გია სურგულაძე, სანდრო დოლიძე

# მომხმარებლის ინტერფეისის დაპროგრამება

## (AngularJS, ReactJS)


„სტუ-ს IT კომპეტენციის ცენტრი"

გია სურგულაძე, სანდრო დოლიძე

# მომხმარებლის ინტერფეისის დაპროგრამება (AngularJS, ReactJS)


დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის "სარე-დაქციო კოლეგიის მიერ

თბილისი
2019

**უაკ 004.5**

განხილულია თანამედროვე ვებ-ტექნოლოგიების განვითარებადი სფერო, კერძოდ, ვებ-აპლიკაციების აგების ძირითადი პრინციპები და ტექნოლოგიები, რომლებიც მასშტაბირებადი (scalable) back-end და front-end სისტემებისთვის გამოიყენება და აქტუალურია. ტრადიციულ HTML, CSS, PHP, BEM მეთოდოლოგიებთან ერთად წარმოდგენილია JavaScript ენის განვითარებით მიღებული ტექნოლოგიები, როგორიცაა AngularJS, Node.js, Ajax და ReactJS ფრეიმვორკები და ბიბლიოთეკები. ბოლო წლებში ამ სისტემებმა კომპიუტერული პროგრამების ბაზარზე საკმაო პოპულარობა მოიპოვა ვებ-დეველოპერებში. ისინი დაცული და მრავალი ფუნქციით აღჭურვილი საშუალებებია, რომლებიც ვებ-აპლიკაციის შექმნას საგრძნობლად ამარტივებს და აუმჯობესებს მწარმოებლურობის ამაღლების თვალსაზრისით. დამხმარე სახელმძღვანელოს ორიგინალური მხარეა React Hooks ინსტრუმენტის გამოყენების კვლევა ვებ-აპლიკაციის სუსტი ადგილების გამოვლენისა და მწარმოებლურობის ამაღლების პრობლემების გადაწყვეტის მიზნით. შედეგი აისახება პროგრამული სისტემის ხარისხის სრულყოფაში, მისი ფუნქციონირების სწრაფქმედების მახასიათებლის გაუმჯობესების საფუძველზე.

**რეცენზენტები:**

- პროფ. გ.ღვინეფაძე
- პროფ. ე. თურქია

**რედკოლეგია:**

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, ნ. ლომინაძე, ჰ. მელაძე, თ. ობგაძე, გ. სურგულაძე (რეადაქტორი), გ. ჩაჩანიძე, ა. ცინცაძე, ზ. წვერაიძე



**ISBN 978-9941-8-0625-4**

**Gia Surguladze, Sandro Dolidze**

# USER INTERFACE DEVELOPMENT
## (AngularJS, ReactJS)

This book deals with the development of advanced web technologies, basic principles of web applications that are widely used. We have presented modern technologies for building web applications that are used for scalable back-end and front-end systems. Together with traditional HTML, CSS, PHP and BEM methodologies we have presented technologies developed by JavaScript language such as AngularJS, Node.js, Ajax and ReactJS Freeworks and libraries. In recent years, these systems have gained popularity among web developers in the computer software market. They are protected and equipped with a variety of features that greatly simplify the creation of web applications and increase productivity. The original aspect of the work is to explore the use of the React Hooks tool to identify vulnerable points in a web application and solve performance problems. The result is reflected in the improvement of the quality of the software package based on the improvement of the performance indicators of its work.


**ISBN 978-9941-8-0625-4**

## *„ვუძღვნით სახელოვანი მეცნიერისა და საზოგადო მოღვაწის, პროფესორ თეიმურაზ დოლიძის ნათელ ხსოვნას (1955–2012)“*

თეიმურაზ დოლიძე დაიბადა 1955 წ. 28 მარტს ქუთაისში. წარჩინებით დაამთავრა საშუალო სკოლა და 1972 წ. სწავლა განაგრძო თსუ-ს კიბერნეტიკისა და გამოყენებითი მათემატიკის ფაკულტეტზე „მენეჯმენტის ინფორმაციული სისტემების პროგრამული უზრუნველყოფის“ სპეციალობით. 1983-1987 წლებში სწავლობდა სანქტ-პეტერბურგის ჩრდილო-დასავლეთის ტექნიკური უნივერსიტეტის „მართვის საინფორმაციო სისტემების" ასპირანტურაში, აქვე, დაიცვა დსერტაცია ტექნიკის მეცნიერებათა კანდიდატის (Ph.D.) ხარისხით.

1996-1999 წ. იყო შვეიცარია-საქართველოს ერთობლივი სამედიცინო დიაგნოსტიკური ცენტრის „CITO“ დირექტორი. 2002-2004 წლებში გაიარა სამეცნიერო და საერთაშორისო ბიზნესის სტაჟირება ტურინსა (იტალია) და ლონდონში „საჯარო სამსახურის რეფორმის მართვა"; ჰარვარდის უნივერსიტეტში (ბოსტონი, მასაჩუსეტსი, აშშ) „სამეწარმეო მენეჯმენტი ჯანდაცვის სექტორის რეფორმაში: ბიზნეს დაგეგმარება სამთავრობო და არასამთავრობო ორგანიზაციებისათვის". იყო ჯანდაცვის სამინისტროს ხაზით მსოფლიო ბანკის პროექტის „საქართველოს საავადმყოფოების რესტრუქტურიზაცია“ აღმასრულებელი დირექტორი. მისი უშუალო ხელმძღვანელობით აღდგენილი და გადაიარაღებულ იქნა 20-ზე მეტი საავადმყოფო თბილისსა და რეგიონებში.

თ. დოლიძე არის 50-ზე მეტი სამეცნიერო ნაშრომისა და ინოვაციური პროექტის ავტორი: ჯანდაცვისა და მრეწველობის მენეჯმენტის საინფორმაციო სისტემების, გადაწყვეტილებათა მიღების სისტემების სფეროში. 2006 წ. მისი ინიციატივით შეიქმნა ინტერნეტ საძიებო სისტემა www.e-info.ge. იყო საერთაშორისო სიმპოზიუმების, სემინარების მონაწილე აშშ-ის. დიდი ბრიტანეთის, საფრანგეთის, იტალიის და სხვ. ქვეყნებში. სტუ-ს „პროგრამული ინჟინერიის“ კათედრის პროფესორი, უმაღლესი დონის მეცნიერ-პედაგოგი მენეჯმენტსა და პროგრამულ ტექნოლოგიებში, ინგლისური და რუსული ენების უნაკლოდ მცოდნე, სტუდენტების დიდი მეგობარი და დამრიგებელი, ქართული პოეზიის, სიმღერის და საქართველოს ისტორიის დიდი ქომაგი.

# შინაარსი

## გამოყენებული აბრევიატურების ნუსხა

AJAX – Asynchronous JavaScript and XML
AJS – AngularJs
API - Application programming interface
APP – Applied Software
ASP – Active Server Pages
AWS – Amazon Web Services
BEM – Block Element Modifier
CMS – Content Management System
CSS – Cascading Style Sheets
DOM – Document Object Model
EF – Entity Framework
ES6 – ECMAScript 6
HTML – Hypertext Markup Language
HP – Hypertext Prepocessor
JS - JavaScript
MIT – Massachusetts Institute of Technology
MVC – Model-View-Controller
PHP – Hypertext Preprocessor
SAAS – Syntactically Awesome Stylesheets
SPA – Single Page Application
SQL – Structured Query Language
TS - TypeScript
URL – Uniform Resource Locator

## *ავტორების შესახებ:*

**გია სურგულაძე** – 

ტექნიკის მეცნიერებათა დოქტორი, ინფორმატიზაციის საერთაშორისო აკადემიის ნამდვილი წევრი (1994 წლიდან, გაეროსთან არსებული IIA), საქართველოს ტექნიკური უნივერსიტეტის პროფესორი. ინფორმატიკის ფაკულტეტის „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის)“ აკადემიური დეპარტამენტის უფროსი. 76 წიგნის (მათ შორის 16 მონოგრაფია, 60 სახელმძღვანელო), 55 ელ-სახელმძღვანელოს და 300-მდე სამეცნიერო ნაშრომის ავტორი მართვის საინფორმაციო სისტემების, პროგრამული ინჟინერიის, მონაცემთა რელაციური და NoSQL ბაზების, დაპროგრამების ჰიბრიდული ტექნოლოგიების, იმიტაციური მოდელირების, პეტრის ფერადი ქსელების, რიგების თეორიის და სხვ. სფეროში.

**სანდრო დოლიძე** - 

დაბ.თარიღი 5.11.1993. ინფორმატიკის მაგისტრი (საქართველოს ტექნიკური უნივერსიტეტი 2019 -100% დაფინანსებით). თბილისის თავისუფალი უნივერსიტეტის „მათემატიკისა და კომპიუტერულ მეცნიერებათა სკოლა" (100%-დაფინანსებით), კომპიუტინგის ბაკალავრი (2011-2016).

აქვს საზღვარგარეთის და საქართველოს ფირმებში მუშაობის პრაქტიკული გამოცდილება ვებ- და დესკტოპ სისტემების დეველოპინგის სფეროში. კერძოდ, ციურიხი, შვეიცარია (03-06. 2016), Google-ს Software Engineering Intern; კანადა (09. 2018) Flexdealer (Toptal), Senior Full Stack Developer; თბილისი: „Alta Software“ (20117-18) და „Vidal“ (2012-16) - დეველოპერი.

***ტექნიკური უნარები*** - TypeScript, JavaScript, C #, Java და სხვ.

***ინტერესები:*** React, Node, AWS, Docker, Rx, GraphQL, Redux, Functional Programming, Event sourcing.

# შესავალი

პროგრამული ინჟინერიის სფეროში გამოყენებითი პროგრამული უზრუნველყოფის (Applied Seftware) ანუ დესკტოპ- და ვებ-აპლიკაციების დიდი პროექტების დამუშავება მნიშვნელოვანი მიმართულებაა [1]. განსაკუთრებული ყურადღება ექცევა აქ პროგრამული სისტემის სწორი მასშტაბირების შერჩევის საკითხს [2]. იგი, როგორც წესი, უნდა გადაწყდეს დაპროექტების ეტაპის დაწყებამდე, რადგან ცვლილებების განხორციელება პროექტირების დასრულების შემდეგ გაცილებით მეტ დროს და ხარჯებს მოითხოვს.

მასშტაბირება - (Scalable) არის ვებ-აპლიკაციის პროექტის დამახასიათებელი თვისება, რომ მას შეეძლოს თავისი შესაძლებლობების გაფართოება ახალი ფუნქციების შეასაბამისი მოდულების რაოდენობის გაზრდით, რომლებიც ასრულებს იგივე ამოცანებს.

პროექტის მასშტაბირება რთული პროცესია. არსებობს მასშტაბირების სქემებისა და მეთოდების დიდი სიმრავლე: front-end, back-end, database [2,3].

- front-end: ასოცირდება მომხმარებლის ინტერფეისთან (ვებ-სისტემის კლიენტის მხარეს), ის, რასაც ბრაუზერი კითხულობს და შეუძლია ეკრანზე გამოტანა ან ამუშავება. ასეთი სისტემებია: HTML, CSS და JavaScript.

– HTML (Hyper Text Markup Language) ბრაუზერს აწვდის გვერდის შედგენილობას, მაგალითად, სათაური, პარაგრაფი,სია, ელემენტები და ა.შ.;

– CSS (Cascading Style Sheets) ბრაუზერს კარნახობს თუ როგორ ასახოს ელემენტები ეკრანზე. მაგალითად, „პირველი პარაგრაფის შემდეგ აბზაცი 25 პიკსელი“ ან „body ელემენტში მთლიანი ტექსტი იყოს ლურჯი და დაწერილი Sylfaen ფონტით“.

– JavaScript აძლევს ბრაუზერს მითითებებს, თუ როგორ იმოქმედოს გარკვეულ მოვლენებზე. ამისათვის იგი იყენებს დაპროგრამების მსუბუქ ენას და ცვლის ეკრანზე გვერდის შიგთავსს.

- back-end: ასოცირდება საიტის სერვერულ მხარესთან - სერვისის სახით. ყველაფერი, რაც მუშაობს სერვერზე (და არა ბრაუზერზე) ან კომპიუტერზე, რომელიც ჩართულია ინტერნეტში და პასუხობს სხვა მომხმარებელთა პერსონალური კომპიუტერების მოთხოვნებს. BackEnd - ისთვის გამოიყენება სერვერზე არსებული ყველა ინსტრუმენტი, რომელიც უზრუნველყოფს შეტყობინებების მომსახურებას. აქ გაოიყენება დაპროგრამების უნივერსალური ენები: Java, JavaScript/Node, PHP, Python და სხვ.
- Database - მონაცემთა ბაზების მართვის სისტემები გამოიყენება სერვერის მხარეს დაპროგრამების ენებთან და ტექნოლოგიებთან ერთად. აქ მოიაზრება რეალაციური და NoSQL მონაცემთა ბაზები: MsSQL_Server, PostgreSQL, Oracle, MySQL, MariaDB, MongoDB, Cassandra და სხვ.

