Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK : 004.3
DOI: https://doi.org/10.24867/10BE41Degenek

ПОРЕЂЕЊЕ ANGULAR, REACT.JS И VUE.JS JAVASCRIPT ПРОГРАМСКИХ ОКВИРА НА ПРИМЕРУ УПРАВЉАЊА СТАЊИМА КЛИЈЕНТСКИХ АПЛИКАЦИЈА

COMPARISON OF ANGULAR, REACT.JS AND VUE.JS JAVASCRIPT FRAMEWORKS ON THE EXAMPLE OF CLIENT APPLICATION STATE MANAGEMENT

Арсеније Дегенек, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У овом раду дат је опис, као и поређење шаблона за ефикасно управљање стањима клијентских апликација у три тренутно најпопуларнија JavaScript програмска оквира. На основу детаљне анализе и примене библиотека специфичних за сваки програмски оквир, изведени су закључци о томе колико је сваки од њих погодан за коришћење. Сврха читаве анализе јесте доћи до одговора којом стратегијом и уз помоћ ког програмског оквира је најлакше структурирати клијентску апликацију, тако да она остане погодна за одржавање упркос својој комплексности. У циљу ефикасног поређења, употребом сваког програмског оквира креирана је по једна прототипска апликација. Све оне су базиране на истом скупу функционалности, осмишљених тако да дају реалну слику о предностима и манама примењеног шаблона.*

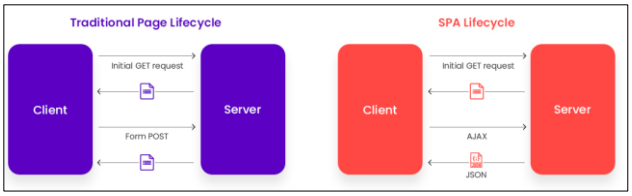
Кључне речи: *Програмски оквир, библиотека, управљање стањима*

Abstract – *This paper provides a description, as well as a comparison of patterns for efficient state management of client applications in three currently most popular JavaScript frameworks. Based on a detailed analysis and implementation of framework-specific libraries, conclusions were drawn about how suitable each of them is for use. The main goal of the whole analysis is to decide which strategy and framework are the most convenient to structure the client application, so that it remains easy for maintenance, despite its complexity. Using each framework, one prototype application was created for the purpose of more efficient comparison. All of them are based on the same set of functionalities, designed to give the real picture of the advantages and disadvantages of the applied pattern.*

Keywords: *Framework, library, state management*

1. УВОД

Као последица велике популарности веб апликација последњих година, захтеви за модерним и интерактивним корисничким интерфејсом су порасли.
То је довело до замирања традиционалног приступа развоја коришћењем вишестраничних апликација (енг. ***Multi-Page Applications***). Уместо њих, појавиле су се једностраничне апликације (енг. ***Single Page Applications***). Оне су омогућиле раздвајање клијентске и серверске стране, због чега је серверски део постао истовремено употребљив и за апликације које се покрећу на платформама мобилних уређаја. Такође, једностраничне апликације симулирају природно понашање претраживача, што значи да страница неће бити поновно учитана приликом добављања новог садржаја са сервера. Разлика између ова два приступа дата је на слици 1.



Слика 1. Вишестраничне и једностраничне апликације

Ослањајући се на овај концепт, дошло је до појаве различитих програмских оквира базираних на JavaScript језику. Они представљају основу са предефинисаним функционалностима, коју је могуће искористити за бржи и квалитетнији развој једностраничних апликација. Сви овде анализирани програмски оквири су засновани на компонентама, као основним градивним јединицама које заједно формирају изглед и понашање странице приказане у претраживачу.

1.1. Мотивација

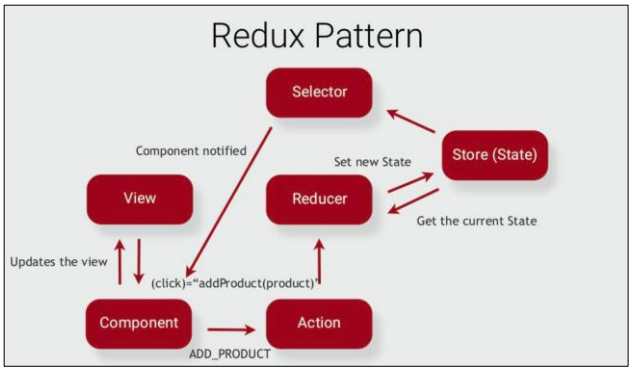
Употреба програмских оквира знатно је олакшала развој комплексних веб апликација. Ствари као што су рутирање, интеракција са DOM-ом (енг. ***Document Object Model***), комуникација са сервером, употреба форми и слично знатно су упрошћене и добро документоване. Међутим, проблем се јавља у ситуацијама које није било лако предвидети приликом пројектовања програмског оквира. У сва три наведена програмска оквира, компоненте међусобно комуницирају на два могућа начина:

- Child-To-Parent комуникација.
- Cross-Component комуникација.

