Зборник радова Факултета техничких наука, Нови Сад

УДК: 004.438
ДОИ: https://doi.org/10.24867/27BE19Bogdanovic

ПРИМЕР КОМПЛЕТНЕ ВЕБ АПЛИКАЦИЈЕ У ПРОГРАМСКОМ ЈЕЗИКУ РАСТ

EXAMPLE OF A FULLSTACK WEB APPLICATION IN THE RUST PROGRAMMING LANGUAGE

Огњен Богдановић, *Факултет техничких наука, Нови Сад*

Област – ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО

Кратак садржај – *Овај рад се бави развојем комплетне веб апликације помоћу програмског језика Раст, где је и серверски и клијентски део апликације имплементиран у одговарајућим радним оквирима овог језика. Анализом тренутне слике развоја веб апликација, примећено је да решења, иако махом прате савремене технологије, не придају превише значаја безбедности меморије и самим перформансама. Са друге стране, Раст поседује потенцијал да направи искорак у веб развоју пружањем сигурније алтернативе постојећим технологијама. Због тога, решење имплементирано у овом раду представља апликацију управо користи предности овог језика, и обезбеђује сигурнији, ефикаснији и одрживи развој веба.*

Кључне речи: *веб апликација, Раст, Веб Асембли*

Abstract – *This paper deals with the development of a fullstack web application using the Rust programming language, where both the server and client part of the application are implemented in the appropriate frameworks of this language. Analyzing the current state of web application development, it has been noticed that the solutions, although they mostly follow modern technologies, do not give too much attention to memory security and performance. On the other hand, Rust has the potential to make a breakthrough in web development by providing a safer alternative to existing technologies. Therefore, the solution implemented in this work is an application that uses the advantages of this language, and ensures safer, more efficient and sustainable web development.*

Keywords: *web application, Rust, Web Assembly*

1. УВОД

Мотивација иза овог истраживања лежи у потрази за ефикаснијим, сигурнијим и свестранијим развојем веб апликација. Како потражња за веб апликацијама наставља да расте, расту и изазови повезани са стварањем робусног, скалабилног и поузданог софтвера.

На мапи развоја веб апликација традиционално су доминирали језици као што су Јава, Јаваскрипт (енгл. Javascript) и Пајтон (енгл. Python). Иако су се ови језици показали ефикасним, они такође уводе забринутост за безбедност и перформансе због свог динамичког куцања и управљања меморијом. Програмски језик Раст, познат по својим гаранцијама за безбедност меморије и оптимизацији перформанси, нуди обећавајући пут за решавање ових изазова [1].

Овај рад настоји да дефинише тренутно стање веб развоја у Расту, идентификује недостатке и изазове и пружи радни пример апликације у потпуности изграђене у Расту, наглашавајући њен потенцијал за побољшану безбедност, перформансе и могућност одржавања [1].

2. ОСНОВНИ ТЕОРИЈСКИ КОНЦЕПТИ ПРОГРАМСКОГ ЈЕЗИКА РАСТ

Раст представља један од ретких новијих језика који је доживео оволику експанзију. Он сигурно испуњава тржишну нишу - има за циљ да помогне системским програмерима и другима који желе да креирају код који истовремено барата са десетинама, хиљадама или чак милионима догађаја [2].

2.1 Основе програмског језика Раст

Раст је програмски језик опште намене, првенствено креиран да обезбеди високе перформансе сличне онима који нуде језици Це (енгл. *C*) и Це++ (енгл. *C++*), као и са посебним нагласком на безбедност кода, где референце у коду увек упућују на исправну меморију. Такође, подршка за конкурентно програмирање, као и ефективни компајлер су још неки од разлога зашто су се познате софтверске компаније, као и многе компаније у развоју одлучиле за овај језик [1].

2.2 Специфичности и главне предности

Синтакса језика Раст је слична оној у језику Це++, с тим да решава и његове главне проблеме: грешке у меморији и конкурентно програмирање, и управо ово се сматра једном од његових главних предности [1].

Један од начина како Раст ово превазилази јесте концепт позајмљивања (енгл. *borrowing*) - део компајлера који обезбеђује да референце не надживе податке на које се односе.

Ова функционалност помаже у управљању меморијом, и они се откривају у време компајлирања те је прикупљање смећа самим тим непотребно [1].

