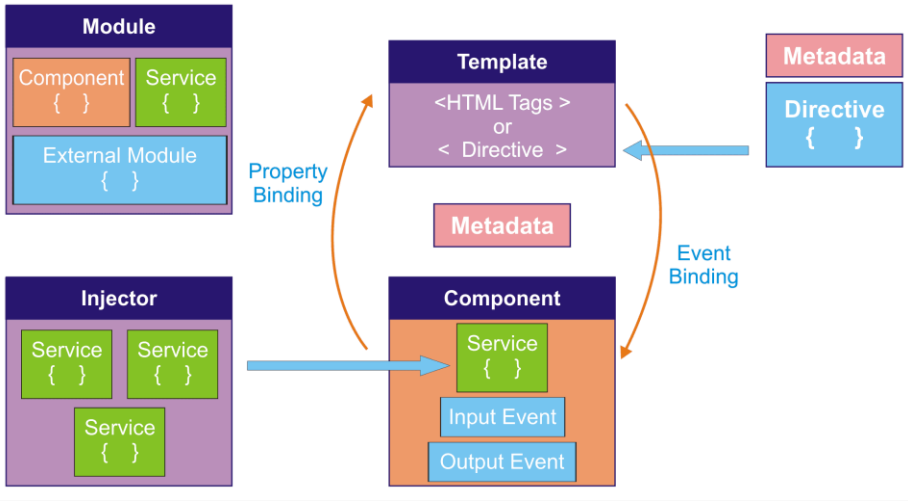
Veidnē ar īpaši papildinātu HTML tiek aprakstīts, kā komponente ir jāattēlo pārlūkā. No veidnes ir iespējams piekļūt komponentē implementētajai funkcionalitātei un mainīgajiem.

Lai paplašinātu veidņu definēšanas funkcionalitāti, ietvars piedāvā izstrādājam iespēju veidot sevis definētas direktīvas papildus jau iebūvētajām. Tās ir izmantojamas papildus HTML birku vai atribūtu veidā.

Lai starp komponentēm varētu dalīties ar kopīgu funkcionalitāti, ietvars piedāvā izstrādāt Angular servisus. Viens no tipiskākajiem pielietojumiem ir veidot servisu kā API uz ārēju tīmekļa servisu. Ietvars ļauj komponentēm piekļūt pie šiem servisiem izmantojot atkarību injekciju – iespējams definēt kurā komponentē un kā tiek izveidota servisa instance, kā rezultātā apakškomponentes spēj tālāk izmantot to pašu definēto instanci.

Lai organizētu lietotnē izstrādātās daļas – komponentes, servisus, direktīvas, Angular piedāvā komplektēt šis daļas pa moduļiem, kas veidotu Angular lietotni. Modulī var importēt citus moduļus, deklarēt komponentes un direktīvas, kas pieder modulim, kā arī norādīt izveidojamos Angular servisus.

Tālākās nodaļās tiks sīkāk aprakstītas šīs Angular lietotnes daļas.



2.1. att. **Angular lietotnes arhitektūra [33]**

2.1.2. Komponentes

Komponentes ir Angular lietotnes būvēšanas bloki. Visa Angular lietotne ir uztverama kā komponenšu koks, kas sastāv no atkārtoti izmantojamām komponentēm. Vienas komponentes definīcijā ietilpst:

- TypeScript klase ar komponentes implementēto loģiku.
- HTML veidne, kurā ir pieeja komponentes loģikai.
- CSS stils.

Komponentes klasei ir jāpievieno dekorators, lai sasaistītu šos failus un lai norādītu papildus komponentes īpašības, kā piemēram, nosaukumu, ar kuru komponenti iespējams izmantot citās veidnēs.

Katrai komponentes instancei ir dzīvescikls, kas sākas ar brīdi, kad Angular izveido instanci un uzrenderē komponenti uz ekrāna un beidzot ar brīdi, kad komponente vairs netiek attēlota un instance tiek izdzēsta. Ietvars piedāvā veidu, kā izstrādātājs var reaģēt uz šiem brīžiem. To var izdarīt komponentes klasei implementējot nepieciešamo dzīvescikla interfeisu, lai varētu atbilstošā funkcijā rakstītu kodu, kurš izpildīsies tieši dzīvescikla posma iestāšanās brīdī. Tas noder piemēram gadījumos, kad jālikvidē mainīgo instances, likvidējot komponenti.

Komponentēm ir iespējams definēt ievades un izvades mainīgos, kas nosaka to, kā notiek datu apmaiņa, izmantojot šo komponenti.

2.1.3. Veidnes

Veidne ir HTML teksts, ar kuru katrai komponentei tiek definēts attēlojamais skats. No veidnes iespējams piekļūt komponentes mainīgajiem, veidojot vienvirziena vai divvirziena saites, kā arī tajās iespējams izpildīt JavaScript izteiksmes. Veidnēs var izmantot citas Angular komponentes HTML elementu veidā. Papildus HTML iespējām, veidni var definēt izmantojot direktīvas.

2.1.4. Direktīvas

Direktīvas ir īpaši definētas klases, kuras iespējams izmantot veidnēs, lai papildus izmainītu veidnes elementu uzvedību. Ir iespējams izmantot gan iebūvētās direktīvas, gan pašam tās definēt. Direktīvas iedalās 3 veidos:

- Komponentes – tipiskākais direktīvas veids. Veidnē komponentes ir izmantojamas īpašu birku veidā.
- Atribūtu direktīvas – izmaina HTML elementu uzvedību, pieliekot tiem direktīvu atribūta veidā. Piemēram, ar NgModel direktīvu iespējams izveidot 2 virzienu saiti starp formas elementu un komponentes mainīgo.
- Strukturālās direktīvas – ar to palīdzību iespējams pārveidot *document object model* (DOM) struktūru, pievienojot, noņemot vai izmainot elementus, kuriem direktīva ir pievienota. Populārākās no iebūvētajām direktīvām ir NgIf, NgFor, NgSwitch, kas ļauj veidnēs izmantot tipiskās programmēšanas valodu konstrukcijas.

2.1.5. Atkarību injekcija

Atkarības Angular kontekstā ir objekti, kuru sauc par servisiem, kas klasei ir nepieciešami, lai veiktu kādu funkcionalitāti. Angular piedāvā vairākus iebūvētos servisus, kā piemēram, HttpClient priekš *hypertext transfer protocol* (HTTP) pieprasījumu veikšanas. Izstrādātājam ir arī iespējams definēt savus injicējamos servisus. Tas ir izdarāms izveidojot klasi un tai pievienojot "@Injectable" dekoratoru. Lai servisus varētu izmantot citās komponentēs, to konstruktora parametros nepieciešams norādīt injicējamos servisus. Rezultātā, brīdī, kad Angular ietvars veido jaunu komponentes instanci, atbilstoši konstruktora parametriem tiek atrastas un injicētas norādītās servisu instances.

2.1.6. Formas

Formu apstrāde ir daudzu tīmekļa lietotņu pamatā. Tās ļauj lietotājiem ielogoties, ļauj ievadīt piegādes datus veicot pasūtījumus u.c. Angular ietvars piedāvā 2 dažādus veidus, kā atvieglot formu pārvaldību un validāciju. Abu veidi izmanto principu, ka uz formu pamata tiek izveidoti datu un formu modeļi, pie kuriem komponentē ir piekļuve, lai manipulētu un iegūtu informāciju par formām.

Veidņu-bāzētas formas – formas un to lauki tiek definēti veidnē un tās tiek sasaistītas ar komponentes mainīgajiem, izmantojot NgModel direktīvu priekš tālākām darbībām. Šis formu definēšanas veids ir paredzēts vienkāršākiem gadījumiem.

Reaktīvās formas – formu modelis izstrādātājam ir jāizveido komponentē, pēc tam veidnē piesaistot izveidoto formu modeli. Šāda pieeja ir stabilāka un dod plašākas iespējas manipulēt formu modeli.

2.1.7. Maršrutēšana

Lai Angular lietotnēs varētu īstenot klientu bāzēto maršrutēšanu, ietvarā ir iebūvēts atsevišķs modulis šim mērķim RouterModule. Tā izmantošanas sagataves iespējams automātiski pievienot, veidojot jaunu projektu caur Angular *command line interface* (CLI). Izmantojot maršrutēšanu, atsevišķā modulī (tipiski AppRoutingModule) nepieciešams konfigurēt maršrutēšanu un to ceļus. Vienkāršākais veids maršrutu definēšanai ir *uniform resource locator* (URL) norādīšana un komponentes nosaukuma norādīšana, kuru ietvaram jāattēlo, kad tiek apmeklēts URL (skat. 2.2. att.).

```
const routes: Routes = [
  { path: 'about-component', component: AboutComponent },
  { path: 'posts-component', component: PostsComponent },
  { path: 'contacts-component', component: ContactsComponent }
];
```

2.2. att. **Angular maršrutēšanas definēšana**

Tālāk saknes modulī ir jāimportē izveidotais AppRoutingModule un tas dod iespēju veidnēs veidot hipersaites uz vēlamo komponenti, kā arī veidnē var norādīt pozīciju, kurā tiks ģenerēts apmeklētās komponentes saturs.

Šādā veidā Angular ietvarā iespējams pilnvērtīgi veikt klienta bāzēto maršrutēšanu.

2.2. React

React ir Javascript bibliotēka priekš dinamisku tīmekļa lietotņu izstrādes. Tā koncentrējas uz deklaratīvu un komponenšu bāzētu lietotāju saskarņu būvēšanu. Atbilstoši Google Trends rezultātiem, React ir populārākais starp darbā apskatītajiem ietvariem.