У првом случају, две компоненте не могу да комуницирају директно, без да њихова комуникација иде преко посредника, који је у овом случају родитељска компонента. Ово за последицу има веома

НАПОМЕНА:
Овај рад је проистекао из мастер рада чији ментор је био др Драган Дину, доцент.

велико уплитање зависности, које постаје готово немогуће контролисати, јер је зарад комуникације између две компоненте потребно увести трећу. У другом случају, компоненте комуницирају преко дељене магистрале. Комуникација је базирана на емитовању и прихватању догађаја (енг. ***Events***). Ипак, магистрала временом постаје оптерећена, непрегледна и неконзистентна са становишта података, јер компоненте могу несметано да јој приступају.
Решење за овај проблем лежи у употреби Redux шаблона. Настао с циљем олакшавања комуникације између компоненти, овај шаблон уводи нове појмове који су објашњени у наставку. Идеја је да на нивоу апликације у сваком тренутку постоји само један извор истине у формату JSON објекта, односно једно стање (енг. ***State***). Ново стање се креира од нуле при свакој промени било којег његовог дела и оно је садржано у складишту (енг. ***Store***). Такође, једини начин да се оно промени јесте отпремањем акције (енг. ***Action***). Акције су функције које имају интеракцију са редукторима (енг. ***Reducer***) и омогућавају извршавање асинхроног кода. Редуктори су једине методе које директно могу да мењају стање. На крају, круг се затвара тако што компоненте бивају информисане о промени стања преко селектора (енг. ***Selector***). На слици 2 приказана је структура Redux шаблона.



Слика 2. Ток података у Redux шаблону

Дакле, у наставку ће бити анализиране и поређене имплементације овог шаблона у Angular, React.js и Vue.js програмским оквирима.

2. ПРЕГЛЕД СТАЊА У ОБЛАСТИ

Сходно популарности коју JavaScript програмски оквири имају последњих година, постоји велики број истраживања како појединачног програмског оквира, тако и њихових међусобних односа. Међутим, ниједно истраживање везано за њихово поређење не даје акценат на област управљања стањима.
У оквиру рада [1] дато је детаљно поређење ова три програмска оквира по популарности, тежини савладавања и перформансама, али без осврта на управљање стањима приликом дефинисања наведених критеријума.
Рад [2] се такође бави поређењем поменутих програмских оквира, али као предмет поређења узима уграђени сет функционалности, подршку и величину заједнице која стоји иза сваког од њих.
Треће истраживање [3] даје врло детаљан осврт на сва три програмска оквира, али улази у детаље интеракције са DOM-ом, а то није од велике користи за потребе овог истраживања.
Што се тиче самог Redux шаблона, стручна литература постоји и даје веома прецизно објашњење како он функционише и у којим случајевима га треба користити [4]. Оно што свакако фали, а често се доводи у питање јесте који програмски оквир изабрати, уколико је донета одлука да се за управљање стањем клијентске апликације користи Redux шаблон.

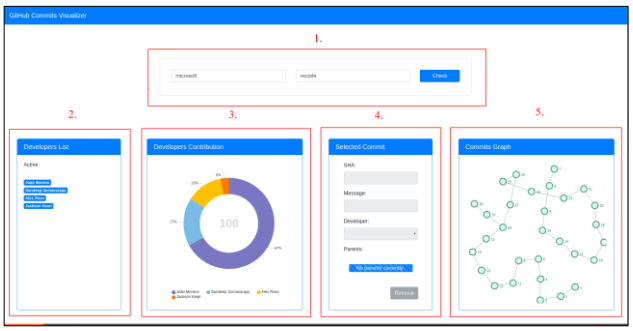
3. МЕТОДОЛОГИЈА

С обзиром да је предмет истраживања поређење Redux-а и његова анализа у различитим срединама, најтачнији и најверодостојнији закључци изведени су из директне имплементације самог шаблона у сва три поменута програмска оквира. За развој React.js и Vue.js апликација коришћен је Javascript језик, односно његова ES6 спецификација. У случају Angular-а, изворни код писан је у TypeScript-у.