➢ **front-end და back-end ურთიერთმოქმედების სტრუქტურა:**

არსებობს front-end და back-end -ის ურთიერთმოქმედების რამდენიმე ძირითადი არქიტექტურა:

- *სერვერული აპლიკაციები:* ამ შემთხვევაში HTTP-მოთხოვნები პირდაპირ ეგზავნება სერვერულ აპლიკაციას, ხოლო სერვერი პასუხობს HTML-გვერდით.

მიღებული მოთხოვნის საფუძველზე სერვერი ეძებს ინფორმაციას მონაცემთა ბაზაში და შედეგს ათავსებს პასუხის შაბლონში. როცა გვერდი ჩატვირთულია ბრაუზერში, HTML განსაზღვრავს თუ **რა** უნდა იყოს ნაჩვენები (DOM - Document Object Model), CSS – თუ **როგორ** უნდა გამოიყურებოდეს (CSSOM – CSS Object Model) და JS - ყველა სხვა ქმედება;

- *კავშირი AJAX-ით (Asynchronous JavaScript and XML):* ბრაუზერში ჩატვირთული JavaScript აგზავნის HTTP-მოთხოვნას

(XML HTTP Request) გვერდის შიგნიდან და ღებულობს XML-პასუხს. გამოიყენება ასევე JSON ფორმატი;

- *კლიენტების ერთგვერდიანი აპლიკაციები:* AJAX იძლევა საშუალებას ჩაიტვირთოს მონაცემები გვერდის ცვლილების გარეშე. ეს უმეტესად გამოიყენება ისეთ ფრეიმვორკებში, როგორიცაა Angular და Ember. აწყობის შემდეგ ასეთი დანართები იგზავნება ბრაუზერში, ხოლო კლიენტის მხარეს (ბრაუზერში) კი სრულდება ნებისმიერი მომდევნო რენდერინგი (DOM-ის ვიზუალური წარმოდგენა). აქ front-end ურთიერთობს back-end -თან HTTP-საფუძველზე JSON- ან XML-პასუხების გამოყენებით;
- *უნივერსალური / იზომორფული აპლიკაციები:* ზოგიერთი ბიბლიოთეკა და ფრეიმვორკი (მაგალითად, React და Ember) საშუალებას იძლევა აპლიკაცია შესრულდეს როგორც სერვერზე, ასევე კლიენტზე. ასეთ დროს front-end -ის კავშირისთვის back-end -თან აპლიკაცია იყენებს AJAX-საც და სერვერზე დასამუშავებელ HTTP-ს;

➢ **front-end და back-end -ის გარეშე**

- *ავტონომიური front-end:* ვებ-აპლიკაციების გამოყენების ეფექტურობის ერთ-ერთი მახასიათებელია მათი ქსელთან მიერთების აუცილებლობა. პროგრესული ვებ-აპლიკაციები მხოლოდ ერთხელ ჩაიტვირთება და შემდეგ მუშაობს მუდმივად. ბრაუზერში შეიძლება მონაცემთა ბაზის შენახვა. ვებ აპლიკაციას დასჭირდება back-end მხოლოდ პირველი ჩატვირთვისა და შემდეგ მონაცემების სინქრონიზაციისა / დაცვისათვის. ასეთ შემთხვევაში აპლიკაციის ლოგიკის დიდი ნაწილი განთავსებული იქნება კლიენტის მხარეს;
- *მსუბუქი **back-end** :* დროთა განმავლობაში back-end ხდება სულ უფრო მსუბუქი. დოკუმენტების საცავებისა და მონაცემთა გრაფული ბაზების ტექნოლოგიები საგრძნობლად ამცირებენ back-end -თან მიმართვის რაოდენობას მონაცემთა ხელმეორედ აგრეგირების მიზნით. კლიენტის ამოცანაა განსაზღვროს თუ

რომელი მონაცემები სჭირდება მას (გრაფული მბ), ან ამოიღოს მონაცემთა ყველა განსხვავებული ფრაგმენტი, რომლებიც მას უნდა. შესაძლებელია back-end სერვისების შექმნა, რომლებიც მუშაობს არა ყოველთვის, არამედ როცა საჭიროა. აქ მუშაობს უსერვერო არქიტექტურა, როგორიცაა მაგალითად, AWS Lambda (Amazon Web Services);

- *არამკაფიო საზღვრები:* გამოთვლითი ამოცანები შესაძლებელია გადაადგილდეს front-end და back-end -ს შორის. აპლიკაციის სახის შესაბამისად გამოთვლები შეიძლება განხორციელდეს ან სერვერზე ან კლიენტზე. ორივეს აქვს თავისი დადებითი და უარყოფითი მხარეები. სერვერი შედარებით სტაბილურია და აქვს მეტი რესურსები. მაგრამ იგი სულ ჩართული უნდა იყოს ქსელში. მრავალი მომხმარებელი იყენებს ბოლო ვერსიის ბრაუზერებს, რომლებიც სწრაფად ასრულებენ რთულ გამოთვლებსაც და აქვთ უკეთესი ინტერფეისები.

➢ Node.js არის back-end თუ front-end ?

Node.js პროგრამული პლატფორმა თავიდან შეიქმნა Google Chrom-ის JavaScript სამუშაო გარემოსთვის სწარაფი და მასშტაბირებადი ქსელური აპლიკაციების მარტივად ასაგებად. მისი V8 შესრულების მექანიზმი, რომელიც დაწერილია C++ ენაზე, უზრუნველყოფს JavaScript კოდის კომპილირებას მანქანურ კოდში. Node.js გამოიყენება ძირითადად სერვერის მხარეს, როგორც back-end. შესაძლებელია მისი გამოყენება ასევე მომხმარებლის მხარესაც, როგორც front-end, მაგრამ თანამედროვე აქტუალური ენების (მაგალითად, Angular და სხვ.) პოპულარულ ინტერფეი-სებში ეს არაა რეკომენდებული.

Node.js ხელს უწყობს JavaScrip-ის შესაძლებლობების გაფართოებას C++ და სხვა ენების გარე ბიბილიოთეკების მისაერთებლად [4].

მართალია, Node.js გამოიყენება ძირითადად სერვერზე, ვებ-სერვერის როლში, მაგრამ არსებობს ასევე შესაძლებლობა მისი

საშუალებით დესკტოპის ფანჯრული დანართების ასაგებად (მაგალითად, NW.js და AppJS) და მიკროკონტროლერების დასაპროგრამებლად (მაგალითად, tessel da esprunio) [5].

Node.js პლატფორმას საფუძვლად უდევს მოვლენებზე-ორიენტირებული პროგრამირება (event-driven programming) და ასინქრონული (ან რეაქტიული) პროგრამირება (reactive programming) [6,7]. ჩვენ ამ საკითხებს მომავალშიც განვიხილავთ უფრო დეტალურად.

წინამდებარე ნაშრომში განხილულია თანამედროვე ვებ-ტექნოლოგიების განვითარებადი სფერო. ვებ-აპლიკაციების ძირითადი პრინციპები, რომლებიც ფართოდ გამოიყენება. დაგვჭირდება ქსელთან დაკავშირებლი მოწყობილობა, ლეპტოპი, პლანშეტი, სმარტფონი და ინტერნეტში შესასვლელი პროგრამა (internet browser).

წარმოდგენილი გვაქვს აგრეთვე ის თანამედროვე ტექნოლოგიები, რომლებიც მათ შესაქმნელად გამოიყენება და აქტუალურია, როგორიცაა: HTML, CSS, PHP, BEM მეთოდოლოგია [8,57]. ხოლო ყველაზე მთავარი რასაც ჩვენი ნაშრომი მოიცავს ეს გახლავთ Angular და ReactJS სისტემების განხილვა, რომლებმაც ბაზარზე საკმაო პოპულარობა მოიპოვა ვებ-დეველოპერებში (ნახ.1). ისინი დაცული და მრავალი ფუნქციით აღჭურვილი საშუალებებია, რომლებიც ვებ-აპლიკაციის შექმნას საგრძნობლად ამარტივებს და აუმჯობესებს მწარმოებლურობის ამაღლების თვალსაზრისით.


**ნახ.1. პოპულარული ვებ-ფრეიმვორკები**

თანამედროვე ტექნოლოგიებს თავისი დადებით და უარყოფითი მხარეებიც გააჩნია. კარგია მაგალითად ის, რომ კომუნალური გადასახადების გადახდისთვის აღარ უწევთ ადამიანებს კონკრეტულ ადგილებში მისვლა და რიგებში დგომა, რადგან აღნიშნული ოპერაციის განხორციელება შესაძლებელია ტერმინალების საშუალებით ან უბრალოდ ინტერნეტთან ჩართული მოწყობილობებით.

უარყოფითი მხარეებია ის, რომ თუ თანამედროვე ტექნოლოგიებს ხანდახან ხარვეზებიც ახასიათებთ, და აგრეთვე ყველაზე დიდი ნაკლი, რაც ტექნოლოგიების გამოყენებას მოჰყვა, ეს არის ცოცხალი ურთიერთობების ჩანაცვლება ტექნიკური საშუალებებით თუ სოციალური ქსელებით, ვირტუალურმა სამყარომ რეალობას მოწყვიტა ბევრი ადამიანი, რის გამოც საზღარგარეთ უკვე არსებობს მკურნალობის ციკლები სოციალური ქსელით დაავადებული ადამიანებისთვის [9].

დღეისათვის კომპიუტერულ მოწყობილობად არამარტო პერსონალური კომპიუტერი ან ლეპტოპი მოიაზრება, არამედ სმარტფონებიც, რომლებიც მცირე ზომის კომპიუტერული მოწყობილობებია. შეიძლება გამოვყოთ ორი სახის მოთხოვნადი პროგრამული აპლიკაცია, რომლებიც ფართოდ მოიხმარება, ესენია სმარტფონებისთვის განკუთვნილი აპლიკაციები (Android, iOS) და ვებ აპლიკაციები [10]. პროგრამისტები, რომლებიც ერთ-ერთ ან ორივე სფეროს ფლობენ, მათ დასაქმების პრობლემა არ აქვთ, როგორც უცხოეთში, ასევე ჩვენს ქვეყანაში. რა თქმა უნდა, ორივე სფერო მნიშვნელოვანია, ორივე სფერო პროგრამირების ცოდნას მოითხოვს, თუმცა პროგრამირების ენები და ტექნოლოგიები, რომელთა მეშვეობით ხდება სმარტფონ და ვებ-აპლიკაციების შექმნა განსხვავებულია. ასევე სმარტფონ აპლიკაციების შექმნა android სისტემისთვის სხვა საშუალებებით ხდება, ios ისთვის სხვა [11,12].

ჩვენი ნაშრომის საკვლევი თემატიკა, რომელიც მომდევნო თავებში იქნება განხილული, ეხება თანამედროვე ვებ-აპლიკაციებს, მათ განვითარებას, გამოყენებას, ინტერფეისების ანალიზს. თუ ადრე ვებ-აპლიკაცია მორგებული იყო desktop ტიპის ეკრანებზე, დღევანდელ დღეს ე.წ. responsive web design ის საშუალებით მორგებულია სხვადასხვა ზომის ეკრანებზე, რაც დიდი დადებითი მხარეა.

# I თავი: ლიტერატურული მიმოხილვა: მასშტაბირებადი ვებ-აპლიკაციების აგების ინსტრუმენტული საშუალებების ანალიზი

## 1.1. ვებ-გვერდი, საიტი და აპლიკაცია

*ვებ-გვერდი* ფაილების ერთობლიობაა, რომელიც განთავსებულია სერვერზე, მასთან წვდომისთვის საჭიროა მხოლოდ მოწყობილობა, რომელიც ჩართულია ინტერნეტში (სმარტფონი, პლანშეტი, ლეპტოპი, პერსონალური კომპიუტერი).

*ვებ-საიტი* არის ვებ-გვერდების, სურათების, ვიდეოების და ა.შ. ერთობლიობა, რომელიც ატვირთულია ერთ ან რამდენიმე სერვერზე და რომელიც, ჩვეულებრივ, ხელმისაწვდომია ინტერნეტით, ტელეფონით. HTTP/HTTPS პროტოკოლის საშუალებით [13].

მსოფლიოში პირველი საიტი შეიქმნა 1991 წელს, შემქმნელი ტიმ ბერნერს-ლი. მასში განთავსებული იყო ინფორმაცია ახალი ტექნოლოგიის შესახებ World Wide Web (WWW) (მსოფლიოს ფართო ქსელი). ინფორმაციის გადაცემა დაფუძნებული იყო HTTP პროტოკოლზე, ხოლო საიტი შექმნილი იყო ვებ პროგრამირების HTML ენაზე. საიტზე ასევე განხილული იყო ვებ სერვერის და ვებ ბრაუზერის მუშაობის პრინციპები. საიტი ასევე გახდა მსოფლიოში პირველი ინტერნეტ კატალოგი, რადგან მოგვიანებით ბერნერს ლიმ მასზე განათავსა სხვა საიტის მისამართები. ყველაფერი რაც საჭირო იყო პირველი საიტის ასამუშავებლად ბერნერს ლიმ გაცილებით ადრე 1990 წელს შექმნა. ამ დროს შეიქმნა პირველი ვებბრაუზერი და პირველი სერვერი, რომელიც დაფუძნებული იყო NeXTcube-ის ბაზაზე.