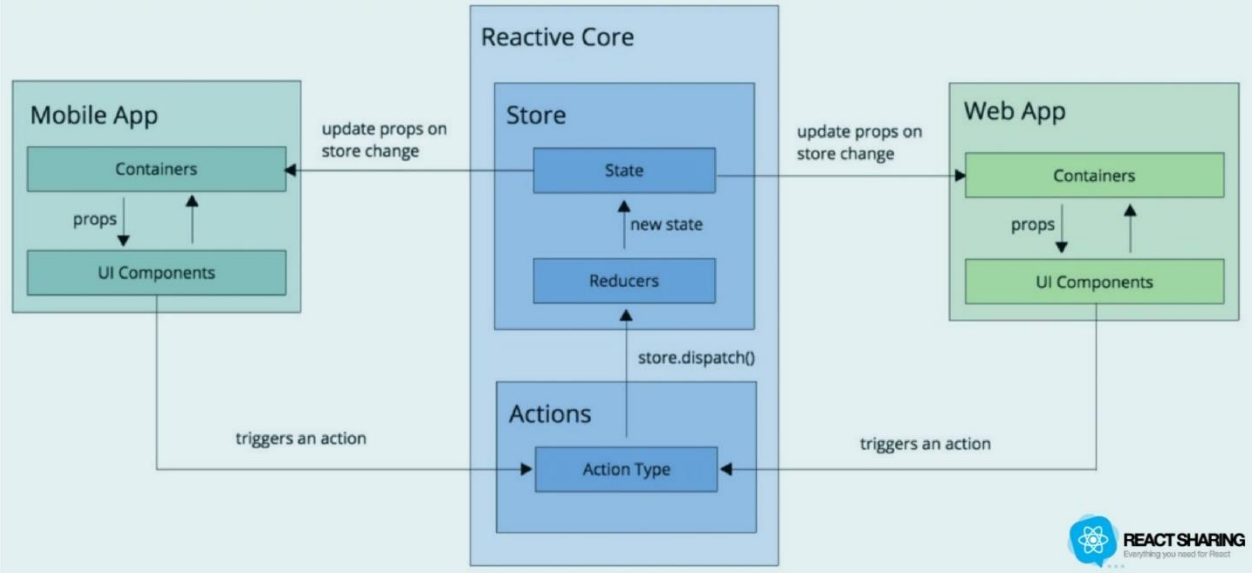
Pirmo React prototipu "FaxJS" izstrādāja 2011. gadā Facebook darbinieks *Jordan Walke*. Sākotnēji ietvars tika izmantots uzņēmuma iekšienē, lai padarītu Facebook Ads sistēmu vieglāk pārvaldāmu. 2012. gadā Facebook īpašumā nonāca Instagram, kurš vēlējās izmantot Facebook jauno tehnoloģiju. Lai to varētu izdarīt, Facebook padarīja ievaru publiski pieejamu, kā rezultātā 2013. gadā React kļuva pieejams jebkuram izstrādātājam [9].

Tālākajos gados ietvara popularitāte un iespējas turpināja augt. 2015. gadā tika izlaists React Native – tehnoloģija, kas ļauj izstrādāt dzimtās mobilās lietotnes populārākajām OS, izmantojot React. Kā vēl viens būtisks papildinājums pieminams 2019. gadā izlaistā React 16.8 versija, ar kuru tika ieviests React Hooks koncepts, kas ļauj veidot komponentes sīkākās, vieglāk pārvaldāmās daļās, uzlabojot lietotnes uzturēšanu [10].

2.2.1. Ietvara struktūra

Salīdzinājumā ar Angular, React lietotnei ir raksturīga vienkāršāka struktūra. 2.3. attēlā ir apskatāma struktūra tīmekļa un mobilai React lietotnei, kas izmanto stāvokļu pārvaldības bibliotēku Redux. Vienkāršākajā gadījumā React lietotne sastāv tikai komponentēm, kuras satur HTML veidni ar loģiku un kuras var grupēt kokveida struktūrā veidojot lietotni. Tas notiek veidnēs atkārtoti izmantojot kādu citu komponenti un nododot tai ievaddatus caur HTML atribūtiem. Shēmā tas ir redzams "Web App" apgabalā, kur ir attēlota komponenšu savstarpējā komunikācija. No augškomponentes iegūtos datus React lietotnēs sauc par *props*.

Sarežģītākās React lietotnēs, kurām ir nepieciešams centralizēti pārvaldīt tās stāvokli, lai samazinātu tiešu datu apmaiņu starp komponentēm, izmanto kādu no stāvokļu pārvaldības bibliotēkām, kā piemēram, Redux. Shēmas vidējā daļā ir attēlota Redux stāvokļa pārvaldības sastāvdaļas. Tā pamatā ir stāvoklis, kas ir JavaScript objekts, kurš raksturo lietotnes stāvokli. Lai to izmanītu, ir jāizstrādā vairākas *reducer* funkcijas, kas to spēj paveikt. Redux nodrošina iespēju kā no jebkuras komponentes var piekļūt lietotnes stāvoklim. Komponentēm ir iespēja tos izmainīt izsaucot iepriekš definētus darbību tipus, kas tālāk izsauks *reducer* funkciju, izmainot lietotnes stāvokli.



2.3. att. **React Redux lietotnes arhitektūra [34]**

2.2.2. JSX

JSX (JavaScript XML) ir Facebook izstrādāts ECMAScript strandarta paplašinājums ar primāro mērķi atvieglot rakstīt React komponenšu HTML skatus. Tā vietā, lai React komponentē būtu jādefinē UI skats kā kokveida objekts, izstrādātājam ir iespēja izmantot plaši pazīstamo HTML sintaksi ar nelieliem pielāgojumiem, lai iegūtu to pašu efektu. 2.4. attēlā ir apskatāma atšķirība starp UI skata izveidi JSX un JS pieejā. Lai pārlūkprogrammas spētu izpildīt JSX kodu, projekta kompilēšanas brīdī tas tiek transpilēts uz JS kodu [11].

```jsx
const elementJsx = (
  <h1 className="greeting">
    Hello, world!
  </h1>
);

const elementJs = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
);
```

2.4. att. **JSX pirmkoda piemērs**

Lai gan JSX stipri līdzinās HTML kodam, tam ir veikti vairāki pielāgojumi, lai to varētu izmantot kā veidni un varētu labāk sasaistīt React komponentes loģiku. Kā svarīgākie no tiem ir pieminami:

- Iespēja ievietot JS izteiksmes un mainīgos.
- Iespēja izmantot citas izveidotās React komponentes īpašu birku veidā.
- HTML "class" atribūts ir pārsaukts par "className", jo JS valodā jau tiek izmantots vārds "class".

JSX izmantošana arī aizsargā pret injekciju uzbrukumiem. Rezultējošā HTML renderēšanas procesā visu JSX ievietoto JS mainīgo un izteiksmju vērtības tiek pārvērstas par tekstu, tādā veidā nedodot iespēju injicēt kaitīgu kodu un nodrošinot aizsardzību pret XSS.

2.2.3. Komponentes

Komponentes ir React lietotnes būvēšanas bloki, kurus izstrādātājs var atsevišķi izstrādāt, ievērojot *separation of concerns* un strukturēt tās kopā kokveida struktūrā. Komponentes galvenais uzdevums ir atgriezt UI skatu, kas tipiski tiek definēts ar JSX palīdzību.

React komponentēm ir raksturīga vienvirziena datu plūsma – komponentēm ir iespēja nodot datus apakškomponentēm, bet pretējā virzienā tiešā veidā datus nevar padot. No augškomponentes iegūtie dati komponentēm ir pieejami mainīgajā *props*, kas ir pieejams kā konstruktora parametrs. Tādā veidā iespējams, piemēram, komponentei "Preču saraksts" nodot attēlojamās preces datus komponentei "Prece".

Komponentēs iespējams uzturēt savu iekšējo stāvokli *state* kā JavaScript objektu. Lietotājam darbojoties ar lietotni, var mainīties lietotnes stāvoklis. Šīs stāvokļa izmaiņas iespējams piefiksēt komponentes iekšējā stāvoklī un šo informāciju izmantot, lai atgrieztu izmainītu UI skatu.

Piemēram, ieķeksējot lauku “Piekrītu noteikumiem”, komponentē “Pasūtījuma forma” iespējams izmainīt tās stāvokli un tad atgriežot komponentes skatu, iespējams atbilstoši iespējot vai atspējot “Pasūtīt” pogu.

Veidojot sarežģītākas komponentes, kas izmanto datu ielādi, trešo pušu bibliotēkas un citas īpašības, var rasties nepieciešamība izmantot komponentes dzīvescikla piedāvātās iespējas. Komponentes dzīves ciklā, sākot ar tās izveidi un beidzot ar tās likvidēšanu, ietvars izsauc virkni funkciju. Komponentēs ir iespējams implementēt šīs funkcijas, lai koda gabals izpildītos nepieciešamajā brīdī. Piemēram, komponentē “Preču saraksts” nepieciešams ielādēt preču datus brīdī, kad komponente atveras. Klašu komponentes gadījumā to ir iespējams izdarīt veicot datu ielādi funkcijā *componentWillMount()*, kuru ietvars izsauc tieši pēc komponentes inicializācijas.

2.2.4. Hooks

Sākot ar React 16.8 versiju, ietvaram tika pievienots būtisks papildinājums – Hooks. Tas ļauj izmantot stāvokli un citas React īpašības bez klases veidošanas, padarot komponenšu pierakstīšanu kompaktāku un vieglāk pārvaldāmu [12]. Rezultātā komponentes ir iespējams pilnvērtīgi pierakstīt kā funkcijas. Hooks ir pieejamas kā JavaScript funkcijas, kuru sākumā ir pierakstīts vārds “use”. Dažas no populārākajām Hooks funkcijām:

- *useState()* – Ļauj pārvaldīt stāvokli funkcionālajās komponentēs. Funkcija atgriež pāri, kas satur stāvokļa mainīgo kopā ar tā uzstādīšanas funkciju. Mainīgo nav ieteicams tiešā veidā izmainīt, jo tad React dzinējs nebūs informēts par nepieciešamību pārrenderēt UI skatu.
- *useEffect()* – Sniedz iespēju izpildīt komandas brīžos, kad tiek komponente tiek pārrenderēta vai tiek pirmo reizi inicializēta. Šis Hook aizstāj vairākas dzīvescikla funkcijas, kuras ir pieejamas klašu komponentēs.