3.1. Софтверска имплементација

Приликом спецификирања захтева прототипског софтвера, акценат је био на креирању апликације која ће у позадини моћи да имплементира све концепте које обухвата Redux, а да има примену и да са визуелног аспекта буде погодна за коришћење. Из тог разлога, апликација садржи дијаграме и графове, чија стања се чувају на централизованом месту (Store) и сведоче о исправној функцији примењеног шаблона. Апликација је функционално једнака за све програмске оквире, али постоје минималне разлике у корисничком интерфејсу јер сваки од њих подржава различите пакете за визуелизацију података. Скуп подржаних функционалности (слика 3):

- Преузимање последњих тридесет промена (енг. ***Commit***) са подразумеване гране жељеног GitHub репозиторијума.
- Приказ свих програмера који су починили промене и филтрирање промена по програмеру који их је направио.
- Приказ ангажовања сваког програмера на дијаграму (Pie Chart).
- Приказ почињених промена у структури графа.
- Модификација одређене промене и њено брисање.



Слика 3. Апликација и компоненте

Библиотеке које имплементирају Redux у сваком од програмских оквира:

- ngrx (Angular),
- redux (React.js),
- vuex (Vue.js).

3.2. Интеграција са Redux библиотекама

С обзиром на то да Redux не долази уграђен ни у један програмски оквир, један од критеријума поређења био је његова интеграција са већ постојећом апликацијом која користи неки од традиционалних приступа за управљање стањима. У сва три случаја, језгро шаблона интегрише се као пакет преко npm-a (енг. ***Node Package Manager***). Оно што се разликује јесте постављање Store-a тако да му буде могуће програмски приступити. Такође, додатна својства шаблона као што су нпр. комуникација са сервером или ослушкивање промена у неким програмским оквирима су подржана као део основног пакета, док се у осталим морају накнадно инсталирати.

3.3. Нормализација података

Препорука је да се Redux шаблон користи искључиво у комплексним клијентским апликацијама које изискују велику међусобну зависност између компоненти и раде над великом количином података. Такве апликације у 99% случајева комуницирају са сервером и од њега добијају податке. С обзиром да серверске апликације већином раде над релационим подацима, те релације пренеће се и на клијентску страну, сваки пут када подаци стигну. Сходно томе, иако није имплементирана приликом инсталације основног пакета Redux-a, нормализација података се сматра неизоставним делом у његовој употреби. Њена намена је да нам обезбеди такву структуру Store-a да нам његово коришћење на дуге стазе не донесе више штете него користи.
Под овим се подразумева реорганизација JSON објекта тако да је олакшана претрага и повлачење података и спречавање њиховог дуплирања (сва референцирања иду преко јединственог идентификатора). У случају свих програмских оквира коришћена је ***normalizr*** библиотека. Она захтева креирање шема на основу којих се структура Store-a постави приликом првобитног уписа података. Библиотека је базирана на ES6 спецификацији, али постоји њен подскуп који се користи за TypeScript.

На основу наведеног се закључује да је ова библиотека од изузетне важности, али се различито укапа у зависности од језика којим је имплементирана логика апликације. Због тога, интеграција са њом и једноставност коришћења API-ја које она нуди, коришћени су као други критеријум за поређење ових програмских оквира.

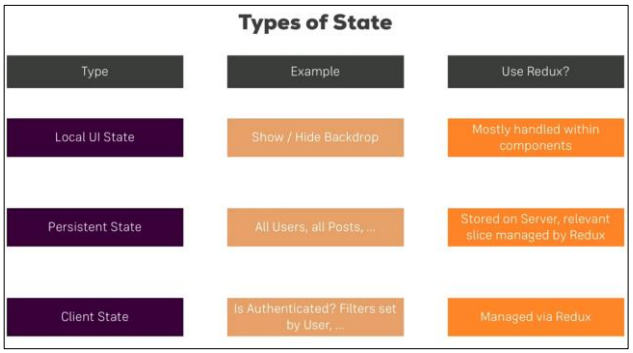
3.4. Модуларизација Store-a

Још један битан критеријум представља могућност поделе Store-a на модуле. Ово је важно из два разлога. Први је што Store постане изузетно непрегледан и тежак за одржавање при великој количини података, упркос томе што су они нормализовани. У том случају, има смисла поделити га на мање делове тј. модуле, тако да сваки од њих садржи податке који припадају истој логичкој целини. Са становишта шаблона, овде се ништа не мења. Сви модули на крају ће бити спојени у један велики Store, као и до сада. Ово је само структурално побољшање, које може бити од велике користи.
Други разлог заснива се на чињеници да готово никада неће бити потребно нормализовати све податке који се налазе у Store-у. Ствари као што су флегови, подаци о ауторизацији и слично неће бити нормализовани, те нема потребе мешати их са остатком података. Због тога нам подршка модуларизације омогућава да поред логичких целина, апликационо стање раздвојимо и на основу тога који његови делови су нормализовани, а који не.