*ვებ-აპლიკაცია* არის კლიენტ-სერვერული აპლიკაცია, რომელშიც კლიენტი უკავშირდება სერვერს ბრაუზერის გამოყენებით. ვებ-სერვერი პასუხისმგებელია სერვერის მუშაობაზე. ვებ-აპლიკაციის ლოგიკა განაწილებულია სერვერსა და კლიენტს შორის, მონაცემები ინახება უმეტესად სერვერზე,

ინფორმაციის გაცვლა ხდება ქსელის საშუალებით. ამ მიდგომის ერთ-ერთი უპირატესობაა ის ფაქტი, რომ კლიენტები არ არიან დამოკიდებული კონკრეტულ მომხმარებელთა ოპერაციულ სისტემაზე, ამიტომ ვებ აპლიკაციები პლატფორმათაშორისო სერვისებია.

ვებ-გვერდი შედგება ორი მთავარი ნაწილისგან, რომლებსაც ეწოდებათ: Front-End და Back-End. განვიხილოთ 1.1 ნახაზი. რას წარმოადგენს თითოეული მათგანი ?


**ნახ.1.1. Front-End და Back-End**

Front-end არის ვებ-აპლიკაციის ის ნაწილი, რომლის დანახვა მის მომხმარებლებს შეუძლიათ. ვებ - ტექნოლოგიების სპეციალისტი შეიძლება ფლობდეს როგორც front-end აგრეთვე back-end მხარეს, თუმცა შესაძლებელია მხოლოდ ერთი მიმართულებით იყოს განვითარებული, მაგალითად front მხარეს, ასეთი სახის სპეციალისტს, შეგვიძლია front-end დეველოპერი ვუწოდოთ.

თავისუფლად შეგვიძლია ვთქვათ, რომ საიტის ვიზუალურ მხარეს დიდი ეფექტის მოხდენა შეუძლია კლიენტზე, რა თქმა

უნდა საიტის სპეციფიკაც გასათვალისწინებელია, არსებობს საიტები, მაგალითად უნივერსიტეტის ელექტრონული ჟურნალები, რომლებიც ვიზუალურად შეიძლება არც გამოირჩეოდნენ, მათი მთავარი მიზანია სისწრაფე, და სიმარტივე, ვიზუალური ეფექტებით დატვირთვა საიტის, მის სისწრაფეზე უარყოფითად აისახება, თუმცა თუ საქმე გვაქვს მაღალი დონის მონაცემების კომპიუტერთან და მაღალ სიჩქარიან ინტერნეტთან, აღნიშნული პრობლემა ფაქტიურად შეუმჩნეველია, მაგრამ უნდა გავითვალისწინოთ გლობალური მომხმარებლები, და მათი რიცხვი, ამ შემთხვევაში ჯობს საიტი იყოს სწრაფი. თუმცა ვიზუალურ მხარეს, დიდი მნიშვნელობა აქვს, როგორც უკვე აღვნიშნეთ, თუ საიტის back-end მხარე (რომელიც სერვერის მხარეს წარმოადგენს, და მასზე მოგვიანებით ვილაპარაკებთ) საუკეთესო დონეზეა შესრულებული, მაგრამ საიტის ვიზუალური მხარე მოძველებულია, ან არ არის ვიზუალურად გამართული, მაშინ შეგვიძლია შევადაროთ კარგი ძრავის და მონაცემების მანქანას, რომლის ვიზუალური მხარე ან ნაავარიევია, ან თანამედროვეობას არ შეესაბამება, მიუხედავად იმისა მისი სამართავი არა ხილული ნაწილები გამართულია, ალბათ ცოტა თუ მოიძებნება ისეთი მყიდველებს შორის, ვინც მსგავს ავტომობილზე შეაჩერებს აქცენტს, რადგან მყიდველი ძირითადად დაინტერესებულია როგორც ვიზუალური, ასევე ძრავის და სხვა შიგა ნაწილების გამართულობით.

საიტის front-end მხარის დამზადება ხდება საშუალებებით, როგორიცაა :

- HTML
- CSS
- Javascript

არსებობს AngularJs, რომელიც JavaScript-ის გაფართოებითაა მიღებული. ის აგრეთვე front-end დეველოპმენტში მოიაზრება. აღნიშნული ენები წარმოადგენენ საიტის ვიზუალური მხარის

შექმნის საშუალებებს, მათი წაკითხვა ხდება ინტერნეტ ბრაუზერების დახმარებით, როდესაც ინტერნეტ ბრაუზერით ვახდენთ საიტის გამოძახებას, მაგალითად ინტერნეტ ბრაუზერის URL-ში იწერება http://www.gtu.ge დომეინი, მასთან დაკავშირებისას, მოხდება სერვერზე არსებული ფაილებთან წვდომა და ჩვენი მოთხოვნის შესაბამისი მონაცემების გამოგზავნა „არხით“ რომელსაც HTTP პროტოკოლი ეწოდება (Hypertext Transfer Protocol). იგი ახდენს hypertext ის ჩამოტვირთვას ჩვენს ბრაუზერში.

საყურადღებოა აგრეთვე TypeScript - პროგრამირების ენაა, რომელიც Microsoft-მა 2012 წელს წარმოადგინა და ვებ-აპლიკაციის დამუშავების ინსტრუმენტია, რომელიც ვრცელდება Java Script Developer-ის შესაძლებლობებით [15,58]. TypeScript-ის ავტორია დანიის ტექნიკური უნივერსიტეტის პროფესორი ანდერს ჰეილსბერგი (Anders Hejlsberg), რომელმაც მანამდე შექმნა Turbo Pascal, Delphi და C# [16].

TypeScript თავსებადია JavaScript-თან და კომპილირედბა მასში. ფაქტობრივად, პროგრამის კომპილაციის შემდეგ TypeScript-ზე იგი შეიძლება შესრულდეს ნებისმიერ თანამედროვე ბრაუზერში ან გამოყენებულ იქნას Node.js სერვერის პლატფორმაზე. კომპილატორის კოდი, რომელსაც TypeScript გადაჰყავს JavaScript- ში, ვრცელდება Apache ლიცენზიით.

TypeScript განსხვავდება JavaScript-სგან ტიპების ცხადი სტატიკური დანიშნულების შესაძლებლობით, სრულფასოვანი კლასების გამოყენებით (როგორც ტრადიციულ ობიექტ-ორიენტირებული ენებშია), ასევე მოდულების ჩართვით, რაც აჩქარებს დამუშავების პროცესს, ამარტივებს კითხვადობას, რეფაქტორინგს და კოდის მრავალჯერად გამოყენებას, აგრეთვე შეცდომების პოვნას დამუშავებისა და კომპლიაციის ეტაპებზე, და შესაძლოა დააჩქაროს პროგრამების შესრულება.

ვებ-საიტი შედგება აგრეთვე კლიენტისთვის უხილავი მხარისგან, რომელიც, საიტის ფუნქციონირებას ახდენს, მაგრამ თავად კლიენტისთვის უხილავია, მაგალითად აღნიშნულ საიტზე თუკი მოვახდენთ მენიუს პუნქტში „სწავლება“ დაჭერას, უხილავი მხარე back-end მოახდენს ჩვენი მოთხოვნისა დამუშავებას და მიღებულ შედეგს გამოგვიტანს ჩვენს ინტერნეტ ბრაუზერში.

Back-end ენებად მოიაზრება:

- PHP
- .NET (C#, VB)
- Python
- Javascript (Node JS)
- Actionscript (Flash Media Server)
- C (CGI)
- Erlang
- oh, and SQL for db queries
- და სხვ.

ჩამონათვალი არ არის მცირე, მათი გამოყენება ცოდნის და სპეციფიკიდან გამომდინარეც შეიძლება მოვახდინოთ, თუ პროგრამისტი არ არის მხოლოდ ერთ პროგრამირების ენაზე დამოკიდებული, ეს კარგია იმ მხრივ რომ თითოეულ საიტს მისი შინაარსიდან გამომდინარე, შეიძლება მისთვის შესაბამისი ენა შევურჩიოთ და მისი წარმადობა უკეთესი იყოს, მაგალითად PHP ს მაგივრად გამოვიყენოთ Node JS. თუმცა, როდესაც back-end ის დეველოპერი მრავალ ენაზე წერს, ამასაც აქვს გარკვეული მინუსი, იმ შემთხვევაში თუ ყველა ენაზე თანაბარი და არა სიღრმისეული ცოდნა აქვს, უფრო უკეთესია ის მდგომარეობა, როდესაც რომელიმე ენას, მაგალითად PHP ძირეულად ფლობს პროგრამისტი, და დანარჩენ ენებზეც აქვს შეხება და წარმოდგენა. საბოლოოდ შეგვიძლია ვთქვათ, რომ საიტი წარმოადგენს აპლიკაციას, რომელიც შედგება კლიენტისთვის ხილულ და

უხილავ მხარეებად, თუ დავსვავთ კითხვას, რომელი მათგანი უფრო პრიორიტეტულია და საჭიროა, შეგვიძლია ვთქვათ, რომ ამაზე ცალმხრივად პასუხის გაცემა არასწორია, რადგან არსებობს საიტის სფეციფიკები, რომლებზეც ვიზუალური მხარე უფრო მნიშნველოვანია, და არსებობს საიტები, რომლებზეც სერვერ-კოდის ანუ back-end დეველოპმენტი უფრო მნიშნველოვანია, თუ საქმე გვაქვს მაგალითად ისეთ საიტთან, როგორიცაა საათების სარეკლამო და ონლაინ მაღაზია, მაშინ რა თქმა უნდა მნიშნველოვანია მისი დიზაინი მომხმარებლისთვის, ხოლო თუ საქმე გვავქვს საძიებო სისტემასთან, მაგალითად Google თან, ალბათ დამეთანხმებით მისი ვიზუალური მხარე არ გამოირჩევა სილამაზით, თუმცა მისი სპეციფიკიდან გამომდინარე არც მოითხოვს რომ ვიზუალურად იყოს დატვირთული, რადგან მისი მთავარი დანიშნულება გახლავთ სწრაფად მოიძებნოს ის ინფორმაცია რომელსაც მომხმარებელი ეძებს, და მიღებული შედეგებიც მარტივად იქნას ასახული ვებ - ბრაუზერში.

## 1.2. ჰიპერ-ტექსტების მარკირების ენა - HTML

როგორც აღვნიშნეთ, front-end ნაწილის შესაქმნელი ერთ-ერთი ენაა HTML მარკირების ენა. იგი შექმნეს 1989 წელს [17]. შეგვიძლია ვთქვათ, რომ იგი წარმოადგენს საიტზე გამოსატანი ინფორმაციის მოწესრიგების საშუალებას, მისი წერის სინტაქსი, ანუ რისგანაც შედგება არის ე.წ. ტეგები.

*ტეგები* - მოწესრიგებული საშუალებაა ინფორმაციის გამოსატანად, ტეგის წერა ხდება კუთხოვანი ფრჩხილების საშუალებით, <html>, ესაა მთავარი ტეგი, რომლითაც იწყება html დოკუმენტის აღწერა და აუცილებელია მისი დახურვა ასეთი სახით </html>. ანუ შეგვიძლია ვთქვათ, რომ არსებობს წყვილი ტეგები, რომლებსაც სჭირდება გახსნა და დახურვა, ხოლო ინფორმაციის და სხვა ტეგების მოთავსება ხდება გახსნა დახურვას შორის, ასევე ისეთ ტეგები, რომლებიც არ არის წყვილი სახით

წარმოდგენილი, არ სჭირდება დახურვა, ასეთია მაგალითად <img> ტეგი.

მოვიყვანოთ მცირე მაგალითი, თუ როგორ შეიძლება გამოვიტანოთ ინფორმაცია უნივესიტეტის შესახებ, ისე რომ ტეგები სწორად გამოვიყენოთ. მაგალითად გვაქვს ინფორმაცია უნივერსიტეტის ჩვენს შესახებ გვერდის, პირველ რიგში მოვიყვანოთ html დოკუმენტის მთავარი ნაწილები, ეს გახლავთ Head ტეგი. რომელიც საიტის ის ნაწილია, რომლის ინფორმაცია გარდა title-ს, ფაქტობრივად, უხილავია კლიენტისთვის, თუ არ გახსნის სპეციალური საშუალებით მომხმარებელი კოდს და არ ნახავს.

Head-ში ხდება ფონტების, სტილების, სკრიპტების შემოტანა. ჩვენ შემთხვევაში მოვახდინოთ ორი მთავარი ტეგის შეტანა head ტეგში, ეს გახლავთ <meta charset="utf-8">, ეს არის ტეგი, რომელიც მოახდენს დოკუმენტში იმის განსაზღვრას, რომ დაინახოს ქართული სიმბოლოები და ის სიმბოლოები რომლებიც utf-8 უნიკოდს მიეკუთვნება.

შემდეგი გახლავთ <title></title>, მასში შეტანილი ინფორმაცია, ბრაუზერებში როგორებიცაა google chrome, mozilla, internet explorer, ტაბების დასახელებებში შეგიძლიათ იხილოთ.