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

2.3 Примена

Раст представља већ зрелу технологију која се користи у индустрији. Као системски програмски језик, омогућава потпуну контролу над детаљима ниског нивоа, где постоји могућност бирања да ли ће се подаци чувати на стеку (енгл. *stack*) или на хипу (енгл. *heap*) [1].

С обзиром да Раст нема „сакупљач смећа“ који непрекидно ради, његови пројекти могу да се користе као библиотеке од стране других програмских језика [1].

3. РАЗВОЈ КОМПЛЕТНЕ ВЕБ АПЛИКАЦИЈЕ У ОКВИРУ ПРОГРАМСКОГ ЈЕЗИКА РАСТ

Раст већ сада представља поуздани програмски језик; веб оквири као што су Рокет (енгл. *Rocket*), Актикс (енгл. Actix) и други омогућавају програмерима да креирају веб апликације са Растом, уз Дизел (енгл. Diesel) који представља најпродуктивнији начин интеракције са базама података у Расту [2].

Поред тога, постоје и бројни веб оквири у Расту који се могу користити за фронтенд (енгл. *frontend*) део веб апликације, које се користе у комбинацији са новим алатом: веб асембли (енгл. *webAssembly*) који представља мали, брзи бинарни формат који обећава приближне перформансе сходне веб апликацији [2].

3.1 Радни оквир Рокет

Рокет представља радни оквир за Раст, који има за циљ да буде брз, лак и флексибилан док нуди загарантовану сигурност и што је могуће већу безбедност. Обезбеђује примитиве за прављење веб сервера и апликација са Растом, као што су: рутирање, претходну обраду захтева и накнадну обраду одговора [3].
Дизајн овог радног оквира је усредсређен на три основна концепта које диктирају сам Рокетов интерфејс:

- Сигурност, исправност и искуство програмера су најважнији.
- Све информације о руковању захтевима треба да буду откуцане и самосталне.
- Одлуке не треба форсирати.

Главни задатак Рокета је да „ослушкује“ долазне веб захтеве, пошаље захтев апликацији и врати одговор клијенту. Процес који иде од захтева до одговора може да се представи као „животни циклус“ који се састоји из рутирања, валидације, обраде и одговора [3].

Апликације у овом радном оквиру су фокусиране на руте и хендлере. *Рута* представља комбинацију параметара који се поклапају са долазећим захтевом и хендлером, једноставном функцијом која прима произвољан број аргумената и враћа било који тип података, који служи за обраду захтева и враћање одговора (слика 1) [3].

```
#[get("/world")]                  // <- route attribute
fn world() -> &'static str {  // <- request handler
    "hello, world!"
}
```

Слика 1. *Пример дефинисања једне руте.*

Пре него што Рокет може да пошаље захтеве на руту, рута треба да буде увезана (енгл. *mounted*) на сервер, и након тога, Рокет почиње да сервира захтеве након покретања (слика 2) [3].

```
#[launch]
fn rocket() -> _ {
    rocket::build().mount("/hello", routes![world])
}
```

Слика 2. *Покретање серверске апликације*

Обрада података тела захтева, као и већи део Рокета је навођена на основу типа. Да би хендлер знао да очекује податке из тела захтева, неопходна је анотација типа *data* = „параметар“, где је параметар аргумент који мора да имплементира особину *FromData* (слика 3) [3].

```
#[post("/", data = "<input>")]
fn new(input: T) { /* .. */ }
```

Слика 3. *Захтев са параметром* ***data***

Свака обрада апликације може довести до одређених грешака. У оквиру Рокет радног оквира, грешке могу да настану са више различитих места (неуспешан одговор, грешка при рутирању) [3].

Рокет у себи садржи концепт који дозвољава програмерима да ручно обрађују грешке – хватачи грешака (енгл. *error catchers*). Ако се било која од ових наведених грешака догоди, Рокет враћа грешку клијенту. Да генерише грешку, Рокет позива хватач који одговара статусном коду грешке и опсегу (слика 4). [3]

```
#[catch(404)]
fn not_found(req: &Request) { /* .. */ }
```

Слика 4. *Хватач грешке статусног кода 404*

3.2 Веб Асемблы

Веб Асемблы представља мали, брзи бинарни формат који обећава скоро изворне перформансе за веб апликације. Осим тога, дизајниран је да буде компајлиран за било који језик, где јаваскрипт представља само један од њих [5].