3.5. Остали критеријуми поређења

Поред издвојених критеријума, коришћено је још мањих, који су од значаја приликом избора програмског оквира за подржавање Redux шаблона. Читљивост и прегледност изворног кода након интеграције са Redux-ом свакако је први следећи критеријум вредан разматрања. Треба нагласити да оно што се овде пореди није читљивост кода пре и после употребе шаблона. Сама чињеница да је шаблон настао, већ нам говори да је изворни код знатно побољшан. Овде се мисли на међусобно поређење читљивости код ова три програмска оквира, након што сваки од њих имплементира шаблон, над истим сетом функционалности.
Друга битна ствар је процес дебаговања Store-a. Ово је од изузетне важности за брзо и ефикасно отклањање проблема. У сва три случаја, алати за дебаговање се морају додатно инсталирати у претраживач и увезати са апликацијом. Међутим, они нису исти за све програмске оквире, што даје додатни простор за поређење.
На крају овог поглавља, треба додати да поред увођења Redux шаблона у апликацију, постоје њени делови који неће бити садржани у оквиру глобалног стања. Овде пре свега спадају флегови који су везани искључиво за изглед саме компоненте и нису од значају другим компонентама. Њихово измештање у Store не доноси никакве бенефите, него напротив уводи компликације у цео процес, те се сматра лошом праксом. На слици 4 приказано је који типови

података иду у глобално стање, а који остају везани за компоненту.



Слика 4. Употреба Redux-а у зависности од типа података

4. ЗАКЉУЧАК

Након имплементације шаблона који омогућују лакше и поузданије управљање стањима клијентских апликација и његовог поређења кроз три JavaScript програмска оквира, изведени су закључци везани за одабир најпогоднијег од њих. На почетку, одмах, треба нагласити да не постоји методологија која ће по свим критеријумима прогласити један програмски оквир као најбољи избор. У том случају, не би постојала остала два и не би имала толико широку употребу.

Са становишта интеграције са Redux библиотеком, највише посла је било при изради апликације у React.js програмском оквиру. Разлог за ово је што Redux библиотека није уско везана за програмски оквир као друге две, те је поред њеног укључивања било потребно укључити и библиотеку за њено конектовање са програмским оквиром. Такође, за рад са селекторима потребно је укључити додатну библиотеку, али то је последица оптимизације која не постоји у остала два програмска оквира.

Када је реч о нормализацији података, за очекивати је било да ће тај процес бити много лакши у React.js и Vue.js програмским оквирима, јер је библиотека базирана на ES6 спецификацији. Упркос томе што постоји њена подршка за TypeScript, она у тренутку израде Angular апликације није била довољно развијена, те није подржавала креирање сложенијих акција над нормализованим Store-ом. Из овог разлога, библиотека је враћена на верзију коју користе друга два програмска оквира, али као последица тога шеме и функције за нормализовање нису креиране у духу TypeScripta.

Процес поделе Store-а на модуле могућ је у сва три програмска оквира. Ипак, најједноставније га је било имплементирати у Vue.js-у, јер у њему не постоји концепт редуктора, који је као последица модуларизације трпео промене у остала два програмска оквира.

Програмерски алати за дебаговање су исти у случају Angular i React.js програмских оквира. Прегледни су и једноставни за употребу. Vue.js има свој, засебан алат, али упркос томе што није компатибилан са осталим програмским оквирима, нуди готово све исте функционалности.

На крају, треба навести главне адуте сваког од наведених програмских оквира. Angular је дефинитивно прави избор за изградњу стабилне апликације отпорне на велику количину багова, захваљујући статичкој типизацији коју омогућава TypeScript. Ово се наравно одражава и на Store, јер се он такође писан у овом језику.

Са друге стране, redux библиотека коју користи React.js је дефинитивно најразвијенија од све три, првенствено због чињенице да је овај шаблон првобитно настао за потребе овог програмског оквира. То за последицу има дужи процес савладавања библиотеке, али и њено много ефикасније коришћење јер су функционалности поприлично оптимизоване у позадини.

Vue.js је настао као програмски оквир који је покупио најбоље од претходна два и покушао то да поједностави. То је приметно и у случају управљања стањима. Његовом употребом сам процес развоја ће ићи брзо. Коришћење vuex библиотеке је упрошћено и што је најважније, притом нису изгубљене функционалности.

ЛИТЕРАТУРА

[1] Elar Saks *"JavScript frameworks: Angular vs React vs Vue"*. [Преузето: 20.09.2020]

[2] Eric Wohlgethan *"Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React adnd Vue.js"*. [Преузето: 20.09.2020]

[3] Mattias Levlin *"DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue and Svelte"*. [Преузето: 20.09.2020]

[4] Nir Kaufman *"Thinking in Redux"*. [Преузето: 20.09.2020]

Кратка биографија:

Арсеније Дегенек рођен је у Бачкој Тополи 1995. године. Мастер рад на факултету техничких наука из области Електротехнике и рачунарства – Рачунарство и аутоматика одбранио је 2020. год.