მომდევნო მთავარი ნაწილი, რომელიც ბრაუზერში ჩანს, ესაა <body></body>. ფაქტობრივად, ეს საიტის ის ნაწილია, რომელიც მთელ მონაცემებს და საიტის ვიზუალურ მხარეს განსაზღვრავს. ჩვენ შემთხვევაში, როდესაც გვსურს, რომ გამოვიტანოთ ინფორმაცია უნივესიტეტის შესახებ, დაგვჭირდება ერთ-ერთი მთავარი ტეგი, რომელიც გახლავთ heading - ტეგი <h1></h1>. იგი არის საიტის კონკრეტული გვერდის ყველაზე პრიორიტეტული დასახელება, რომელსაც საძიებო სისტემა მიანიჭებს უპირატესობას, სხვა სიტყვებით, მასში შეგვიძლია შევიტანოთ ის მთავარი დასახელება, რომელიც განსაზღვრავს საიტის გვერდის სახელს, მაგალითად „უნივერსიტეტის ისტორია“, ხოლო შემდეგი ტეგი რომელსაც გამოვიყენებთ, ეს გახლავთ პარაგრაფი ,

მასში შეგვიძლია ტექსტური ინფორმაცია შევიტანოთ. საბოლოო სახე ჩვენი დოკუმენტის შეგიძლიათ იხილოთ:

```
<html>
  <head>
    <meta charset=”utf-8”>
    <title>საქართველოს ტექნიკური უნივერსიტეტი</title>
  </head>
  <body>
    <h1>უნივერსიტეტის ისტორია</h1>
    <p>ინფორმაცია უნივერსიტეტის დაარსების შესახებ...</p>
  </body>
</html>
```

ტეგებში არსებობს ასევე ატრიბუტები, რომლებიც კონკრეტულ დავალებებს ასრულებს, ასეთია უკვე აღწერილი meta ტეგი რომელსაც გავუწერეთ charset – რომელიც meta ტეგის ატრიბუტია და მისი გაწერით მოვახდინეთ meta ტეგის ტიპის განსაზღვრა. მასში შეტანილი utf-8 მნიშნველობით კი დოკუმენტის უნიკოდის განისაზღვრა.

არსებობს ასევე სხვა არაერთი ტეგი, რომლებიც კონკრეტულ შემთხვევებში გამოიყენება, როგორიცაა: ლისტები (ul, ol), <br> - რომლის შემდეგ შეტანილი ინფორმაციი ახალ ხაზზე გამოტანა მოხდება, <a href=”http://gtu.ge”></a> ეს გახლავთ ტეგი, რომელიც გადამისამართებთ იმ ბმულზე რომელიც href ატრიბუტშია გაწერილი.

ჩვენს მიერ მოყვანილი html დოკუმენტის მაგალითი პრიმიტიულია, რადგან ხშირ შემთხვევაში, ჩვენ გვჭირდება html - ტეგების გარკვეული სახით დახარისხება. მაგალითად, როდესაც მივდივართ სუპერმარკეტში და ვყიდულობთ ხილს, რძის ნაწარმს, კოსმეტიკას, ჰიგიენის საგნებს, ტკბილეულს, რა თქმა უნდა, ყველაფერი ამის ერთად მოთავსება შეგვიძლია ერთ ჩანთაში ან რაიმე კონტეინერში, თუმცა ამას გარკვეული პრობლემები

შეიძლება მოყვეს, მაგალითად, რაიმე რბილ ხილს ან რძის ნაწარმს შეიძლება ზედა მხრიდან მოხვდეს ისეთი ნივთები, რომლებიც მათ დაზიანებას გამოიწვევს. ამისათვის შეიძლება მოვახდინოთ გარკვეული ნაწარმის დაფასოება სხვადასხვა ნიშნით, სხვადასხვა კონტეინერში, ხოლო შემდეგ ერთად მოთავსება ერთ დიდ კონტეინერში. ასევე შეგვიძლია ვთქვათ, რომ html ტეგები, სპეციფიკიდან გამომდინარე, მოითხოვს გარკვეულ კონტეინერებს, რის მიხედვითაც მოხდება ინფორმაციის დაფასოება და მოთავსება კონტეინერებში. ამისათვის გამოიყენება კონტეინერი .

ვებ-გვერდები შედგება სამი ნაწილისგან, ესენი გახლავთ

- header
- content
- footer

ყველა მათგანს თავისი დანიშნულება აქვს.

მაგალითად header ნაწილი ძირითადად შედგება საიტის ლოგოსგან, მენიუსგან, ენების გადამრთველისგან.

Content ნაწილი შეიცავს ძირითად ნაწილს, სადაც შესაძლებელია იყოს მოთავსებული სლაიდერი, სურათები, სიახლეები, ვიდეოები, ტექსტები, დამატებითი მენიუები, ფილტრები.

Footer ნაწილი კი ძირითადად შედგება ლოგოსგან, მენიუსა და copyright ტექსტისგან.

აღნიშნული ნაწილების განსაზღვრისთვის გამოყენება div კონტეინერი, თუმცა ყველა დივი ერთი დასახელებისაა და როგორ გამოვყოთ, რომელია header, content, footer ნაწილები? ამისათვის არსებობს ატრიბუტები, რომლებსაც ეწოდება id, class. განსხვავება მათ შორის გახლავთ ის, რომ id არის დასახელება ელემენტის, რომლის გამოყენება კონკრეტულ ერთ გვერდზე შეგვიძლია მხოლოდ ერთ ელემენტზე, ხოლო class გახლავთ ატრიბუტი, რომლის დარქმევა შეგვიძლია იმდენ ელემენტზე რამდენზეც დაგვჭირდება, სურვილისამებრ.

მათზე სტილების მინიჭება ხდება CSS ენით, რომელზეც მომდევნო პარაგრაფში ვისაუბრებთ, ჩვენ შემთხვევაში შეგვიძლია გამოვიყენოთ id ატრიბუტები, საბოლოოდ აღვწეროთ გვერდი, რომელიც ეკუთვნის უნივერისტეტს, აქვს მენიუ header - ში, აქვს უკვე მოყვანილი მაგალითი, ტექსტი უნივერსიტეტის შესახებ, რომელიც content კონტეინერში მოთავსდება, და copyright, რომელიც footer ში მოთავსდება.

მივიღებთ შემდეგ კოდს:

```
<body>
  <div id=”header”>
    <ul>
      <li><a href=”home.html”>მთავარი</a></li>
      <li><a href=”about.html”>უნივერსიტეტის შესახებ</a></li>
      <li><a href=”contact.html”>კონტაქტი</a></li>
    </ul>
  </div>
  <div id=”content”>
    <h1>უნივერსიტეტის ისტორია</h1>
    <p>ინფორმაცია უნივერსიტეტის დაარსების შესახებ...</p>
  </div>
  <div id=”footer”>
    <p>copytight © 2019</p>
  </div>
</body>
```

Html - ენასაც თავისი ვერსიები აქვს, მისი პირველი ვერსია როგორც აღვნიშნეთ 1989 წელს გამოვიდა, რომელიც ვითარდებოდა. დღეისთვის არსებობს HTML 5.1 ვერსია, რომელიც 2016 წელს შეიქმნა. მასში ბევრი ახალი ტეგია დამატებული.

ერთ-ერთი პრობლემა, რაც HTML_5 ვერსიამდე არსებობდა, იყო კონტეინერის საკითხი, რომელიც უკვე ვახსენეთ. რამდენჯერაც რაიმე ახალი კონტეინერის შექმნა დაგვჭირდებოდა

უნდა გამოგვეყენებინა  ტეგი, აღნიშნული პრობლემა გადაიჭრა HTML_5 ვერსიაში, მაგალითად ზემოთ მოყვანილი ვარიანტი კონტეინერების შეგვიძლო html-ის ახალ ვერსიაში ამდაგვარად ჩაგვეწერა:

```
<header id=”header”>
</header>
<main id=”content”>
 </main>
 <footer id=”footer”>
 </footer>
```

როგორც მაგალითიდან ჩანს, უკვე არსებობს თანამედროვე html ვერსიაში header, footer ტეგები, რომლებიც შინაარსობრივად უკვე წარმოაჩენს კიდეც რასთან გვაქვს საქმე, ხოლო main ტეგი გახლავთ აგრეთვე ახალი ტეგი, რომელშიც უნდა გაიწეროს საიტის ის ნაწილი, რომლის ცვლილებაც ხდება სხვა და სხვა გვერდზე შესვლისას, მაგალითად ჩვენ შემთხვევაში შეიძლება მთავარ გვერდზე, უნივერსიტეტის შესახებ და კონტაქტის გვერდზე header, footer, ნაწილები უცვლელი იყოს, ხოლო მთავარ გვერდზე main ნაწილში შეიძლება იყოს მოთავსებული სლაიდერი, შემდეგ სტატიები, უნივერსიტეტის შესახებ გვერდზე ინფორმაცია უნივერსიტეტის შესახებ, ხოლო საკონტაქტო გვერდზე, უნივერსიტეტთან დასაკავშირებელი ინფორმაცია, ამიტომ ყველა ეს ცვლადი ნაწილი შეგვიძლია main ტეგში მოვაქციოთ. რა თქმა უნდა ბევრი სხვა ტეგებია html ენაში, რომლებზეც არ გაგვიმახვილებია ყურადღება, მათი ნახვა და გამოყენების წესები შეიძლება [18].

ბოლოს შეიძლება ვთქვათ, რომ საიტის გვერდების სტრუქტურის შემუშავება ხდება HTML მარკირების ენით, მაგრამ ის ვიზუალურად უსარგებლო იქნება, თუკი არ მოვახდენთ სტილიზებას, რაც CSS სტილების მეშვეობით ხოეციელდება. ეს საკითხი მომდევნო პარაგრაფში გვაქვს აღწერილი.

## 1.3. სტილები CSS და SCSS

CSS იშიფრება Cascading Style Sheet, მისი შექმნა მოხდა 1996 წელს, და გამოიყენება მარკირების ენის როგორიცაა html სტილების განსაზღვრისთვის {19}. გამოყენების პრინციპები საკმაოდ მარტივია, შესაძლებელია html ტეგებზე მიმართვა რამდენიმე სახით, მათგან ძირითადებია:

- ტეგის დასახელების მიხედვით (ამ შეთხვევაში ყველა ტეგზე მოხდება სტილების გაწერა);
- Id ატრიბუტის მიხედვით (მხოლოდ ერთი ელემენტის სტილიზებაა შესაძლებებლი);
- Class ატრიბუტის საშუალებით (ყველა იმ ელემენტს მიმართავს, რომელსაც აქვს ერთდაიმავე დასახელების კლასი).

არსებობს სხვა მიმართვებიც, როგორიცაა ნებისმიერი სხვა ატრიბუტის მიხედვით გარდა Id, class - ისა. ასევე რომელიმე ელემენტის შემდეგ ელემენტს შეგვიძლია მივმართოთ „+" სიმბოლოთი, ასევე შეგვიძლია მივმართოთ ელემენტებს, რომლებსაც არ აქვს კლასი. მაგალითად, გვსურს ყველა ელემენტის სტილიზება ერთიდაიმავე ფერად, გარდა იმ ელემენტისა, რომელსაც აქვს კონკრეტული კლასი.

სინტაქსი, თუ როგორ ხდება ელემენტებზე მიმართვა, შემდეგია:

#elementIdName {} – id ის შემთხვევაში;

.elementClassName {} – კლასის შემთხვევაში.

„#" სიმბოლოს შემდეგ იწერება ის id დასახელება, რომელიც html ენაში გამოყენებულია კონკრეტულ ტეგზე, ხოლო „ . " სიმბოლო გამოიყენება class სახელის ტეგების მოსანიშნად, ხოლო სტილების გაწერა კი ხდება ფიგურული ფრჩხილების შიგნით.

სტილები მრავალი არსებობს:

- ფერები
- ზომები

- დაშორებები
- საზღვრები
- ...

მაგალთად, მოვიყვანოთ მხოლოდ რამდენიმე სტილი, ვთქვათ გვაქვს ტეგი <header id="header"> რომელსაც გვსურს ფონის ფერი მიენიჭორ მწვანე, ხოლო სიმაღლეში 50px, აღნიშნული სტილების გაწერა header ტეგისთვის ამდაგვარად მოხდება:

```
#header {
  background-color: green;
  height: 50px;
}
```

CSS - ის ბოლო ვერსიაა CSS3 [20].

ერთ-ერთი სიახლე, რასაც დღევანდელი რეალობა გვთავაზობს, ესაა CSS ენის preprocessor-ები, როგორებიცაა less, sass ტექნოლოგიები. მათი საშუალებით შესაძლებელია უფრო მოქნილად, დინამიკურად და მარტივად მოვახდინოთ სტილების დაწერა.

sass - რომლის ფაილების გაფართოებაა .scss, გვაძლევს საშუალებას მოვახდინოთ:

- ცვლადების განსაზღვრა;
- ფუნქციების შექმნა;
- ასევე კლასების თუ სხვა მიმმართველი საშუალებების ერთმანეთში ჩადგმა.

ცვლადების განსაზღვრის უპირატესობებია ის, რომ მათი საშუალებით ხდება გარკვეული, მაგალთად, საიტზე გამოყენებად ფერის ცვლადის სახით შენახვა. მაგალითად, ვიყენებთ საიტზე ორ ძირითად ფერს, მთავარი ფერი გვაქვს ნაცრისფერი, ხოლო მეორე წითელი, შეგვიძლია ასეთნაირად მოვახდინოთ მათი განსაზღვრა:

```
$base-color: gray;
$second-base-color: red;
```

ასევე CSS - ში შესაძლებელია მოვახდინოთ მიმართვები ჩადგმული კლასების საშუალებით, მაგალითად, გვაქვს header, რომლის id გახლავთ header, და მასში მოთავსებული menu კლასის მქონე ul ტეგი.