2.3. Vue

Vue ir atvērta pirmkoda JavaScript bibliotēka, kas paredzēta lietotāju saskarņu būvēšanai. Tas tika izlaists 2014. gadā un tā autors ir bijušais Google darbinieks Evans Ju. Autora sākotnējā motivācija bija radīt Angular ietvara alternatīvu, kas būtu “viegls” *(lightweight)*, bet tai pašā laikā saglabātu galvenās ietvara īpašības, kā piemēram, datu saites [3]. Rezultātā tika radīts progresīvs ietvars, kurš ir adaptīvs konkrētā projekta sarežģītībai.

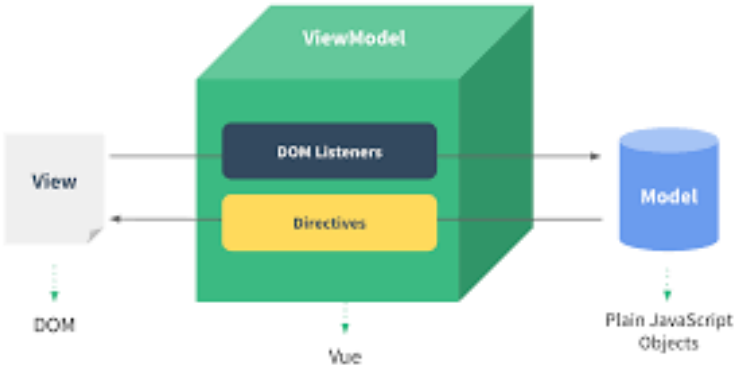
Atvieglotai Vue apguvei, ir pieejama vietne vuemastery.com, kurā ietvara autori ir sagatavojuši video materiālus un “špikera” lapas. Vue galvenajā vietnē ir arī saite uz vueschool.io,

kurā ir izvietoti video materiāli ietvara apguvei. Mājaslapā ir pieejama interaktīva vide ietvara pamatu apguvei.

Šajā nodaļā tiks apskatīts Vue 3 – jaunākā no Vue versijām.

2.3.1. Ietvara struktūra

Līdzīgi kā React ietvars, Vue ietvara pamatā ir komponentes, kas satur to definējošu veidni un loģiku. Tās var strukturēt un vienu otrā izmantot HTML birku veidā. 2.5. attēlā ir apskatāmas galvenās komponentes sastāvdaļas – skats/veidne, kā arī komponentes objektu un funkciju modelis. Vue ietvars nodrošina DOM un komponentē definētās loģikas izpildi. Vue ietvarā ir dažas līdzības no Angular piedāvātajām iespējām. Tajā arī tiek izmantotas direktīvas un izstrādātājam ir iespēja izstrādāt savas direktīvas. Lai starp komponentēm varētu atkārtoti izmantot loģiku, ietvars piedāvā veidot *composables*, kas ir pielīdzināms React ietvara piedāvātajiem *custom hooks*.



2.5. att. **Vue lietotnes komponentes arhitektūra [35]**

2.3.2. Komponenšu rakstīšanas pieejas

Komponentes ir Vue lietotņu būvēšanas bloki. To izveidei ietvars piedāvā 2 pieejas – Composition API un Options API.

Options API pieeja balstās uz ideju "komponentes instance", kas ir vieglāk uztverami izstrādātājiem ar iepriekšējām *object oriented programming* (OOP) zināšanām. Pieeja ir arī iesācējiem draudzīgāka, jo tiek apslēptas reaktivitātes detaļas, organizējot komponenti kā opciju grupas.

Composition API pieeja balstās uz reaktīvu stāvokļu manīgo definēšanu vienas funkcijas apgabalā. Sarežģītākiem izstrādes gadījumiem, lietotnes stāvoklis tiek veidots no vairākām funkcijām, kurās ir reaktīvie mainīgie. Šī izstrādes pieeja ir brīvākā formā un tās elastība ļauj labāk

organizēt funkcionalitāti un atkārtoti izmantot loģiku. Tomēr šī pieeja ir iesācējiem mazāk draudzīga, jo efektīvai izmantošanai, ir nepieciešams izprast, kā strādā Vue reaktivitāte [4].

2.3.3. Viena faila komponente

Viena faila komponentes (SFC) ir īpašs faila formāts ar .vue paplašinājumu, kurš ļauj izstrādāt komponentes vienā failā – tajā ietilpst gan HTML veidne, gan veidnes stili, gan pašas komponentes JavaScript loģika. Fails ir atbilstoši sadalīts 3 sadaļās. Šāda komponenšu komplektēšana dod sekojošas priekšrocības:

- Izstrādātājs var kompakti organizēt komponentes.
- Pēc būtības saistītu rūpju *(concerns)* apvienošana, ļaujot vieglāk pārskatīt komponentes.
- Veidņu pirms-kompilēšana, paātrinot lietotnes ātrdarbību.
- Ērtāka sintakse, strādājot ar Composition API pieejas komponentēm.
- Vairāk kompilēšanas brīža optimizāciju, analizējot veidni un skriptu.
- IDE atbalsts ar koda pabeigšanu un tipu pārbaudi priekš veidņu rakstīšanas.
- *Hot-Module Replacement* atbalsts, paātrinot izstrādes laiku ar dinamisku komponenšu pārlādi to izmaiņu gadījumā.

Vue piedāvā arī iespēju rakstīt komponentes neizmantojot SFC, bet gan tīrā JavaScript bez būvēšanas soļa. Tas noder gadījumos, kad Vue tiek minimāli izmantots, lai uzlabotu statisku HTML ar vienkāršu interaktivitāti. Šajā gadījumā papildus būvēšanas solis un piedāvātās priekšrocības var būt liekas [5].

2.3.4. Nuxt.js

Nuxt.js ir atsevišķs ietvars priekš Vue.js lietotņu izstrādes. Tā mērķis ir ļaut Vue izstrādātājiem izmantot jaunākās mājaslapu tehnoloģijas ātrā un vienkāršā veidā. Piedāvātajās iespējās ietilpst servera puses renderēšana (SSR), statisku vietņu ģenerēšana (SSG), koda dalīšana *(code-splitting)* u.c. Tās ir iespējams ieslēgt ar konfigurācijas palīdzību tā vietā, lai katru no tām ieviestu atsevišķi [6]. Ietvars arī atvieglo izstrādātājam darbu ar Vue ietvaru, piemēram, automātiski veicot atkarību *(dependency)* importēšanu, rezultātā tas nav jāveic faila sākumā. Tālāk ir sīkāk aprakstītas tehnoloģiju piedāvātās priekšrocības.

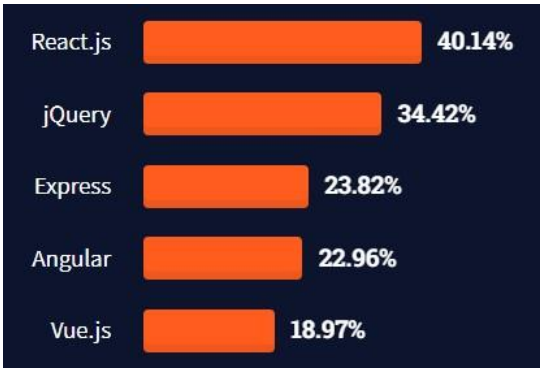
- **SSR** – Servera pusē veic HTML dokumenta renderēšanu tā vietā, lai to veiktu klients. Rezultātā tiek risināta SPA lietotnēm raksturīgā SEO problēma, jo, atverot vietni, ir pieejams nolasāms HTML. Vēl tiek sniegta arī ātrāka lapas atvēršana gadījumos, kad ir lēns interneta savienojums, jo nav jāpārraida renderēšanas JavaScript.

- **SSG** – Šī ir uzlabota pieeja SSR pieejai. Gadījumos, ja lapa ir statiska un tās renderēšanai dati ir zināmi un nemainīgi, tad SSR vietā to ir iespējams vienu reizi servera pusē renderēt un uz lietotāju pieprasījumiem atbildēt ar statisku HTML saturu. Tas papildus uzlabo lapas ielādes ātrumu. Šī pieeja var būt īpaši noderīga satura balstītām vietnēm, piemēram, dokumentāciju, blogu vietnēm [31].
- **Koda dalīšana** – Dod iespēju kompilēšanas rezultātā lietotnes JavaScript failu sadalīt vairākos failos tā, lai atverot lietotni, tiktu lejupielādēti tikai tajā brīdī nepieciešamie faili, paātrinot lapas ielādes laiku. Nuxt.js jau noklusēti nodrošina koda dalīšanu katrai lietotnes lapai. lietotne. Sadala JS kodu vairākos failus – lai ātrāks un lētāks klientiem. Tādā veidā ielādējot lapu, ielādējas tikai JavaScript faili/*bundles* ar nepieciešamajām komponentēm [32].

2.4. Salīdzinājums

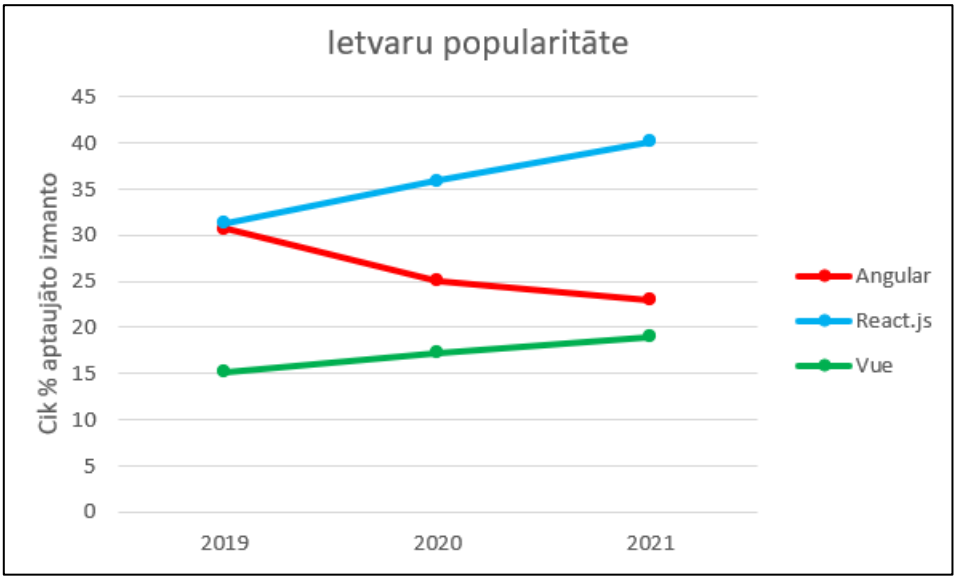
2.4.1. Popularitāte

IT interesentu vidū plaši pazīstamā jautājumu un atbilžu vietne Stack Overflow ik gadu aptaujā izstrādātājus par populārākajām tehnoloģijām, par izstrādātāju darba vidi u.c [13]. 2021. gadā tika aptaujāti 80’000 izstrādātāji no 180 dažādām valstīm. Aptaujas rezultāti rāda, ka visi 3 ietvari ierindojas populārāko tīmekļa ietvaru top 5 (skat. 2.6. att.). Tomēr izteikti visbiežāk tiek izmantots tieši React, apsteidzot citus tīmekļa ietvarus. Līdzīga situācija ir novērojama, salīdzinot ietvaru lejupielāžu skaitu no NPM repozitorija [14] (skat. 1. pielik.).