Веб Асемблы је дизајниран са великим бројем случајева коришћења заснованих на претраживачу који захтевају велике перформансе: игре, стриминг музике, уређивање видеа, шифровање и препознавање слика. Иако је тек у повоју, овај нови језик има за намеру да буде што кориснији могуће кроз низ иницијатива, као што су: примитиве за сакупљање смећа, рад са нитима и алати за отклањање грешака (дебаговање) [4].

3.3 Радни оквир Ју

Ју се истиче као значајни новитет у домену веб развоја, који не само да се издваја од традиционалних

оквира за веб развој, већ служи и као доказ његовог потенцијала да редефинише начин на који се веб апликације праве [8].

Као можда и најважнија особина Јy радног оквира, могуће је писати изразе који подсећају на основни ХТМЛ користећи *html*! макро. Овај макро претвара код написан у прилагођеној синтакси сличној ХТМЛ-у у Раст код. Постоји само једно главно правило код употребе *html*! макроа – повратна вредност мора бити само један коначно затварајући елемент (слика 5) [6].

```
let header_text = "Hello world".to_string();
let header_html: Html = html! {
    <h1>{header_text}</h1>
};

let count: usize = 5;
let counter_html: Html = html! {
    <p>{"My age is: "}{count}</p>
};

let combined_html: Html = html! {
    <div>{header_html}{counter_html}</div>
};
```

Слика 5. *Коришћење макроа* ***html!***

Елементарну градивну јединицу овог радног оквира шредстављају *компоненте*. Неке од њених главних особина су [6]:

- узимају аргументе у облику *Props*-а,
- могу имати сопствено стање,
- обрађују делове ХТМЛ-а који су видљиви кориснику.

За креирање функције која је компонента, додаје јој се атрибут *#[function_component]* и приликом рендеровања, Јy ће изградити виртуелно стабло ових компоненти. Позваће функцију приказа сваке (функције) компоненте да би израчунала виртуелну верзију ДОМ-а (ВДОМ) коју ће корисник библиотеке видети као ХТМЛ тип (слика 6) [6].

```
use yew::{function_component, html, Html};

#[function_component]
fn HelloWorld() -> Html {
    html! { "Hello world" }
}

// Then somewhere else you can use the component inside `html!`
#[function_component]
fn App() -> Html {
    html! { <HelloWorld /> }
}
```

Слика 6. *Функција као компонента*

Атрибут #[function_component] такође дозвољава да функције опционо приме Пропс у аргументима. Да би их обезбедили, они се додељују преко атрибута у html! макроу.

Својство које имплементира компонента (енгл. Component) садржи два асоцирана типа: раније поменути Props и Message.

Тип Message се користи за слање порука компоненти након што се десио догађај; на пример, уколико постоји радња која треба да се догоди када корисник кликне на дугме или помери страницу надоле (слика 7) [6].

```
impl Component for MyComponent {
    type Message = Msg;
    type Properties = Props;

    // ...
}
```

```
enum Msg {
    Click,
    FormInput(String)
}
```

Слика 7. *Асоцирани тип* ***Message***

Поред ових типова, да би се одговорило на захтев, неопходно је користити и „повратне позиве“ (енгл. *callbacks*), који иницијално служе како би се комуницирало са услугама, агентима и надређеним компонентама унутар Јуа (слика 8) [6].

```
    fn view(&self, ctx: &Context<Self>) -> Html {
        let onclick = ctx.link().callback(|_| Msg::Clicked);
        html! {
            <button {onclick}>{ "Click" }</button>
        }
    }
}
```

Слика 8. *Пример повратног позива*

4. ИМПЛЕМЕНТАЦИЈА И ДЕТАЉАН ПРИКАЗ КОМПЛЕТНЕ ВЕБ АПЛИКАЦИЈЕ У ПРОГРАМСКОМ ЈЕЗИКУ РАСТ

Имплементација и детаљан приказ комплетне веб апликације у програмском језику Раст представља имплементацију основних функционалности типичне за једну веб страницу (где је као основни ентитет рада струкутра машина) уз додатак који представља учитавање КјуАр (енгл. QR) кода.

4.1 Архитектура система

Имплементирани систем има трослојну архитектуру која се састоји из:

- корисничког интерфејса - који служи за интеракцију са корисницима (радни оквир Јy),
- серверског дела - где је смештена сва логика (радни оквир Рокет),
- базе података - где се чувају подаци.