ასევე გვაქვს menu კლასის მქონე ტეგი, რომელიც footer- შია მოთავსებული. თუ გვსურს მიმართვა მოვახდინოთ მხოლოდ header id-ის მქონე კონტეინერში მოთავსებულ menu კლასის ტეგზე და გვსურს, რომ მივანიჭოთ ფონის ფერი წითელი, ასევე გვსურს შავი ფერის ჩრდილის დანიშვნა, ასეთნაირად მოხდება მისი ჩაწერა:

```
#header .menu {
  background-color: red;
  -webkit-box-shadow: 0 0 5px black;
  box-shadow: 0 0 5px black;  }
```

როგორც ჩანს მაგალითიდან ჩრდილის სტილი box-shadow, ორჯერ გამეორდა, რადგან საჭიროა მისთვის -webkit- ბრძანების დაწერა წინ ერთ შემთხვევაში, იმის გამო რომ ყველა ბრაუზერი ვერ ხედავს მხოლოდ box-shadow ბრძანებას და დამატებით -webkit- სჭირდება, ხოლო sass ის დახმარებით აღნიშნული კოდის დაწერა შემდეგნაირად შეიძლება:

*ცვლადებისა და ფუნქციის განსაზღვრა:*

```
$base-color: gray;
$second-base-color: red;
@mixin box-shadow() {
  -webkit-box-shadow: 0 0 5px black;
  box-shadow: 0 0 5px black;
}
```

სტილების გაწერა კი ამდგვარად მოხდება

```
#header {
        .menu {
          background-color: $second-base-color;
```

```
        @include box-shadow();
    }
}
```

როგორც ხედავთ menu კლასი პირდაპირ header id-ის შიგნითაა მოთავსებული, ასევე ფონის ფერს ცვლადით ვანიჭებთ, ხოლო ორხაზიანი ჩრდილის ბრძანების მაგივრად გვაქვს ერთხაზიანი ფუნქცია შემოტანილი, რაც იგივე შედეგს მოგვცემს, რასაც პირველ რიგში მხოლოდ css-ში წერისას ვახდენდით. უპირატესობა ისაა, რომ თუ კი ფერის შეცვლა მოგვინდება საიტზე, შევცვლით მხოლოდ ცვლადში, ასევე სადაც ჩრდილის გამოყენება მოგვინდება, უბრალოდ ფუნქციას გამოვიძახებთ, საბოლოოდ კი არსებობს საშუალებები, რომლითაც ხდება sass-ში დაწერილი კოდის css კოდში გადაყვანა, მივიღებთ იგივე შედეგს, რასაც css-ში ჩვენითაც დავწერდით, მაგრამ sass-ის უპირატესობაა დეველოპერებისთვის საქმის გაადვილება, ის არ ახდენს კლიენტებისთვის რაიმე მნიშვნელოვან ცვლილებას, მაგრამ ბევრად მარტივია preprocessor-ით მუშაობა, ვიდრე css-ის სტილების ხელით გაწერა.

ყველაზე თანამედროვე css საშუალება ელემენტების განლაგებისთვის გახლავთ flexbox, რომელიც ბევრად ამარტივებს სტილების გაწერას [21].

## 1.4. ფრეიმვორკი Bootstrap

დღეისათვის არსებობს არაერთი უფასო და მარტივი საშუალება, რომლებიც გვეხმარება თანამედროვე და სხვადასხვა მოწყობილობებზე მორგებული დიზაინების შექმნაში, ერთ-ერთი მათგანი გახლავთ bootstrap [22]. მისი დოკუმენტაციის ნახვა და ჩამოტვირთვა შეიძლება ბმულიდან [23].

Bootstrap არის HTML, CSS, JS framework, რაც გულისხმობს, რომ ის მზა ბიბლიოთეკაა, რომელიც შედგება უკვე არსებული კლასების, კომპონენტების და სტილიზებული ფანჯრების

ფორმებისგან, რომელთა გამოყენებისთვის მხოლოდ და მხოლოდ მათი ნახვაა საჭირო. მაგალითად, bootstrap გვთავაზობს მზა ღილაკების სტილებს, რომელთა გამოსაყენებლად მხოლოდ კლასის ნახვაა საჭირო და შემდეგ ღილაკის ტეგში მისი გაწერა (ნახ.3).


**ნახ. 3. bootstrap ბიბილიოთეკის ფრაგმენტი**

ყველაზე არსებითი, რასაც bootstrap გვთავაზობს, ესაა ბადეები. მას აქვს 12 ბადიანი სისტემა, რომლის დაყოფაც შესაძლებელია 1, 2, 3 ,4 … 12 სვეტად და მათი სურვილისამებრ გამოყენება.

შეიძლება ვიფიქროთ, რომ შეგვიძლია ჩვენ თვითონაც შევქმნათ bootstrap - ის მსგავსი framework და ჩვენვე გამოვიყენოთ. ეს ნამდვილად ასეა და სურვილის შემთხვევაში მისი გაკეთებაც შესაძლებელია, თუმცა bootstrap გახლავთ დახვეწილი განახლებადი, მისი ბოლო ვერსია 3.3.7-ია. ასევე გამოდის ვერსია 4, მასში ძალზე ბევრი სხვა შესაძლებლობა და სტილია უკვე გამზადებული, რომლებსაც ამჯერად არ შევეხებით

## 1.5. ვებ-დეველოპმენტის BEM მეთოდოლოგია

Front-end დეველოპერების ერთ-ერთი მტკივნეული საკითხია ტეგებზე და მათ ელემენტებზე დასახელებების შერჩევა, ასევე პრობლემატურია ის, რომ ყველა დეველოპერი მუშაობის განმავლობაში იძენს საკუთარი წერის სტილს, რომელსაც ხვეწავს და რომელიც კარგია, თუმცა პრობლემა იბადება მაშინ, როდესაც ერთი დეველოპერის მიერ გაკეთებული პროექტის გადაკეთება ან მის მიერ გაწერილ სტილებში ჩარევა უწევს სხვა დეველოპერს, ამ პრობლემის გადაწყვეტისთვის შემუშავდა მეთოდოლოგია BEM [8,57].

BEM (Block Element Modifier) გვთავაზობს ელემენტებზე სახელების დარქმევის წესებს. მაგალითად, როგორც 1.2 პარაგრაფში html-ის შესახებ ვნახეთ, სადაც გვქონდა header - ტეგი, ამ შემთხვევაში შეგვიძლია ვთქვათ, რომ header არის ბლოკი, რომლის შიგნით მოთავსებული ტეგები გახლავთ ელემენტები:

```
<header class="header">
  <img class="header__logo" src="logo.svg">
</header>
```

ჩვენ შემთხვევაში BEM გვთავაზობს ამგვარი სახელების დარქმევის წესებს, მთავარ ბლოკს ვარქმევთ სასურველ სახელს, ხოლო მის შვილობილ ელემენტს ვარქმევთ ბლოკის და ორი ქვედა__ხაზით გამოყოფილ ელემენტის დასახელებას, რაც ამარტივებს გარჩევას იმისას თუ რას ეკუთვნის კონკრეტული ელემენტი, ხოლო იმ შემთხვევაში თუ გვსურს იგივე ელემენტის გადასტილვა, მაგალითად, გვაქვს გაწერილი სტილი:

```
.btn__save {
    background-color: red;
    color: white;
    padding: 10px 6px;
    font-size: 1.2em;   }
```

და გვსურს რომ იგივე სტილების, მაგრამ სხვა ფერის იყოს სხვა ღილაკის სტილი, ამისათვის ვიყენებთ შემდეგ საშუალებას:

```
.btn__save—modifier {
   background-color: green;
}
```

გაწერილი სტილის მოდიფიცირება ხდება ორი დეფისით და მოდიფიცირების სახელის დარქმევით.

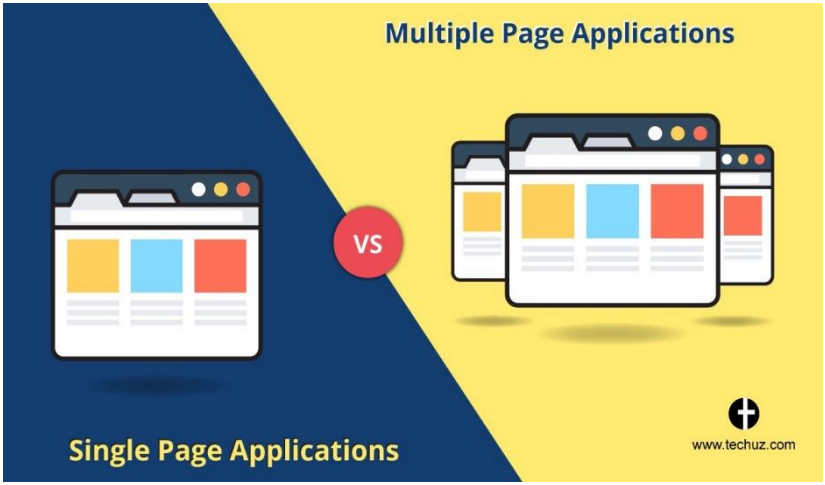
ასევე BEM წესებში შედის, რომ არ უნდა გამოვიყენოთ ID მიმმართველები, მიზეზი გახლავთ ის, რომ გაწერილი სტილები გათვლილია მრავალჯერად გამოყენებაზე, რაც css კოდს დინამიკურს ხდის.

ერთ-ერთი დიდი პლიუსი BEM მეთოდოლოგიის ის გახლავთ, რომ მისი მცოდნე დეველოპერები ადვილად გაარკვევენ ერთმანეთის მიერ დაწერილ კოდს და მისი ცვლილებაც მარტივი გახდება, იმ შემთხვევაშიც თუ სხვის მიერ დაწერილ კოდთან გვაქვს საქმე [8,24].

მომდევნო თავში განვიხილავთ ვებ-დეველოპმენტის თანამედროვე ფუნდამენტურ ტექნოლოგიებს, რომლებიც აქტიურად გამოიყენება დღეს ამ სფეროში.

## 1.6. მულტიგვერდებიანი და ერთგვერდიანი აპლიკაციები

ვებ-აპლიკაციები ძირითადად იყოფა ორ სახეობად [35] (ნახ.4): მულტიგვერდებიანი აპლიკაცია და ერთგვერდიანი აპლიკაცია (SPA) .




ნახ.4. მულტი- და ერთ-გვერდიანი აპლიკაცია

- *მულტიგვერდებიანი აპლიკაცია* არის ტრადიციული ვებ-აპლიკაცია, რომელიც ახდენს მთლიანი გვერდის გადატვირთვას და ახლის ჩამოტვირთვას, როდესაც მომხმარებელი ითხოვს ახალ გვერდს.

ყველა ჯერზე, როდესაც კლიენტი ახდენს ახალი გვერდის მოთხოვნას, ხდება ახალი გვერდის მოთხოვნის გაგზავნა სერვერთან და მისი გამოგზავნილი პასუხის ასახვა ვებ-ბრაუზერში. თუმცა, შესაძლებელია გვერდის გარკვეულ კომპონენტებზე მოქმედების შესრულება Ajax-ით [30].

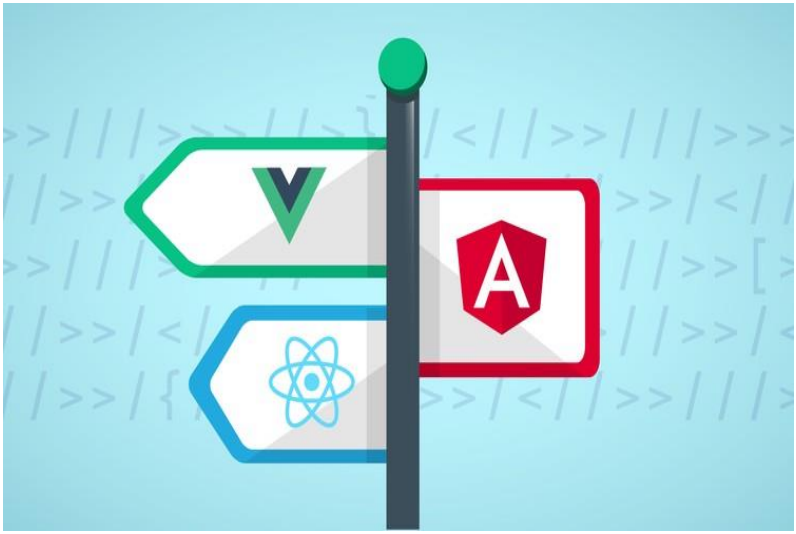
- *ერთგვერდიანი აპლიკაცია*, როგორც სახელიდან ჩანს, შედგება მხოლოდ ერთი გვერდისგან. ის ახდენს საიტის მთელი

ნაწილის მხოლოდ ერთ გვერდში ჩამოტვირთვას და არა ნავიგაციას სხვადასვა გვერდებზე.