2.6. att. **2021. gada Stack Overflow iekšējās aptaujas rezultāti [13]**

Skatoties Stack Overflow aptaujas rezultātus pēdējo 3 gadu laikā, React un Vue popularitāte ir kāpusi, savukārt Angular popularitāte ir nedaudz kritusies (skat. 2.7. att.). Šeit gan jāpiemin, ka 2019. gada aptaujā pie Angular kategorijas tika pieskaitīts arī Angular.js, kas šajā darbā detalizēti netiek apskatīts, tāpēc Angular popularitātes kritums varētu būt mazāk izteikts.



2.7. att. **Pēdējo 3 gadu Stack Overflow iekšējo aptauju rezultāti**

2.4.2. Ekosistēma

Izstrādājot tīmekļa lietotnes, izvēlētais ietvars ne vienmēr piedāvās visu nepieciešamo funkcionalitāti. Tādēļ tālāk tiks 2.1. tabulā apkopotas un salīdzinātas biežāk lietotās funkcionalitātes un to pieejamība darbā apskatāmajos ietvaros.

2.1. tabula

Ar ietvariem biežāk saistīto bibliotēku apkopojums

	Angular	React	Vue
Stāvokļa pārvaldība	NgRx	Redux	Vuex
Komponenšu bibliotēka	Angular Material	Material UI, Ant Design	Vuetify, BootstrapVue
Maršrutēšana	Iebūvēts	React Router	Iebūvēts
Formu validācija, pārvaldība	Iebūvēts	Vairākas (Formik, React Hook Form)	VeeValidate
Starpplatformu atbalsts	PWA, Ionic, NativeScript	PWA, React Native	PWA, Vue Native

Sarežģītākos projektos var rasties nepieciešamība izmantot stāvokļa pārvaldības bibliotēku, lai lietotnē varētu centralizēti pārvaldīt tās stāvokli. Šajā ziņā visiem 3 ietvariem ir speciāli pielāgotas pakotnes – React ar Redux, Vue ar Vuex un Angular ar NgRx.

Viena no komponenšu balstīto ietvaru priekšrocībām ir komponenšu atkallietojamība, tādēļ šādiem ietvariem ir pieejamas vairākas komponenšu bibliotēkas ar biežāk lietotajām komponentēm, piemēram, pogas, ievadlauki, saraksti. Visiem 3 ietvariem šajā ziņā ir pieejama plaša izvēle. Balstoties uz GitHub vērtējumiem, Angular ietvaram vispopulārākā šāda tipa bibliotēka ir Angular Material, kurā komponentes ir balstītas uz Google Material dizaina vadlīnijām [15]. Uz to pašu vadlīniju pamata ir pieejamas arī bibliotēkas React un Vue ietvariem – Material UI un Vuetify.

Klienu bāzētajai maršrutēšanai ir nodrošināts atbalsts visos 3 ietvaros. Angular un Vue ietvarā ir iebūvēta maršrutēšana, savukārt React ietvarā tā ir pieejama kā atsevišķa komūnas uzturēta pakotne React Router.

Lai atvieglotu formu pārvaldību un validāciju, piemēram, e-pasta validēšanu, katram ietvaram ir pieejami vairāki varianti. Angular ietvarā ir iebūvēti 2 veidi kā pārvaldīt formas, kas ir sīki dokumentēti. Formu pārvaldībā ietilpst arī gatavas validāciju funkcijas, piemēram, vērtības garuma pārbaude, e-pasta pārbaude u.c [16]. React un Vue ietvariem formu pārvaldību var nodrošināt ar ārējām bibliotēkām, kuras piedāvā līdzīgu funkcionalitāti, kā Angular iebūvētā formu pārvaldība, piemēram, lauka stāvoklī ir iespējams apskatīt, vai lietotājs ir labojis lauku, kopš lietotne ir atvērta.

Ar visiem 3 ietvariem ir iespēja izstrādāt starpplatformu lietotnes gan tīmekļa, gan darbvirsmu, gan mobilo lietotņu veidā, tomēr var atšķirties metodes, kā tas tiek veikts. Lai nodrošinātu mobilo lietotņu atbalstu, viena metode ir izstrādāt progresīvo tīmekļa lietotni (PWA), kam tiek piedāvāts atbalsts visos 3 ietvaros. Lai izstrādātu mobilās lietotnes, kuras ir pieejamas Android un iOS veikalos, React un Vue ietvari piedāvā papildus ietvarus – React Native un Vue Native. Tos izmantojot, iespējams paralēli tīmekļa lietotnei izstrādāt atsevišķu dzimto mobilo lietotni, kura piedāvā labāku ātrdarbību un plašākas iespējas, kā PWA lietotne [17]. Lai ar Angular ietvaru izstrādātu mobilās lietotnes, ir iespēja izmantot kādu no hibrīdo mobilo lietotņu izstrādes ietvariem, kā piemēram, Ionic, vai kādu no dzimto mobilo lietotņu izstrādes ietvariem, kā piemēram, NativeScript. Tomēr šie ietvari nenodrošina tik labu veiktspēju kā React Native un Vue Native ietvari [20]. Hibrīdajām mobilajām lietotnēm salīdzinājumā ar dzimtajām ir ātrāks izstrādes laiks, tā kā ir iespējams izmantot tīmekļa lietotnes pirmkodu, bet dzimtajām lietotnēm ir raksturīga labāka ātrdarbība, kā arī iespējams izmantot operātājsistēmas specifiskās iespējas un UI elementus [18].

2.4.3. Mācīšanās līkne

Izvēloties jaunu ietvaru tīmekļa lietotnes izstrādei, ir jāizvērtē kāda ir tā mācīšanās līkne. Tas ietekmēs nepieciešamos resursus, lai atsevišķs izstrādātājs varētu apgūt šo ietvaru.

Angular tiek vērtēts kā ietvars ar augstāko mācīšanās līkni, tā kā tam ir vissarežģītākā projekta struktūra. Papildus komponenšu idejai, strādājot ar ietvaru jāapgūst servisi, kurus var injicēt komponentēs, kā arī moduļu veidošana. Papildus tam, izstrādātājam jāapgūst TypeScript programmēšanas valoda [19].

React ietvara apgūšana salīdzinājumā ar Angular ir vienkāršāka. Lai sāktu izstrādi ar React, izstrādātājam ir jāizprot komponenšu ideja, kā arī jāapgūst JSX, lai varētu atgriezt komponenšu veidnes.

Vue apgūšanas vieglums ir vērtējams līdzīgi kā React ietvaram, jo tam ir nepieciešams apgūt komponenšu ideju, kā arī jāapgūst veidņu sintakse, kas pēc principa ir līdzīga JSX veidņu sintaksei React gadījumā.

3. IETVARU PRAKTISKS SALĪDZINĀJUMS

3.1. Ierīces apraksts

Ietvaru testēšanas rezultāti var atšķirties no sistēmas uz kuras tiek veikti šie testi, kā arī no instalētās programmatūras. Tādēļ šajā sadaļā tiks aprakstīti sistēmas parametri, kā arī būtiskāko programmatūru versijas, kas var ietekmēt testēšanas rezultātus.

Lai salīdzinātu ietvaru ātrdarbību, tiek izmantots portatīvais dators ar sekojošiem sistēmas parametriem:

- Procesors: i7-9750H.
- Operatīvā atmiņa: 16GB 2667MHz.
- Datu glabātuve 1 (ar Windows): NVMe SSD 500GB.
- Datu glabātuve 2: SSD 1000GB.

Izmantotā programmatūra:

- Windows 11 21H2.
- Google Chrome v101.0.4951.67.

Testēšanā tiek izmantota populārākā un jaunākā pieejamā operētājsistēma. Tā tika izvēlēta, jo tā bija ierīcē sākotnēji uzstādīta un jo tai ir ļoti plašs programmnodrošinājuma atbalsts.

Kā pārlūkprogramma tika izvēlēts Google Chrome, kas ir populārākā no pārlūkprogrammām, sastādot 64% no tirgus daļas [26]. Google Chrome ir balstīta uz Chromium atvērtā pirmkoda projekta, kurš tiek izmantots arī citās populārās pārlūkprogrammās, piemēram, Windows iebūvētajā pārlūkprogrammā Microsoft Edge.

Lietotņu izstrādei tika izmantota populārākā atvērtā koda IDE Visual Studio Code 1.67.2. Atbilstoši 2021 gada Stack Overflow aptaujai 71% izstrādātāju regulāri to lieto [13]. IDE piedāvā plašu paplašinājumu klāstu, kas atvieglo izstrādi ar papildus koda iekrāsošanu, fragmentu ievietošanu (snippets) un citām funkcijām. Priekš lietotņu izstrādes tika izvēlēti un pievienoti paplašinājumi “Angular Language Service”, “ES7+ React/Redux/React-Native snippets” , “vue”.