Улогу корисника у овом систему представљају радници индустријског предузећа, док је намена апликације основно руковање са ентитетом *машина*, како би помогла радницима у лакшем вођењу њихове евиденције, и како би им пружила брзи преглед информација о машинама на мобилном телефону уз помоћ КјуАр кода.

4.2 Пример описа случаја коришћења

У наставку ће детаљније бити приказана функционалност преглед машина, где ће бити описана њена имплементација, како серверска страна, тако и кориснички интерфејс, како би се представио рад саме веб апликације.

Почетна страница рендерује компоненту *List* која у себи садржи пропс поље: машине. Машине се у оквиру иницијалне компоненте *MachineApp* добављају на свако ново учитавање странице, како би корисник у сваком тренутку имао ажурни приказ свих машина (слика 9).

Слика 9. *Добављање машина*

Директно на *List* компоненти, врши се рендеровање података и преко животног циклуса компоненте и функције *view* (која ће у себи позвати функцију *render_list*) приказују се подаци (слика 10).

Слика 10. *Приказ машина – клијентски део*

Корисницима то изгледа као на слици 11.

Слика 11. *Приказ машина – претраживач*

5. ЗАКЉУЧАК

У оквиру овог рада представљена је комплетна веб апликација, везана за основне функционалности у раду са ентитетом машина. Целокупна апликација је реализована у програмском језику Раст, с тим да је за кориснички интерфејс коришћен радни оквир Ју, док је за серверски део коришћен радни оквир Рокет.

Главна мотивација за развијање веб апликације баш у оквиру овог језика јесте изазов - приближити модерни веб развој апликација као нешто што је свакодневница у оквиру програмерског света са традиционалним језиком какав је Раст који води рачуна меморији и о програмима ниског нивоа.

Развој комплетне веб апликације у Расту наглашава његов изузетан потенцијал у развоју модерних веб апликација. Успешно премошћује јаз између програмирања ниског нивоа и веб развоја на високом нивоу, користећи Растову безбедност меморије, перформансе и архитектуру компоненти Ју. Иако и даље постоје изазови рада са строгим правилима његовог компајлера и библиотекама које се развијају, овај рад доказује да је Раст моћан и безбедан језика за прављење веб апликација.

6. ЛИТЕРАТУРА

[1] Krzysztof Wrobel, „Why is Rust programming language so popular?”, урл: https://codilime.com/blog/why-is-rust-programming-language-so-popular/ [Последњи приступ: .10.2023.]

[2] Eze Sunday, “How to create a web app in Rust with Rocket and Diesel.”, урл: https://blog.logrocket.com/create-web-app-rust-rocket-diesel/ [Последњи приступ: 7.10.2023.]

[3] Get Started with Rocket, урл: https://rocket.rs/ [Последњи приступ: 5.10.2023.]

[4] Serdar Yegulalp, “What is WebAssembly? The next-generation web platform explained.”, урл: https://www.infoworld.com/article/3291780/what-is-webassembly-the-next-generation-web-platform-explained.html [Последњи приступ: 10.10.2023.]

[5] Eric Elliott, “What is WebAssembly? The Dawn of a New Era.”, урл: https://medium.com/javascript-scene/what-is-webassembly-the-dawn-of-a-new-era-61256ec5a8f6 [Последњи приступ: 10.10.2023.]

[6] Get Started with Yew,урл: https://yew.rs/ [Последњи приступ: 10.10.2023.]

[7] Get Started with Diesel, урл: https://diesel.rs/ [Последњи приступ: 5.10.2023.]

[8] Demola Malomo, “Exploring Yew – The Rust-based Frontend Framework”, урл: https://malomz.medium.com/exploring-yew-the-rust-based-frontend-framework-as-a-react-developer-bde10dd95dea [Последњи приступ: 14.10.2023.]

[9] Alice and Bob, “Building a website in Rust using Rocket and Yew”, урл: https://theadventuresofaliceandbob.com/posts/rust_rocket_yew_part1.md [Последњи приступ: 15.10.2023.]

Кратка биографија:

Огњен Богдановић рођен је у Руми 12. јуна 1999. године. Факултет техничких наука у Новом Саду, студијски програм Рачунарство и аутоматика уписао је 2018. године. Након завршених основних студија, 2022. године, уписао је мастер академске студије из исте области.

контакт:
bogdanovicognjen@gmail.com