ზოგადად, უმეტეს საიტზე არის მრავალჯერ გამეორებადი ნაწილები, როგორიცაა (headers, footers, logos, navigation). როდესაც მომხმარებელი მოითხოვს რომელიმე გვერდზე გადასვლას, მნიშვნელობა არ აქვს რომელზე, ხდება სერვერთან მიმართვა და მთლიანი content-ის ხელახლა ჩამოტვირთვა (headers, footers, logos, navigation), რაც ბრაუზერის გადატვირთვას იწვევს და შესაბამისად არც თუ ისე მოსახერხებელია.

SPA - არის სწრაფი, ვიდრე ტრადიციული ვებ-აპლიკაციები, რადგან ისინი ახდენს საიტის ლოგიკის გაშვებას ვებ-ბრაუზერში და არა სერვერზე. კლიენტის მიერ SPA გვერდის ტიპის აპლიკაციასთან წვდომისთანავე ხდება, ფაქტობრივად, სრული გვერდის ჩამოტვირთვა ბრაუზერში, და თუ კლიენტი მოითხოვს რომელიმე კონკრეტულ გვერდს, მოხდება იმ კონკრეტული გვერდის ნაწილის ჩამოტვირთვა და არა მთლიანი გვერდის. მაგალითად თუ კლიენტი გადავა საკონტაქტო გვერდზე, შეიცვლება მხოლოდ საკონტაქტო გვერდის ნაწილი.

Single Page Application ენები ზოგადი ცნებაა, თუმცა როგორც სერვერული ენები არსებობს არაერთი, რომელთა მეშვეობით შესაძლებელია ვებ-საიტის სერვერული მხარის გამართვა, ასევე front-end დეველოპმენტში. SPA ტიპის საიტის შესაქმნელად არსებობს არაერთი ენა, მაგალითად, Vue, React, Angular (ნახ.1.5). აღნიშნულ ენებში არსებობს სინტაქსური განსხვავებები, როგორც back-end ენებში, როგორიცაა PHP, ASP.NET.



ნახ.1.5. SPA ტიპის ენები: Vue, React, Angular

Vue.js არის JavaScript ფრეიმვორკი ღია საწყისი კოდით, გამოიყენება მომხმარებელთა ინტერფეისების ასაგებად [50,51]. იგი ადვილად ინტეგრირდება პროექტებში JavaScript-ის სხვა ბიბლიოთეკებთან (მაგალითად, ReactJS). Vue.js შეუძლია ფუნქციონირება ვებ-ფრეიმვორკის სახით ერთგვერდიანი აპლიკაციების შესაქმნელად MVVM (Model-View-VievModel) შაბლონით, რეაქტიული პროგრამირების სტილში.

MVVM-შაბლონის საფუძველზე Vue.js ფრეიმვორკი იძლევა მონაცემებთან უშუალო დაკავშირების (binding) შესაძლებლობას, რაც უზრუნველყოფს შეტანა-გამოტანის პირდაპირ კავშირს მონაცემთა წყაროსთან. ამის გამო აღარაა საჭირო მონაცემთა ხელოვნურად განსაზღვრა (მაგალითად, jQuery-ს გამოყენებით) HTML DOM -იდან.

# II თავი:
# ვებ-აპლიკაციების დეველოპმენტის თანამედროვე ტექნოლოგიები

## 2.1. JavaSript ენა ვებ-დეველოპმენტის ფუნდამენტური ტექნოლოგია

პირველ თავში განხილული ვებ-აპლიკაციების აგების ინსტრუმენტული საშუალებებისაგან განსხვავებით, JavaScript ენა არაა მხოლოდ front-end დეველოპმენტში გამოყენებადი ენა, ის არის ასევე Server-Side ენა (Node JS) [4,25]. მისი გამოყენება ინტერფეისებში, front მხარესაცაა შესაძლებელი [27]. მისი პირველი ვერსია 1995 წელს გამოჩნდა.

Front-end დეველოპმენტში, JS - ის გამოყენება შესაძლებელია ისეთი რთული შემთხვევების დროს, როდესაც საქმე გვაქვს ისეთ საკითხებთან, რომლებსაც CSS ით ვერ გავართმევთ თავს, მაგალითად სლაიდერის შექმნა, რაიმე ღილაკზე დაჭერისას ფანჯრის გამოტანა, ან ახალი ტეგის ჩამატება წაშლა, მსგავსი მოქმედებებისთვის გამოყენება javascript.

განსაკუთრებით საყურადღებოა JavaScript-ის ახალი ბიბლიოთეკა, რომელიც React.js ან ReactJS სახელით დამკვიდრდა და იგი მომხმარებელთა ინტერფეისების დასამუშავებლად გამოიყენება [26] (ნახ.2.1).


**ნახ.2.1. ReactJS ლოგო**

React შეიძლება გამოყენებულ იქნას ერთგვერდიანი და მობილური აპლიკაციების ასაგებად. მისი მიზანია უზრუნველყოს მაღალი სიჩქარე, სიმარტივე და მასშტაბირება.

## 2.2. JavaScript-ის შვილობილი ტექნოლოგიები

JavaScript - ენის განვითარებით შემუშავებულ იქნა ახალი გაფართოებული ტექნოლოგიები, რომლებიც მეტ მოქნილ საშუალებებს სთავაზობს დეველოპერებს ვებ-აპლიკაციების ასაგებად. JavaScript-ოჯახის ასეთი ტექნოლოგიებია jQuery, Ajax, AngularJS, რომელთაც ქვემოთ დეტალურად შევეხებით.

### 2.2.1. jQuery ტექნოლოგია

Native JavaScript - ენის გამოყენება შედარებით უფრო მეტ შრომას მოითხოვს ვებ-აპლიკაციების შექმნისას, არსებობს უფრო გამარტივებული საშუალება, რომელიც გახლავთ jQuery [28]. იგი შექმნილია JavaScript ენაზე, და გვაძლევს უფრო ადვილად მუშაობის საშუალებას, მასში უფრო მარტივადაა შესაძლებელი იმ ამოცანების შესრულება, რაც მხოლოდ JavaScript ის გამოყენებისას უფრო მეტ შრომას მოითხოვს. jQuery ბიბლიოთეკა (ან ფრეიმვორკი) უერთდება JavaScript ენას.

jQuery-ის შექმნის ძირითადი მიზანი იყო JavaScript ენაზე დაწერილი სცენარების იმ ფრაგმენტთა კოდირების გამარტივება, რომლებიც მრავალჯერად გამოყენებას პოულობს ინტერნეტში განთავსებისათვის განკუთვნილ ფაილებში [28]. მაინც, რას წარმოადგენს ამ ბიბლიოთეკის კომპონენტები და რა შესაძლებლობებს გვთავაზობს ისინი ? ესაა JavaScript-პლაგინები და Ajax-დამატებები, რომლებიც უზრუნველყოფს მოვლენების დამუშავებას, ვიზუალურ ეფექტებს, აგრეთვე DOM-იერარქიულ სტრუქტურაში მოძრაობას XPath-ის იდეოლოგიაზე დაყრდნობით და სხვ.

ორიგინალურია jQuery-ბიბლიოთეკის ორგანიზების კონცეფცია. იგი წარმოგვიდგება კომპაქტური უნივერსალური ბირთვისა და პლაგინების ერთობლიობად, რის შედეგადაც საჭირო აღარაა მასში შემავალი ფუნქციების ყველა შესაძლო შემთხვევაზე გათვლა-ორიენტირება (რაც ძალზე ზრდის (მეტწილად

გამოუყენებადი) კოდის მოცულობას. შედეგად, შესაძლებელი ხდება რესურსისათვის სწორედ JavaScript-ის იმ ფუნქციების მომარაგება, რომელიც, სავარაუდოდ, სავსებით საკმარისი იქნება კლიენტის (ამ შემთხვევაში დამპროექტებლის) წინაშე მდგარი ამოცანების გადასაწყვეტად.

jQuery ბიბლიოთეკის მიერთება შეიძლება ასე:

```
<head>
    <script type="text/javascript" src="js/jquery.js" >
</head>
```

მაშასადამე, jQuery ბიბლიოთეკის შემცველ ფაილს განვათავსებთ ჩვენი HTML -ფაილის მეზობლად შექმნილ js სახელწოდების მქონე დირექტორიაში.

### 2.2.2. Ajax ტექნოლოგია

Ajax აბრევიატურა არის კომბინირებული როგორც Asynchronous JavaScript And XML. იგი ინტერაქტიული ვებ აპლიკაციების შექმნის ტექნოლოგიაა. ეყრდნობა HTML/XHTML, XML/JSON და JavaScript ენებს [29.30].

Ajax-ის დახმარებით JavaScript-ს შეუძლია დაუკავშირდეს სერვერს XMLHttpRequest ობიექტის გამოყენებით. ამ ობიექტის დახმარებით, JavaScript ანახლებს ეკრანზე მასალის ვებ სერვერთან კომუნიკაციის შესაბამისად, გვერდის ხელახლა ჩატვირთვის გარეშე. აბრევიატურაში XML-ის გამოყენების მიუხედავად, პრაქტიკაში, თანამედროვე ვებ აპლიკაციებში XML-ის ნაცვლად JSON გამოიყენება მისი უპირატესობის გამო, რაც JavaScript-თან უკეთეს ინტეგრაციას გულისხმობს.

Ajax იყენებს ასინქრონიზებულ მონაცემთა გადაცემას ბრაუზერსა და ვებ სერვერს შორის და საშუალებას აძლევს ვებ გვერდს სერვერიდან მოითხოვოს ინფორმაციის მხოლოდ ის ბიტები, რომლის მიმოხილვაც სურს მომხმარებელს.

ამგვარად, Ajax ტექნოლოგია გვაკავშირებს სერვერთან, ვაგზავნით გარკვეულ მოთხოვნას, ვიღებთ პასუხს, რომელიც ბრაუზერშია ასახული ისე, რომ ბრაუზერის გადარესტარტება არ მოვახდინოთ [29].

### 2.2.3. Angular-ის ტექნოლოგია

Angular-ის პირველი ვერსია გამოვიდა 2010 წლის 20 ოქტომბერს {31}. დღეისათვის არსებობს ორი განსხვავებული ვერესია: AngularJS და Angular_2 (ბოლო ვეერსია - Angular_7).

AngularJS ტექნოლოგია არის თანამედროვე front-end დეველოპმენტის framework-ი რომელიც შექმნილია google-ს მიერ JavaScript ენის საშუალებით [31,32].

Stack Overflow დეველოპერების კვლევით 2018 წელს Angular ერთ-ერთი ყველაზე პოპულარული პროფესიონალი დეველოპერების მიერ გამოყენებული frameworks/ბიბლითეკებია [33]. Angular არის TypeScript-ზე დაფუძნებული ენა, რომლის მფლობელია Google [34,58].

TypeScript არის „გაუმჯობესებული“ JavaScript, Microsoft-ის მიერ შექმნილი ენა, მასში შესაძლებელია ტიპების განსაზღვრა, კლასების შექმნა, თუმცა TypeScript გამოიყენება აპლიკაციის დეველოპმენტის პროცესში, მისი დასრულებული ფორმა კომპილირებული JavaScript კოდია, რომლის წაკითხვა შესაძლებელია კლიენტის მხარეს ბრაუზერის მეშვეობით.

Angular-ის პირველი ვერსია AngularJS, ახალი ვერსიისგან განსხვავებით, JavaScript-ზეა დაფუძნებული [31]. მისი სტრუქტურა შედგება html გვერდისგან პირველ რიგში, რომელიც შეიცავს დამატებით html ატრიბუტებს. AngularJS სწორედ ამ დამატებით ატრიბუტებს იყენებს იმისათვის, რომ მოახდინოს გვერდზე მანიპულირება JavaScript ცვლადების მეშვეობით, რომელთა მნიშვნელობა შესაძლოა მინიჭებული იყოს ხელით ან დინამიკურად JSON ფორმატით.

როგორც აღვნიშნეთ, Angular ენაში ძველ თუ ახალ ვერსიაში, საბოლოოდ JavaScript ენა ახდენს მანიპულირებას კლიენტის მხარეს, ბრაუზერში.

JavaScript-ია გამოყენებული იმისათვის რომ მოხდეს ბიზნეს-ლოგიკის შესრულება და სერვერთან კომუნიკაცია.

მისი საბოლოო ვერსიები საკმაოდ დახვეწილ სტრუქტურას გვთავაზობს, რომელიც არა მხოლოდ TypeScript ენის არსებობითაა გამორჩეული არამედ ვებ-გვერდის Markup საკითხიც კი (რომელიც გულისხმობს html, css, javascript ენების მეშვეობით გვერდის აწყობას) უფრო მოწესრიგებულადაა შემოთავაზებული.