3.2. Salīdzināšanas kritēriju izvēle

Vienu lapu lietotņu ietvaru salīdzināšanai tika izvēlēti vairāki kritēriji, lai pēc iespējas uzskatāmāk atspoguļotu ietvaru darbības atšķirības. Tika izvēlēti sekojoši kritēriji:

- Lietotnes izmērs.
- Ātrdarbība.

Atverot pārlūkā lietotni, tiek ielādēta visa vai daļa no lietotnes, kas aizņem daļu no lapas ielādes laika. Tāpēc lietotnes izmērs ietekmē lietotnes ielādes laiku. Minimāls lietotnes izmērs ir arī svarīgs situācijās, kad uz izmantotā servera ir ierobežota brīvā vieta.

Svarīgākais kritērijs ietvaru darbības salīdzināšanai ir ātrdarbība. No tās ir atkarīga, cik gluda un dinamiska būs lietotāja pieredze. Lai varētu salīdzināt ātrdarbību, tiks salīdzināts, cik ātri katrs no ietvariem spēj apstrādāt liela datu apjoma apstrādes operācijas. Versiju kontroles mitināšanas platformā GitHub ir pieejams projekts, kurā tiek veikti vairāki ātrdarbības testi daudziem no populārākajiem ietvariem [27]. Balstoties uz to, tabulā 3.1. ir izvēlētas testējamās operācijas.

3.1. tabula

Izstrādājamās lietotnēs testējamās operācijas

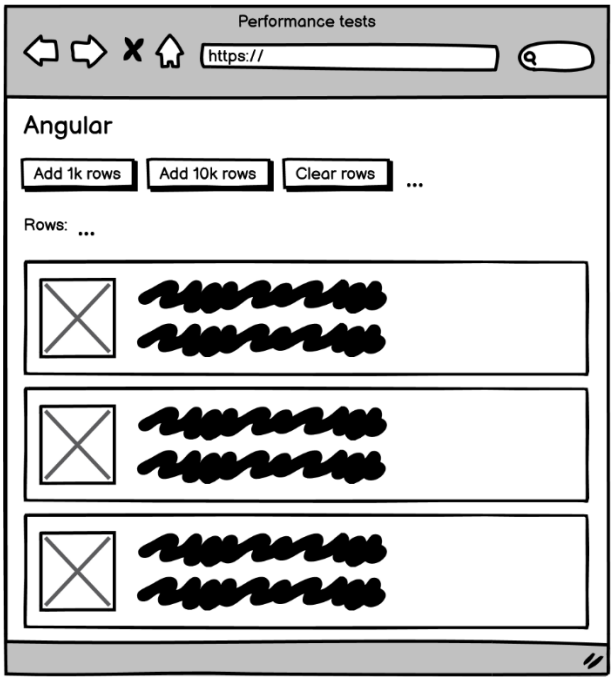
Identifikators	Operācija
ADD_10	Pievienot 10’000 rindas
DEL_10	Nodzēst 10’000 rindas
ADD_1	Pievienot 1’000 rindas
DEL_1	Nodzēst 1’000 rindas
EDIT	Labot 1’000 rindas
EDIT_10	Labot katru 10. rindu no 1’000 rindām
SWAP	Mainīt 2 rindas ar vietām
HIGHLIGHT	Iekrāsot rindu

Pirmās 2 operācijas ir paredzētas, lai testētu ietvarus uz lielāku datu apjomu. Pārējās operācijas testē ietvarus uz vidēju datu apjomu.

3.3. Izstrādājamās mājaslapas apraksts

Lai salīdzinātu ietvarus, ar katru no ietvariem tiks izstrādāta vienu lapu lietotne, kurā būs iespējams izpildīt iepriekš izvēlētās operācijas. To izpildes rezultātā lietotnē tiks ģenerēts un izmainīts saraksts ar iepriekš izvēlētiem datiem.

Lietotnes augšējā daļā būs pieejamas pogas operāciju izpildei. Lietotnes pārējo daļu aizņems saraksts ar datu ierakstiem. Virs saraksta tiks attēlots uzģenerēto rindu skaits (skat. 3.1. att.). Lai varētu izpildīt operāciju “Iekrāsot rindu”, katru no datu rindām būs iespējams iekrāsot ar peles klikšķi.



3.1. att. **Mājaslapas skice**

3.4. Izvēlētā datu kopa

Kā sarakstā attēlojamā datu kopa tika izvēlēts nejauši ģenerēts cilvēku saraksts, kurš tika iegūts no vietnes randomuser.me. Šai atvērtā pirmkoda vietnei ir pieejams API, caur kuru ir iespējams uzģenerēt un JSON formātā lejupielādēt datu kopu ar neīstiem cilvēku datiem, kā piemēram, vārds, uzvārds, atrašanās vieta, e-pasts un pat cilvēka attēls. Cilvēku attēlos ir redzami īsti cilvēki, bet viņi ir devuši piekrišanu brīvi izmantot viņu attēlus mājaslapās.

Vietne piedāvā lejupielādēt līdz pat 5’000 lietotāju. Priekš testējamām operācijām ir nepieciešams pievienot 10’000 ierakstu, tādēļ tajā gadījumā datu apjoms tiks dublēts.

Lai nodrošinātu testu stabilitāti, tā vietā, lai lietotnes veiktu pieprasījumus uz ģenerēšanas API, datu kopa tika vienu reizi manuāli lejupielādēta un pievienota pie lietotņu pirmkoda. Tādā veidā testu rezultāti nebūs tiešā veidā atkarīgi no lietotāju ģenerēšanas API.

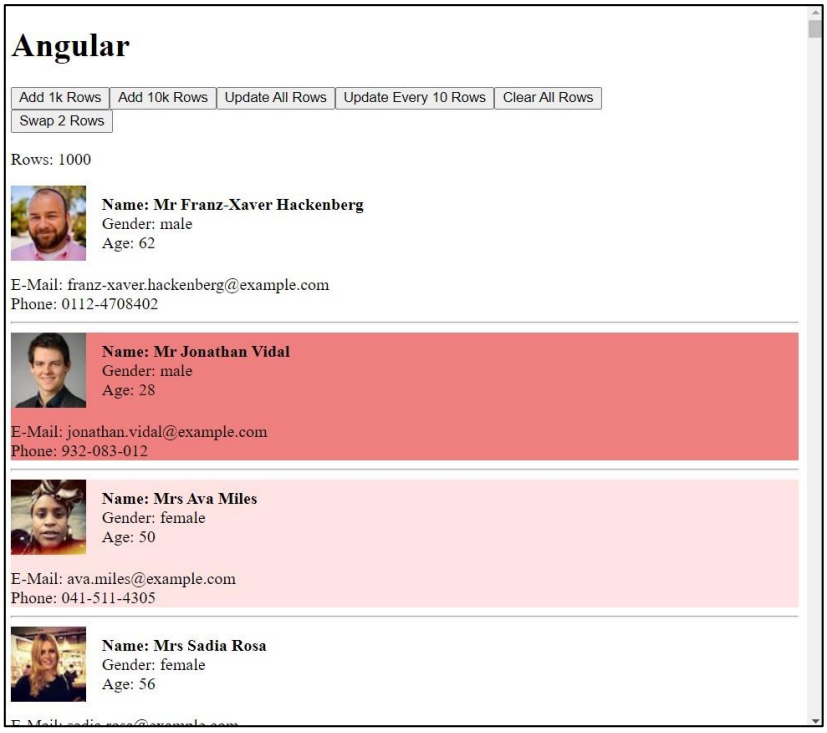
3.5. Izstrādes gaita

Pirmā tika izstrādāta Angular lietotne. Projekts tika ērti izveidots ar Angular CLI palīdzību. Ar tā palīdzību lietotnē tika vēl izveidotas 2 komponentes un 1 serviss. Bija sekojošas komponentes:

- Personas komponente - nodrošina 1 personas attēlojumu, saņemot personas datus.
- Personu saraksta komponente - konteineris operāciju pogām un attēlojamām personām.

Datu iegūšanai tika izveidots Angular serviss, kurš nolasa datus no faila un sanumurē tos, lai veidnēs būtu iespēja piešķirt ierakstiem atslēgas, uzlabojot ātrdarbību. Pēc servisa izveidošanas personu saraksta komponentē tas tika injicēts, lai varētu attēlot datus komponentes atvēršanas brīdī. Pogu loģika tika implementēta personu saraksta komponentē.

Lietotņu izveide React un Vue ietvaros bija līdzīga – tās sastāvēja no 2 komponentēm, bet bez injicējama servisa. Tā kā ietvariem veidnes bija balstītas uz HTML tās starp projektiem varēja kopēt, veicot nelielas modifikācijas. Operāciju implementāciju arī varēja atkārtoti izmantot starp projektiem. Izstrādes gaita šajos ietvaros bija līdzīga. Viena no atšķirībām bija, ka Angular ietvarā ir nodalīta veidne no komponentes atsevišķos failos. 3.2. attēlā ir redzama Angular ietvarā izstrādātā lietotne.



3.2. att. **Ar Angular ietvaru izstrādātas lietotnes skats**

3.6. Testa plāns

Lai lietotnēm veiktu operāciju testēšanu, tas tiks sadalīts 6 atsevišķos atkārtojamos testu gadījumos. Katrs no gadījumiem tiks 5 reizes izpildīts, lai iegūtu stabilākus testa rezultātus. Tabulā 3.2. ir sīkāk aprakstīti testu gadījumi, izmantojot 3.1. tabulā definētos identifikatorus.

3.2. tabula

Izpildāmie testu gadījumi

Veicamās operācijas	Operācijas, kuras tiks testētas
ADD_1 DEL_1	ADD_1 DEL_1
ADD_10 DEL_10	ADD_10 DEL_10
ADD_1 EDIT DEL_1	EDIT
ADD_1 EDIT_10 DEL_1	EDIT_10
SWAP	SWAP
HIGHLIGHT	HIGHLIGHT

Testa gadījumi tiks izpildīti manuāli. Lai varētu mērīt operāciju izpildes ilgumu, tiks izmantots Google Chrome pārlūkā iebūvētais Chrome DevTools Performance rīks. Ar tā palīdzību ir iespējams ierakstīt un analizēt ar lietotni veiktās darbības. Ierakstītajā laika intervālā iespējams analizēt CPU, GPU, atmiņas noslodzi, kadru skaitu sekundē un citus rādījumus. Lai analizētu CPU noslodzi, rīks ļauj kokveida struktūrās apskatīt JavaScript funkciju izsaukumus kā arī pārlūka renderēšanas darbības, kā piemēram, stila pārrēķinu, krāsošanu. Tādā veidā tīmekļa lietotnēm ir iespējams meklēt un analizēt ātrdarbības vājos punktus.