### 2.2.3.1. AngularJS ტექნოლოგია

AngularJS არის კლიენტის მხარის (front-end) დაპროგრამების ტექნოლოგია. იგი გამდიდრებულია სერვერული მხარის ტექნო-ლოგიებიდან აღებული საუკეთესო ფუნქციონალით და გამოიყენება ვებ-ბროუზერში ჩატვირთული HTML-დოკუმენტების სრულყოფისათვის. ამგვარად, AngularJS ტექნოლოგია აიოლებს მდიდარი სამომხმარებლო ინტერფეისის მქონე, ინტერაქტიული ვებ-აპლიკაციების შემუშავების პროცესს. AngularJS ვებ-აპლიკა-ციები ეფუძნება Model-View-Controller (MVC) არქიტექტურულ სტანდარტს, რაც საშუალებას იძლევა შევიმუშაოთ აპლიკაციები, რომლებიც არის [36]:

- *განვრცობადი (Extendible):* პროგრამისტისთვის იოლია გავერკვეს კომპლექსური AngularJS ვებ-აპლიკაციის ლოგიკასა და სტრუქტურაში, დაამატოს ახალი ფუნქციონალი;
- *მხარდაჭერადი (Maintenance):* AngularJS აპლიკაციები იოლად ექვემდებარება Debug პროცესს, შემოწმებასა და შემდგომ მხარდაჭერას;
- *ტესტირებადი (Testable):* AngularJS აპლიკაციები მორგებუ-ლია ტესტირების პროცესზე: Unit და End-To-End ტესტების შემუშავება გაიოლებულია. ეს კი იმას ნიშნავს, რომ ავტომატურ

რეჟიმში შეგვიძლია გავუშვათ ტესტები, ვიპოვოთ და აღმოვ-ფხვრათ ხარვეზები, მანამ, სანამ საბოლოო მომხმარებელი გადააწყდება მათ;

- *სტანდარტიზებული (Standardized):* AngularJS აგებულია თანამედროვე ვებ-სტანდარტების გათვალისწინებით და სარგებ-ლობს უახლესი ტექნოლოგიების ფუნქციონალით (როგორიცაა HTML5 API), პოპულარული ხელსაწყოებითა და არქიტექტურით.

როგორც აღვნიშნეთ, AngularJS აპლიკაციები იმპლემენტირე-ბულია MVC არქიტექტურული სტანდარტის გათვალისწინებით და შედგება 3 ძირითადი ლოგიკური დონისგან: მოდელი, სამომხმარებლო ინტერფეისი (View) და კონტროლერი. ეს კომპონენტები არის გლობალური, ზოგადი საშენი ბლოკები, რაც მოიცავს AngularJS ტექნოლოგიის სპეციფიური საშენი ბლოკების გამოყენებას, როგორიცაა: მოდულები, დირექტივები, ფილტრები, factory ფუნქციები და სერვისები. დაპროგრამების პროცესში ვიყენებთ ამ მცირე კომპონენტებს და თანდათანობით გამოვკვეთავთ უფრო მაშტაბურ MVC არქიტეტურას.

AngularJS აპლიკაციები ეშვება ნებისმიერ თანამედროვე ვებ-ბრაუზერში. მაგალითად, ჩვენ აქ გამოვიყენებთ Google Chrome ბრაუზერს, რადგან იგი თავსებადია W3C სტანდარტებთან და გააჩნია F12 დაპროგრამების ხელსაწყოთა ნაკრები. გადამწყვეტი მიზეზი კი იმაში მდგომარეობს, რომ Google-მ დაამატა სპეცია-ლური Chrome Extension-ი, რომელიც F12 ხელსაწყოთა ნაკრებში ამატებს AngularJS ტექნოლოგიის სექციასაც.

AngularJS ტექნოლოგიაზე დაპროგრამებისთვის ნებისმიერი ტექსტური რედაქტორი გამოდგება. დღეს-დღეობით გავრცელებუ-ლია ხელსაწყოები:

- WebStorm (http://www.jetbrains.com/webstorm/) და
- Sublime Text (http://www.sublimetext.com).

ორივე ფასიანია და მომხმარებელს სთავაზობს AngularJS-თან გაუმჯობესებულ სამუშაო გარემოს. გარდა ამისა, კვლავ რჩება

კომპანია Microsoft-ის ხელსაწყო Visual Studio (უფასო ვერსია Express). Visual Studio ეშვება მხოლოდ Windows გარემოში, მაგრამ გააჩნია შესანიშნავი IDE გარემო და კოდის რედაქტორი არაფრით ჩამოუვარდება ზემოთ ჩამოთვლილ ხელსაწყოებს.

კლიენტის მხარის (Front-end) დაპროგრამების ხელსაწყოთა უმრავლესობა დაწერილია JavaScript ენაზე და ეშვება Node.js გარემოში. Node.js წარმოადგენს Google Chrome ვებ-ბრაუზერში ჩაშენებულ JavaScript ძრავს, თუმცა ასევე გათვალისწინებულია ბრაუზერის გარეთ მუშაობის რეჟიმიც.

Node.js უზრუნველგვყოფს JavaScript აპლიკაციების საწერად ზოგადი დანიშნულების Framework-ით.

***მარტივი ვებ-სერვერის გამართვა*** დაპროგრამების პროცესი-სათვის საკმარისი იქნება. ამიტომ, სერვერის შექმნის მიზნით გამოვიყენებთ Node.js-ის მოდულებს სახელად connect და serve-static.

NPM არის Node-ს პაკეტების ინსტალატორი:

```
npm install connect
```

და

```
npm install serve-static
```

იგი ჩამოტვირთავს ყველა საჭირო ფაილს, რაც connect და serve-connect მოდულებს ჭირდებათ. ამის შემდეგ შევქმნით ახალ ფაილს სახელად server.js:

```
<!- ლისტინგი_1.1: Serve.js ფაილის კონტენტი -----
var connect = require('connect');
var serveStatic = require('serve-static');
var app = connect();
app.use(serveStatic("../angularjs"));
app.listen(5000);
```

ეს მარტივი ვებ-სერვერი საკმარისია დეველოპმენტის პროცესისათვის. იგი ყურს უგდებს პორტ 5000-ზე შემოსულ

მოთხოვნებს და ემსახურება ფაილებს, რომლებიც იმყოფება angularjs დირექტორიაში. ხოლო angularjs დირექტორია იმყოფება იმავე დონეზე, რომელზეც nodejs საინსტალაციო დირექტორია.

***ტესტირების სისტემის ინსტალაცია:*** AngularJS ტექნოლოგიის ერთ-ერთი დიდი უპირატესობაა Unit Testing ტესტირების სისტემასთან მაღალი შესაბამისობის დონე. ეს საშუალებას გვაძლევს დეველოპმენტის პროცესშივე ავტომატურად შევამოწმოთ კოდის სისწორე. ტესტების გასაშვებად გამოიყენება Karma Test Runner ტესტირების გარემო და Jasmine ტესტირების Framework. ორივე მათგანი ფართოდ არის გავრცელებული და ადვილად მოხმარებადია. Node.js საინსტალაციო დირექტორიდან გავუშვათ შემდეგი ბრძანება:

```
npm install –g karma
```

NPM-ი გადმოწერს ყველა იმ ფაილს, რაც Karma-ს ესაჭიროება. შემდგომი კონფიგურირება აღარ არის საჭირო. Karma ტესტირების გარემო Unit Test ტესტებს უშვებს Node.js სერვერზე, ხოლო თავად ტესტი შეგვიძლია შევიმუშაოთ Jasmine, QUnit ან სხვა ბიბლიოთეკების გამოყენებით. ამ ეტაპზე Karma-ს მუშაობის პრინციპებს არ განვიხილავთ.

***AngularJS დირექტორიის შექმნა:*** უნდა შევქმნათ დირექტორია, საიდანაც მოვემსახურებით AngularJS აპლიკაციებს დეველოპმენტის პროცესში. ეს საშუალებას გვაძლევს დაპროგრამების პროცესშივე შევამოწმოთ სამუშაოს პროგრესირება და ორგანიზება გავუკეთოთ ფაილებს. შევქმნათ საქაღალდე სახელწოდებით angularjs. ახალი საქაღლდე იმავე დონეზე უნდა იმყოფებოდეს, რომელზეც შევქმენით nodejs საინსტალაციო დირექტორია.

***ვებ-სერვერის გაშვება*** ხდება Command Prompt-ში (cmd). დავდგეთ Nodejs საინსტალაციო დირექტორიაზე (უნდა ავკრიფოთ cd C:\Program Files\nodejs) და გავუშვათ ბრძანება:

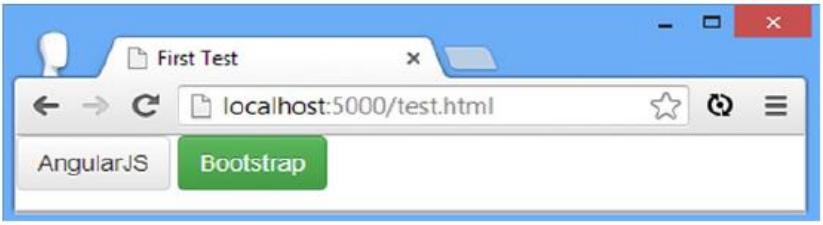
```
node server.js
```

ეს ჩატვირთავს წინა ნაწილში angularjs დირექტორიაში შემნილ server.js ფაილს და პორტზე ნომრით 5000 ჩაირთვება HTTP მოთხოვნებზე ყურის გდების რეჟიმი.

***სატესტო HTML-დოკუმენტის ჩატვირთვა:*** გავხსნათ Chrome ბრაუზერი და მისამართის ველში ავკრიფოთ:

http://localhost:5000/test.html.

შედეგად ბრაუზერში ჩაიტვირთება ახლადშექმნილი HTML-დოკუმენტი (ნახ.2.2).




**ნახ.2.2. სამუშაო გარემოს შემოწმება**

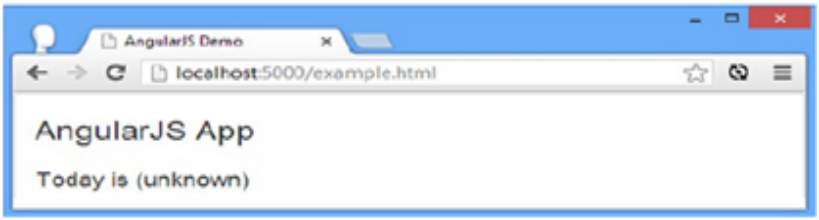
***საილუსტრაციო მაგალითი:*** ავაგოთ მარტივი სტრუქტურის მქონე პროექტი. გავაცარიელოთ angularjs საქაღალდე და დავამატოთ ფაილები: angular.js, bootstrap.css და bootstrap-theme.css.

angularjs საქაღალდეში დავამატოთ ახალი HTML ფაილი, სახელად example.html და დავარედაქტიროთ მისი კონტენტი, როგორც ეს 2_1 ლისტინგშია ნაჩვენები.

```
<!- ლისტინგი 2.1: example.html ფაილის კონტენტი -->
<!DOCTYPE html>
<html ng-app="exampleApp">
<head>
   <title>AngularJS Demo</title>
   <link href="bootstrap.css" rel="stylesheet" />
   <link href="bootstrap-theme.css" rel="stylesheet" />
   <script src="angular.js"></script>
   <script>
```

```
    var myApp = angular.module("exampleApp", []);
    myApp.controller("dayCtrl", function ($scope) {
      // controller statements will go here
    });
  </script>
</head>
<body>
  <div class="panel" ng-controller="dayCtrl">
    <div class="page-header">
      <h3>AngularJS App</h3>
    </div>
    <h4>Today is {{day || "(unknown)"}}</h4>
  </div>
</body>
</html>
```

გვერდი შეიცავს AngularJS ტექნოლოგიის მინიმალურ ნაწილს. გვერდში გამოყენებულია მონაცემთა ბმის გამოსახულება, რომელიც ამ მომენტში არ არის გამართული, ამიტომ JavaScript-ის ლოგიკური ოპერატორის გამოყენებით ( || ) მივუთითეთ, რომ გამოვიტანოთ day ცვლადის მნიშვნელობა, თუ იგი არსებობს, ხოლო თუ day ცვლადი არ არის განსაზღვრული, მაშინ - სტრიქონი (unknown). 2.3 ნახაზზე ნაჩვენებია როგორ გამოიყურება ჩვენი HTML დოკუმენტი ვებ ბრაუზერში.




**ნახ.2.3. Example.html დოკუმენტის ჩატვირთვა ვებ-ბრაუზერში**

### 2.2.3.2. Angular Framework ტექნოლოგია

AngularJS- და Angular (ახალ ვერსიებს შორის), მათი გამოყენების მეთოდები განსხვავდება, AngularJS - ის გამოყენება გაცილებით მარტივია, ჩვენს პროექტში საკმარისია მხოლოდ მისი Javascript ბიბლიოთეკის იმპორტირება მოვახდინოთ, თუნდაც ინტერნეტ ლინკის მეშვეობით. რა თქმა უნდა, შეგვიძლია მისი ჩამოტვირთვა და ლოკალური ფაილის გამოყენება (ნახ.2.4).

```
<!DOCTYPE html>
<html lang="en-US">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="">
  <p>Name : <input type="text" ng-model="name"></p>
  <h1>Hello {{name}}</h1>
</div>

</body>
</html>
```

**ნახ.2.4. Angular-ის ფაილი**

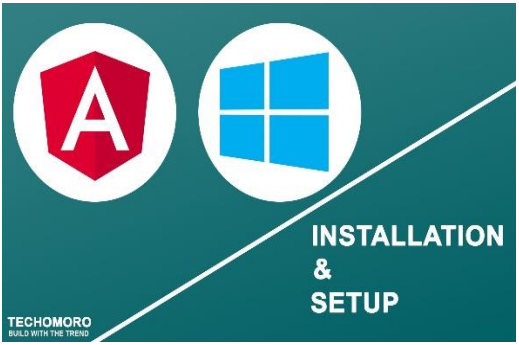
ეს პრინციპი ნაცნობია ვებ-დეველოპერებისთვის, რადგან მრავალი Plugin, რომლებსაც ვებ-დეველოპერები იყენებენ, (მაგალითად, jQuery) ამგვარი მეთოდით ხდება. იმპორტირების შემდგომ ჩვენს პროექტში უკვე ინტეგრირებულია AngularJS და შესაძლებელია მისი გამოყენება.