Šī darba ietvaros rīkā tiks ierakstīti testu gadījumi un katrai operācijai tiks piefiksēts tās izpildes ilgums, sākot ar JavaScript izsaukuma sākumu un beidzot ar pārlūka ekrāna pārkrāsošanu, kad ir nomainījies pārlūkā redzamais kadrs.

3.7. Testu rezultāti

Operāciju testēšana notika 2 iterācijās. 1. iterācijā atbilstoši testa plānam bija veikti testi un rezultāti tika apkopoti atsevišķās tabulās pa ietvariem. Pēc iterācijas autors novēroja vairākas nepilnības, tādēļ tika veikti labojumi un testi atkārtoti.

Testēto ietvaru versijas:

- Angular v13.3.0.
- React v18.1.0.
- Vue v3.2.33.

3.7.1. Pirmā iterācija

3.3. attēlā ir apkopoti 1. iterācijas rezultāti no ievāktajiem datiem (skat. 2., 3., 4. pielik.). Var novērot, ka ar visātrāko operāciju izpildi izceļas Angular un Vue ietvari. Jaunu ierakstu pievienošanas ilgums abiem ietvariem ir līdzīgs, neskatoties uz apjomu. Dzēšanas operācija šiem ietvariem atšķiras atkarībā no datu apjoma. 1'000 rindu dzēšanu Vue veic 2 reizes ilgāk nekā Angular, savukārt, dzēšot 10'000 rindas, ir pretēji – Vue to veic 3 reizes lēnāk nekā Angular. Līdzīgs nevienmērīgums novērojams arī šiem ietvariem veicot EDIT un EDIT_10 operācijas. Rindu mainīšanas operācija ir izteikti ātrāka Vue ietvaram, bet React ietvaram tā ir izteikti lēnāka. Iekrāsošanas operāciju arī ātrāk veic Vue ietvars, bet kopumā visi 3 ietvari to spēj ātri paveikt. Autors novēro, ka React kopumā vislēnāk izpildās - 5 no 8 operācijām React ietvars izpilda vislēnāk, no kurām vislēnākā izpildās 10 sekundes.

	Angular	Vue	React
ADD_10	6575.8	7115.6	10143.8
DEL_10	4478.4	1395.6	1828.6
ADD_1	744	829.2	1086.2
DEL_1	104.6	197.4	192.4
EDIT	142.2	446.6	262.4
EDIT_10	27.4	3.4	149.6
SWAP	69	17.2	193.2
HIGHLIGHT	18	5.2	18.2

3.3. att. **1. iterācijas testu rezultāti**

3.3. attēlā ir normalizēti 1. iterācijas rezultāti, lai varētu uzskatāmāk ieraudzīt ātrdarbības atšķirības. Visievērojamākās ātrdarbības atšķirības ir novērojamas EDIT_10 operācijai – React to izpilda 44 reizes lēnāk, kā Vue ietvars. Aiz tā seko SWAP operācija, kurai arī stipri atšķiras izpildes laiki starp ietvariem, kas varētu liecināt par to ka Vue ietvars šajā ziņā ir ar lielāku veiktspēju. Lai apkopotu operāciju ilgumus, katram no ietvariem tika aprēķināts ģeometriskais vidējais normalizētajiem operāciju laikiem. Rezultātā ir apskatāms, ka Vue ir ieguvis labāko ātrdarbības vērtību, aiz tā sekojot Angular un tad React ietvaram.

	Angular	Vue	React
ADD_10	1	1.0821	1.5426
DEL_10	3.2089	1	1.3103
ADD_1	1	1.1145	1.4599
DEL_1	1	1.8872	1.8394
EDIT	1	3.1406	1.8453
EDIT_10	8.0588	1	44
SWAP	4.0116	1	11.233
HIGHLIGHT	3.4615	1	3.5
Ģeometriskais vidējais	2.0864	1.2787	3.3872

3.4. att. **Normalizēti 1. iterācijas testu rezultāti**

Lielās ietvaru ātrdarbības atšķirības, piemēram, EDIT_10 operācijai, norādīja uz iespējamību, ka testēšanas pieeja varētu būt neprecīza. Autoram citā laika brīdī atkārtojot daļu no operācijām, rezultātos rādījās citas tendences, kā tas ir novērots 3.4. attēlā. Lai situāciju uzlabotu, tika veikti vairāki labojumi testēšanas pieejā un testi tika atkārtoti otrajā iterācijā.

3.7.2. Otrā iterācija

Lai padarītu testa rezultātus precīzākus tika veiktas sekojošas darbības:

- Rezultātos tika ignorētas pirmās testu vērtības. 2. pielikumā ADD_1 operācijai var novērot, ka pirmās vērtības ir netipiski lielas.
- Mājaslapas tika testētas no optimizētiem, kompilētiem failiem nevis izstrādes režīmā, jo praktiskā lietošanā tā tas tiek darbināts.
- Tika noņemti lietotāju attēli, jo tos mājaslapas lejupielādēja no ārējām saitēm, kas samazināja testu stabilitāti.

Pēc otrās testēšanas iterācijas visi izpildes laiki ir ievērojami samazinājušies (skat. 5., 6., 7. pielik.). 3.5. attēlā salīdzinājumā ar pirmās iterācijas rezultātiem ADD_10 React ietvaram izpildās 3 reizes ātrāk. Līdzīgi ir ar citiem laikiem. Pēc šiem rezultātiem Vue ietvars vēl stiprāk izvirzās priekšā izpildes laika ziņā. Vienīgās operācijas, kuras izpilda ātrāk Angular ietvars ir EDIT un EDIT_10. Atšķirībā no pirmās iterācijas ir arī stabilāki rezultāti atkarībā no datu apjoma. Piemēram, neatkarīgi no datu apjoma, Angular ietvars veic datu dzēšanu ilgāk nekā Vue un React.

	Angular	Vue	React
ADD_10	3328	2892	3512
DEL_10	463.6	204.4	286.4
ADD_1	420	375.2	474
DEL_1	42.4	21.4	40
EDIT	128.2	159.4	172.8
EDIT_10	23.4	24.8	41.6
SWAP	45.8	14	47.2
HIGHLIGHT	5	2.6	3.4

3.5. att. **2. iterācijas testu rezultāti**

Analizējot 3.6. attēlā redzamās normalizētās vērtības, redzams, ka salīdzinājumā ar pirmo iterāciju vairs nav tik stipri manāmu atšķirību starp operāciju izpildes laikiem. Ja pirmajā iterācijā atšķirības bija 44 un 11 reizes, tad šeit lielākā atšķirība ir 3.34 reizes, kas ir redzama SWAP operācijai. Var secināt, ka kopumā Vue šīs pamata operācijas veic visātrāk un Angular un Vue ietvari kopumā veic šīs pamata operācijas 1.5 reizes lēnāk kā Vue ietvars.

	Angular	Vue	React
ADD_10	1.1508	1	1.2144
DEL_10	2.2681	1	1.4012
ADD_1	1.1194	1	1.2633
DEL_1	1.9813	1	1.8692
EDIT	1	1.2434	1.3479
EDIT_10	1	1.0598	1.7778
SWAP	3.2714	1	3.3714
HIGHLIGHT	1.9231	1	1.3077
Ģeometriskais vidējais	1.5673	1.0351	1.5977

3.6. att. **Normalizēti 2. iterācijas testu rezultāti**

3.8. Salīdzinājums ar publiski pieejamu testu

GitHub vietnē publiski ir atrodams projekts, kurš ļauj veikt automatizētus testus, lai salīdzinātu daudzu ietvaru ātrdarbību [28]. Tajā tiek salīdzināta gan ātrdarbība pēc šajā darbā līdzīgām operācijām, gan vietņu palaišanas ātrums. Lai papildus veiktu ietvaru salīdzinājumu, darba autors lejupielādēja projektu uz tās pašas ierīces, kurā tika veikti manuālie testi. Pēc GitHub projektā aprakstīto soļu izpildes, tika veikti automatizētie testi, iegūstot sekojošus 3.7. attēlā redzamos rezultātus. Testu izpildes gaitā autors saskārās ar vairākiem tehniskiem šķēršļiem. Caur NPM rīku lejupielādējot projekta saistītās bibliotēkas, radās bibliotēku atkarību konflikts, kuru

izdevās atrisināt, atslēdzot līdzinieku atkarību *peer dependencies* lejupielādi. Projektā bija arī jāatrod un jānomaina norādītā Google Chrome atrašanās vieta, lai varētu veiksmīgi palaist testus.

Name Duration for...	vue- v3.2.26	angular- v13.0.0	react- v17.0.2
Implementation notes			
create rows creating 1,000 rows	110.0 ± 3.0 (1.00)	125.2 ± 3.8 (1.14)	132.7 ± 5.1 (1.21)
replace all rows updating all 1,000 rows (5 warmup runs).	101.7 ± 2.1 (1.00)	113.1 ± 4.7 (1.11)	118.7 ± 2.5 (1.17)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	414.6 ± 14.4 (1.10)	378.1 ± 6.3 (1.00)	460.1 ± 10.2 (1.22)
select row highlighting a selected row. (no warmup runs). 16x CPU slowdown.	64.5 ± 6.0 (1.00)	93.9 ± 4.3 (1.46)	230.2 ± 6.4 (3.57)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	66.2 ± 7.5 (1.00)	529.8 ± 10.8 (8.01)	532.1 ± 15.3 (8.04)
remove row removing one row. (5 warmup runs).	24.9 ± 0.7 (1.04)	23.9 ± 0.8 (1.00)	25.5 ± 0.6 (1.07)
create many rows creating 10,000 rows	1,154.6 ± 21.5 (1.00)	1,239.1 ± 19.0 (1.07)	1,498.7 ± 17.9 (1.30)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	274.8 ± 13.1 (1.00)	343.5 ± 6.3 (1.25)	351.3 ± 7.0 (1.28)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown.	120.9 ± 6.9 (1.00)	322.9 ± 15.6 (2.67)	127.2 ± 7.9 (1.05)
geometric mean of all factors in the table	1.02	1.55	1.65

3.7. att. **Automatizētu testu rezultāti**

Ir novērojams, ka šajos rezultātos ietvaru veiktspējas atšķirības ir stipri līdzīgas manuālajiem testiem. Kopumā, skatoties uz ģeometrisko vidējo, Vue ir izpildījies visātrāk. Angular vienā gadījumā izpildījies 1.56, otrajā 1.55 reizes ilgāk, savukārt React izpildījies 1.6 un otrā gadījumā 1.65 reizes ilgāk. No tā var secināt, ka Vue ietvars tik tiešām pamatdarbības veic visātrāk, aiz tā sekojot Angular un tad React ietvaram.

Automātisko testu rezultātos var novērot atšķirību starp manuālajiem – daļa no tiem izpildās stipri ātrāk. Piemēram, automātiskajos testos ietvari pievieno 1’000 rindas 3 reizes ātrāk nekā manuālajos testos. Tas mudināja autoru atsevišķi iedarbināt kādu no GitHub projektā iekļautajām

testējamām lietotnēm. Angular lietotnei ierakstot pievienošanas operācijas izpildes laiku ar Google DevTools, tas līdzinājās manuāli veiktajiem testiem. Daži no iespējamiem iemesliem šai parādībai.

- Izpildes laikā netika iekļauta daļa no darbībām, piemēram, pārlūka krāsošana.
- Automātiskie testi palaiž pārlūku īpašā darbības režīmā.

Tomēr šīs izpildes laiku atšķirības būtiski neietekmē ietvaru savstarpējo novērtējumu. Gan manuālajiem, gan automātiskajiem testiem tie ir līdzīgi.

3.9. Izmēru salīdzinājums

Lietotnes izmērs ir svarīga sastāvdaļa lietotnes ātrdarbībā. Jo lielāks lietotnes izmērs, jo ilgāk lietotājam būs jāgaida, kamēr pārlūks lejupielādēs lapu.

Tā kā izstrādāto lietotņu pirmkodā tika iekļauts liels apjoms datu, tie tika izkomentēti, lai varētu iegūt precīzākus izmērus. Pēc projektu kompilēšanas tika iegūti sekojoši izmēri:

- Angular - 162.91 KB.
- React - 148 KB.
- Vue 51.3 KB.

Vue projekts ir ar vismazāko izmēru, aiz tā sekojot React un Angular. Tā kā lietotnēs var atšķirties implementācijas detaļas, priekš salīdzināšanas ar katru ietvaru tika izveidoti jauni tukši projekti. To izmēri bija:

- Angular 111.23KB.
- React 144KB.
- Vue 50.65 KB.

Var novērot, ka React un Vue ietvaru izmēri nav īpaši mainījušies, savukārt Angular izmērs ir par trešdaļu sarucis. Tas varētu nozīmēt to, ka izmantojot Angular iebūvētās iespējas, diezgan strauji var pieaugt lietotnes izmērs. Kopumā šie lietotņu izmēri nav lieli, jo vidēji mājaslapas izmērs ir 2MB [29]. Angular projekta iestatījumos noklusēti ir uzstādīt rādīt ziņojumu, ja izmērs pārsniedz 500KB. Tas ir vēl viens apliecinājums tam, ka šie projektu izmēri nav lieli. Pie tam, serverim servējot lietotni, tipiski faili tiek kompresēti ar gzip, kas var samazināt failu izmēru pat par 90% [30].

REZULTĀTI

Bakalaura darba ietvaros tika veikta SPA pamatprincipu izpēte. Izpētes gaitā tika veikts šo lietotņu pamatdarbības izklāsts, kā arī tika radīts tās vizualizējums ar mērķi atspoguļot SPA darbības ideju. Lai varētu izvērtēt šīs pieejas pielietošanas iespēju izstrādes projektos, tika veikts SPA piedāvāto priekšrocību un trūkumu apkopojums.

Tika veikta populārāko ietvaru apkopojums, kas piedāvā iespēju izstrādāt iepriekš apskatītās SPA. Katram no tiem tika veikts detalizēts to piedāvāto iespēju apskats, kā arī tika veikts savstarpējs salīdzinājums. Salīdzinājumā tika analizēta ietvaru popularitāte, apkopota un savstarpēji salīdzināta ar ietvariem saistītā ekosistēma, kā arī tika salīdzināta šo ietvaru mācīšanās līkne.

Darba praktiskajā daļā tika veikta ietvaru salīdzinošā analīze no to praktiskās ātrdarbības skatupunkta. Lai to paveiktu, tika izmantotas iegūtās teorētiskās zināšanas, lai ar katru no ietvariem radītu SPA, kurā ir iespējams veikt vairākas lietotnes DOM manipulējošas operācijas. Lietotņu izstrādes gaitā autors guva praktiskas iemaņas darbā ar šiem ietvariem. Pēc to izstrādes tika izplānoti un veikti testi ar mērķi iegūt lietotnēs iekļauto operāciju izpildes ilgumu. Lai nodrošinātu rezultātu precizitāti, testi tika veikti 2 iterācijās, to starpā veicot nepieciešamos pielāgojumus. Iegūtie ātrdarbības rādītāji tika savā starpā salīdzināti. Papildus analīzei tika arī uzstādīts un lietots publiski pieejams automatizēts rīks, kas testē ietvaru ātrdarbību. No šī rīka iegūtie rezultāti tika salīdzināti ar autora izveidoto lietotņu testu rezultātiem, iegūstot kvalitatīvāku priekšstatu par operāciju izpildes ilgumiem ietvaros.

Tika veikts arī ar ietvara veidoto lietotņu aizņemtās vietas salīdzinājums, kas var manāmi ietekmēt lietotnes atvēršanas laiku. Salīdzinājumā tika iekļautas gan darbā izstrādātās lietotnes, gan no jauna izveidotas tukšas lietotnes, tādā veidā izvērtējot lietotnes sākotnējo izmēru.

SECINĀJUMI

Veicot SPA izpēti, autors secina, ka būtiskākā priekšrocība ir uzlabota lietotāja pieredze darbojoties ar tīmekļa vietni. Tas tiek panākts caur uzlabotu veiktspēju, gludāku vietnes darbību un labāku stabilitāti slikta tīkla apstākļos. SPA vietnēm ir arī vairāki vērā ņemami trūkumi, tomēr tie ir risināmi un tos ir iespējams mazināt, izmantojot SPA izstrādes ietvarus. Piemēram, visos apskatītajos ietvaros ir mazināti draudi XSS uzbrukumiem, kā arī sliktu vietnes SEO var risināt implementējot vietnē servera puses renderēšanu.

Salīdzinot SPA ietvarus, var novērot, ka izstrādātāju vidū visvairāk tiek lietots React ietvars. Var secināt, ka Angular ietvara popularitāte aug mazāk, kā tas bija iepriekšējos gados, kamēr jaunākam ietvaram Vue popularitāte pieaug straujāk. Ekosistēmas ziņā visiem apskatītajiem ietvariem ir pieejamas biežāk lietotās funkcionalitātes. Angular ietvaram liela daļa no funkcionalitātes ir iebūvēta, kamēr React un Vue ietvariem tā ir pieejama kā atsevišķas bibliotēkas, piemēram, maršrutēšana, formu pārvaldība. Vēl autors secina, ka ar visiem apskatītajiem ietvariem var izstrādāt mobilās lietotnes, tomēr lielāks atbalsts ir React un Vue ietvariem, jo tiem ir pieejama atsevišķa bibliotēka, lai izstrādātu ātras dzimtās mobilās lietotnes.

Veicot ietvaru ātrdarbības savstarpējo salīdzinājumu, autors secina, ka Vue ietvars ir ar lielāku veiktspēju priekš vietņu pamata operāciju veikšanas. Ietvaram ir arī raksturīgs mazāks kompilēto failu izmērs, kas nodrošina ātru lietotnes ielādi pārlūkā.

Darba tālākie pētījuma virzieni varētu būt mazāk zināmu, bet ar augošu popularitāti raksturīgu ietvaru izpēte, piemēram, Svelte, kā arī augstāka līmeņa ietvaru izpēte, kas var atvieglot un uzlabot darbu ar apskatītajiem SPA ietvariem, piemēram, Next, Nuxt.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. *Understanding single page apps & client-side routing*. [tiešsaiste]. [skatīts 23.04.2022.]. Pieejams Internetā: https://bholmes.dev/blog/spas-clientside-routing/
2. *React Router and Client-Side Routing*. [tiešsaiste]. [skatīts 23.04.2022.]. Pieejams Internetā: https://betterprogramming.pub/react-router-and-client-side-routing-2e483452fbfb
3. *Between the Wires: An interview with Vue.js creator Evan You*. [tiešsaiste]. [skatīts 23.04.2022.]. Pieejams Internetā: https://www.freecodecamp.org/news/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4/
4. *Vue API Styles*. [tiešsaiste]. [skatīts 24.04.2022.]. Pieejams Internetā: https://vuejs.org/guide/introduction.html#api-styles
5. *Vue Single-File Components*. [tiešsaiste]. [skatīts 24.04.2022.]. Pieejams Internetā: https://vuejs.org/guide/scaling-up/sfc.html
6. *Nuxt.js Fundamentals*. [tiešsaiste]. [skatīts 25.04.2022.]. Pieejams Internetā: https://vueschool.io/lessons/what-is-nuxtjs
7. *Introduction to the Angular Docs*. [tiešsaiste]. [skatīts 03.05.2022.]. Pieejams Internetā: https://angular.io/docs
8. *What is The Major Difference Between Angular and AngularJS?* [tiešsaiste]. [skatīts 03.05.2022.]. Pieejams Internetā: https://www.monocubed.com/blog/difference-between-angular-and-angularjs/
9. *The History of React.js on a Timeline*. [tiešsaiste]. [skatīts 09.05.2022.]. Pieejams Internetā: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/
10. *Introducing Hooks*. [tiešsaiste]. [skatīts 09.05.2022.]. Pieejams Internetā: https://reactjs.org/docs/hooks-intro.html
11. *Introducing JSX*. [tiešsaiste]. [skatīts 10.05.2022.]. Pieejams Internetā: https://reactjs.org/docs/introducing-jsx.html
12. *Hooks at a Glance.* [tiešsaiste]. [skatīts 10.05.2022]. Pieejams Internetā: https://reactjs.org/docs/hooks-overview.html
13. 2021. gada Stack Overflow aptaujas rezultāti. [tiešsaiste]. [skatīts 11.05.2022.]. Pieejams Internetā: https://insights.stackoverflow.com/survey/2021#technology
14. *NPM trends. @angular/core vs react vs vue*. [tiešsaiste]. [skatīts 11.05.2022.]. Pieejams Internetā: https://www.npmtrends.com/@angular/core-vs-react-vs-vue

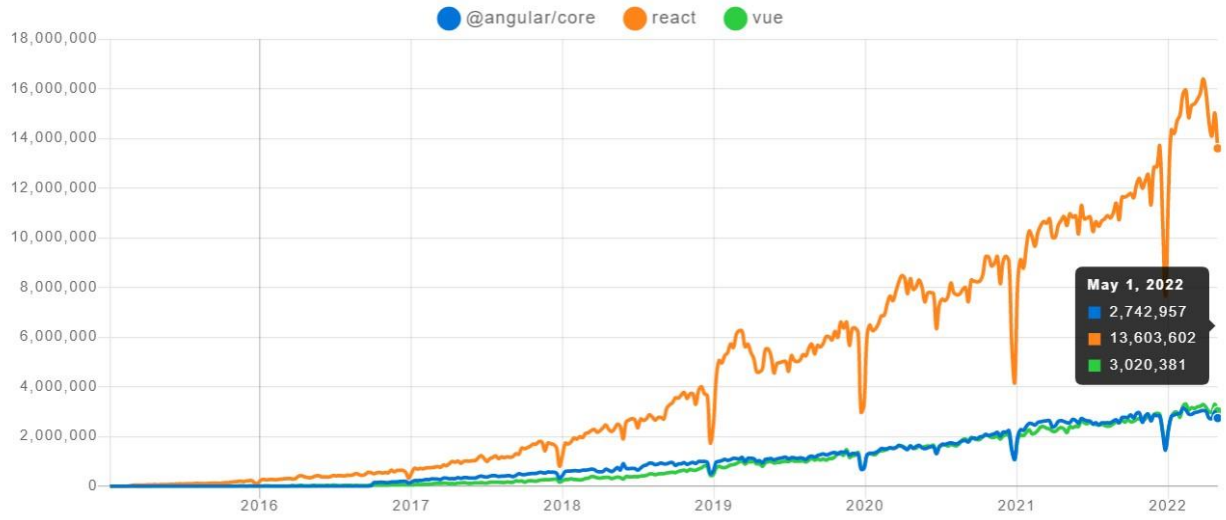
15. *Top Angular Component Libraries to Try in 2022*. [tiešsaiste]. [skatīts 13.05.2022.]. Pieejams Internetā: https://aglowiditsolutions.com/blog/top-angular-component-libraries
16. *Angular Validators*. [tiešsaiste]. [skatīts 13.05.2022.]. Pieejams Internetā: https://angular.io/api/forms/Validators
17. *PWA vs Native App and how to choose between them*. [tiešsaiste]. [skatīts 15.05.2022.]. Pieejams Internetā: https://www.magestore.com/blog/pwa-vs-native-app-and-how-to-choose-between-them/
18. *Difference Between Web vs Hybrid vs Native Apps*. [tiešsaiste]. [skatīts 15.05.2022.]. Pieejams Internetā: https://www.lambdatest.com/blog/web-vs-hybrid-vs-native-apps
19. *Angular vs React vs Vue 2022*. [tiešsaiste]. [skatīts 16.05.2022.]. Pieejams Internetā: https://athemes.com/guides/angular-vs-react-vs-vue/
20. *NativeScript vs. React Native*. [tiešsaiste]. [skatīts 17.05.2022.]. Pieejams Internetā: https://blog.logrocket.com/nativescript-react-native/
21. Michael S. Mikowski, Josh C. Powell. *Single Page Web Applications: JavaScript end-to-end 1st Edition.* Manning. 2013.
22. *What are Single Page Applications and Why Do People Like Them so Much?* [tiešsaiste]. [skatīts 17.05.2022.]. Pieejams Internetā: https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application
23. *What are Single Page Applications and Why Do People Like Them so Much?* [tiešsaiste]. [skatīts 19.05.2022.]. Pieejams Internetā: https://www.apriorit.com/white-papers/437-single-page-apps
24. *What is Single Page Application (SPA)? Pros and Cons with Examples*. [tiešsaiste]. [skatīts 20.05.2022.]. Pieejams Internetā: https://www.netsolutions.com/insights/single-page-application/
25. *The pros and cons of single page applications (SPAs)*. [tiešsaiste]. [skatīts 20.05.2022.]. Pieejams Internetā: https://www.itechart.com/blog/pros-cons-of-single-page-applications/
26. *Browser Market Share Worldwide*. [tiešsaiste]. [skatīts 24.05.2022.]. Pieejams Internetā: https://gs.statcounter.com/browser-market-share
27. *Results for js web frameworks benchmark - official run*. [tiešsaiste]. [skatīts 24.05.2022.]. Pieejams Internetā: https://krausest.github.io/js-framework-benchmark/2021/table_chrome_93.0.4577.63.html

28. *GitHub. js-framework-benchmark.* [tiešsaiste]. [skatīts 28.05.2022.]. Pieejams Internetā: https://github.com/krausest/js-framework-benchmark
29. *Http archive. Report: Page Weight*. [tiešsaiste]. [skatīts 28.05.2022.]. Pieejams Internetā: https://httparchive.org/reports/page-weight?start=2020_01_01&end=latest&view=list
30. *What is Gzip?* [tiešsaiste]. [skatīts 28.05.2022.]. Pieejams Internetā: https://blog.stackpath.com/glossary-gzip/#:~:text=Used%20mostly%20on%20code%20and,files%20by%20up%20to%2090%25
31. *Vue. Server-Side Rendering (SSR).* [tiešsaiste]. [skatīts 28.05.2022.]. Pieejams Internetā: https://vuejs.org/guide/scaling-up/ssr.html
32. *What You Should Know about Code-Splitting with Nuxt.js*. [tiešsaiste]. [skatīts 28.05.2022.]. Pieejams Internetā: https://www.telerik.com/blogs/what-you-should-know-code-splitting-nuxtjs
33. *Angular Architecture*. [tiešsaiste]. [skatīts 29.05.2022.]. Pieejams Internetā: https://www.ngdevelop.tech/angular/architecture/
34. *Reactive Core architecture for React Native and React applications*. [tiešsaiste]. [skatīts 29.05.2022.]. Pieejams Internetā: https://www.pinterest.com/pin/614952524091687742/
35. *Vue. Getting Started*. [tiešsaiste]. [skatīts 29.05.2022.]. Pieejams Internetā: https://012.vuejs.org/guide/

PIELIKUMI

1. pielikums

SPA ietvaru lejupielāžu skaits no NPM repozitorija [14]



2. pielikums

Angular ātrdarbības laiki milisekundēs pēc pirmās testu iterācijas

ADD_1	DEL_1	SWAP	ADD_10	DEL_10	EDIT	EDIT_10	HIGHLIGHT
1038	149	83	6380	4529	191	27	54
733	104	59	6922	4482	124	30	8
675	95	58	6518	4472	130	27	6
677	86	78	6601	4449	123	27	14
597	89	67	6458	4460	143	26	8

3. pielikums

React ātrdarbības laiki milisekundēs pēc pirmās testu iterācijas

ADD_1	DEL_1	SWAP	ADD_10	DEL_10	EDIT	EDIT_10	HIGHLIGHT
1373	178	239	10276	1786	284	151	70
996	193	182	10136	1763	251	148	5
1005	198	179	10220	1773	263	151	5
1006	208	177	10103	1754	251	144	6
1051	185	189	9984	2067	263	154	5

4. pielikums

Vue ātrdarbības laiki milisekundēs pēc pirmās testu iterācijas

ADD_1	DEL_1	SWAP	ADD_10	DEL_10	EDIT	EDIT_10	HIGHLIGHT
1138	225	19	8104	1383	466	3	12
748	192	15	6743	1452	466	4	4
757	189	18	7199	1367	441	3	4
748	196	17	6963	1368	448	3	3
755	185	17	6569	1408	412	4	3

5. pielikums

Angular ātrdarbības laiki milisekundēs pēc otrās testu iterācijas

ADD_1	DEL_1	SWAP	ADD_10	DEL_10	EDIT	EDIT_10	HIGHLIGHT
411	35	56	3540	487	120	24	5
410	41	52	3270	457	135	24	7
421	43	50	3300	451	127	22	5
431	45	36	3240	473	131	23	4
427	48	35	3290	450	128	24	4

6. pielikums

React ātrdarbības laiki milisekundēs pēc otrās testu iterācijas

ADD_1	DEL_1	SWAP	ADD_10	DEL_10	EDIT	EDIT_10	HIGHLIGHT
451	36	52	3490	310	168	42	4
521	36	48	3540	279	170	44	4
451	33	46	3410	284	178	38	3
491	49	45	3590	276	182	42	3
456	46	45	3530	283	166	42	3

7. pielikums

Vue ātrdarbības laiki milisekundēs pēc otrās testu iterācijas

ADD_1	DEL_1	SWAP	ADD_10	DEL_10	EDIT	EDIT_10	HIGHLIGHT
396	20	13	3290	188	154	24	3
357	22	14	2830	170	156	26	3
370	22	11	2970	232	172	29	2
346	21	11	2670	205	183	23	2
407	22	21	2700	227	132	22	3