AngularJS - ისგან განსხვავებით, Angular - ის ახალი ვერსია, როგორც უკვე აღვნიშნეთ, მუშაობს ისეთ ენაზე, როგორიცაა TypeScript, მისი წაკითხვა არ ხდება ბრაუზერის მიერ, რადგან საჭიროა მოხდეს TypeScript - ის კომპილირება javascript ენად,

იმისათვის რომ ბრაუზერმა შეძლოს მისი აღქმა, Angular-ის დაყენება შედარებით მეტ შრომას მოითხოვს.

Angular - ის დაყენება შესაძლებელია სხვადასხვა სისტემაზე როგორებიცაა Microsoft Windows ან Mac OSX.

ჩვენ შემთხვევაში განვიხილოთ Windows-ზე ინსტალაცია (ნახ.2.5).



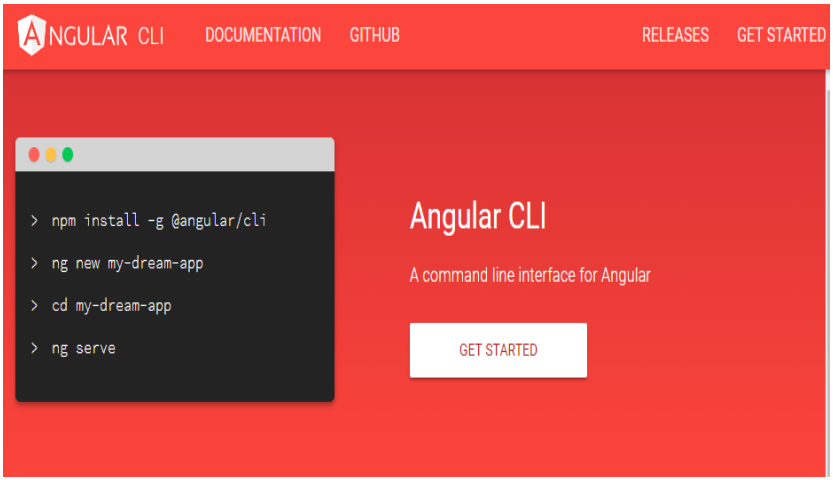

**ნახ.2.5. Windows-ზე ინსტალაცია**

პირველ რიგში საჭიროა ჩამოვტვირთოთ Node.js.

- **Angular CLI**

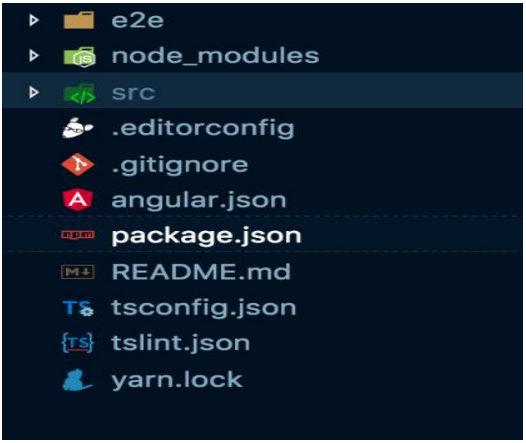
Node.js სა და Windows CMD -ს დახმარებით, შეგვიძლია მოვახდინოთ Angular CLI (command line interface for Angular)-ის დაყენება, ის თუ როგორ ხდება აღნიშნული საკითხის შესრულება შეგვიძლია ვნახოთ ვებ - გვერდზე რომლის ლინკია https://cli.angular.io/. ის ამგვარად გამოიყურება (ნახ.2.6).

Angular CLI - ის მეშვეობით კი შეგვიძლია ჩამოვტვირთოთ Angular - ის ახალი პროექტი, და მოვახდინოთ მისი გაშვება, აღნიშნული საკითხის ბრძანებები, ზემოთმოყვანილ ნახაზზეა აღნიშნული.




ნახ.2.6. Angular CLI - დაყენება

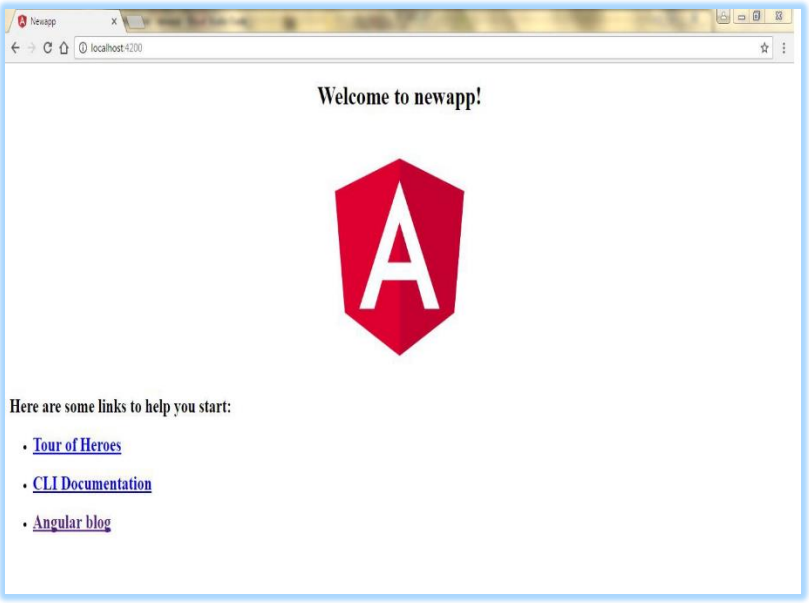
ჩამოტვირთული ფაილების სტრუქტურა ამგვარად გამოიყურება (ნახ.2.7).




ნახ.2.7. ჩამოტვირთული ფაილების სტრუქტურა

იმისათვის რომ მოხდეს აღნიშნული პროექტის ვებ-ბრაუზერში გაშვება, საჭიროა CMD ში მოვახდინოთ ბრძანება ng serve-ის გაშვება, რომელიც საბოლოოდ ჩვენს ვებგვერდს აამუშავებს მისამართზე localhost:4200.

მისი გახსნის შედეგად ვნახავთ რომ ჩვენი პირველი გვერდი შესაძლოა ამგვარად გამოუყურებოდეს (ნახ.2.8).




**ნახ.2.8. ვებგვერდს აამუშავების შედეგი**

რას წარმოადგენს ჩვენი პირველი გვერდი, საიდან მოხდა მისი გახსნა? აღნიშნული გვერდი შექმნილია სადემონსტრაციოდ მხოლოდ, რომელიც წარმოადგენს Angular - ის კომპონენტს.

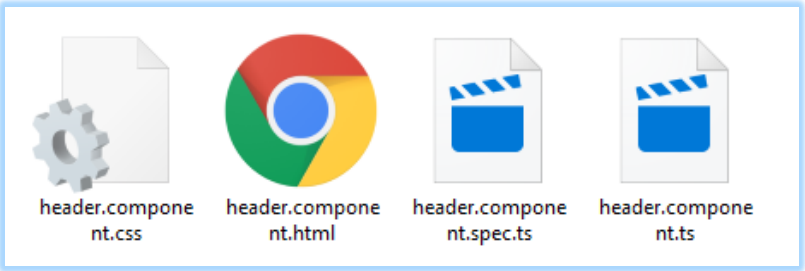
- **Angular კომპონენტები**

Angular - ენის უპირატესობა ტრადიციული ვებ-გვერდიდან სწორედ ეს გახლავთ, რომ ის მუშაობს არა მთლიანი გვერდის გადატვირთვის პრინციპით, არამედ შედგება კომპონენტებისგან, რომლებიც, ვებ-გვერდის სხვადასხვა ნაწილებია. მაგალითად, header, footer, მენიუ, მთავარი გვერდის შიგთავსი.

ისინი მოთავსებულია src საქაღალდის შიგნით App საქაღალდეში, მათ შესაქმნელად გვეხმარება Angular CLI.

კომპონენტის შესაქმნელად საჭიროა CMD ში გავუშვათ ბრძანება ng generate component "კომპონენტის სახელი", კომპონენტის სახელის ადგილას ვწერთ ჩვენთვის სასურველი კომპონენტის სახელს, მაგალთად header, footer, რისი მეშვეობით მოხდება კომპონენტის შექმნა.

რისგან შედგება კომპონენტი ? განვიხილოთ მაგალითად header კომპონენტი. ng generate component header ბრძანების გაშვების შედეგად მივიღებთ საქაღალდე header - ს, რომლის შიგთავსი აღნიშნული ფაილებს შეიცავს (ნახ.2.9).



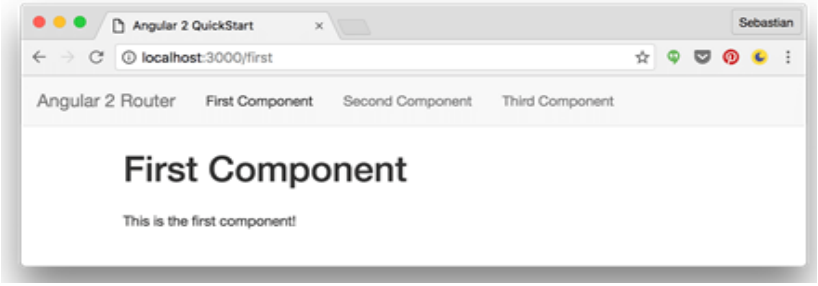

**ნახ.2.9. header-კომპონენტის შიგთავსი**

როგორც ნახაზიდან ჩანს, ის შედგება html, css, ts ფაილებისგან, ხოლო spec.ts გაფართოების ფაილი ტესტირებისთვის საჭირო ფაილს წარმოადგენს.

კომპონენტი header ის html, css ნაწილები ვიზუალურ მხარეს წარმოადგენენ, ხოლო ბიზნეს ლოგიკის გაწერა ts ფაილში ხდება რომელიც TypeScript - ის აბრევიატურაა.

კარგი სიახლე კომპონენტში გახლავთ ის, რომ შეგვიძლია კომპონენტის, სტილები CSS-ები ლოკალურად header - ისთვის გავწეროთ მხოლოდ (რა თქმა უნდა Angular-ს აქვს სტილების გლობალური ფაილიც, რომელიც გლობალურად ახდენს მთელს დოკუმენტში სტილების დანიშვნას). აგრეთვე ხშირ შემთხვევაში, საიტის header, footer ფაქტიურად უცვლელია.

ამგვარად შეგვიძლია მათი ერთჯერადად ჩამოტვირთვა მოხდეს, ხოლო სხვა გვერდების შექმნა, რომლებიც საიტის ცვლადი შიგთავსებია, როგორებიც შესაძლოა იყოს home, contact-კომპონენტები, შეგვიძლია Angular-ის routing სისტემის მეშვეობით ვცვალოთ, რაც ბრაუზერის მისამართის ნაწილში მოახდენს მიმდინარე გვერდის მისამართის გაწერას (ნახ.2.10).




**ნახ.2.10. ბრაუზერის მიმდინარე გვერდი**

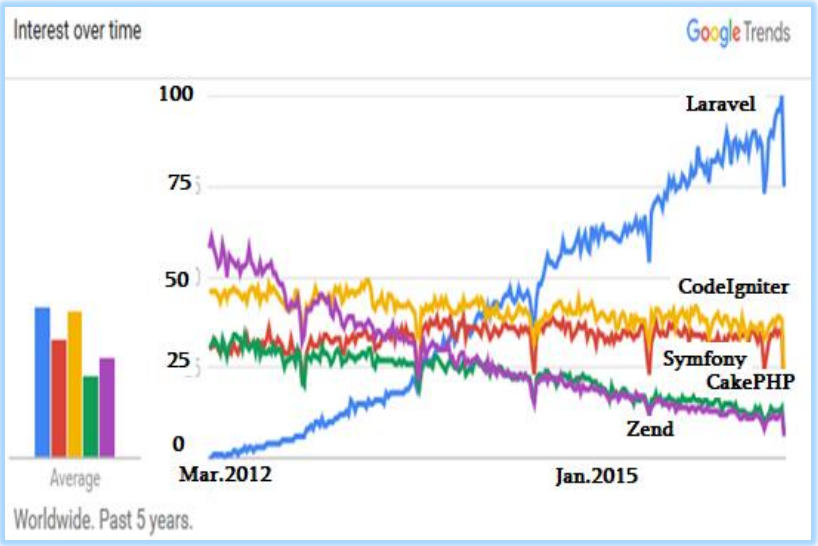
როგორც ნახაზზეა ნაჩვენები, მისამართში first წარმოადგენს routing-ის მისამართის ნაწილს, რომელიც დაკავშირებულია პირველ კომპონენტთან და ახდენს მის გამოტანას ბრაუზერში, რისი შესრულების დროსაც არ არის საჭირო გვერდის გადატვირთვა. Angular-ში, რა თქმა უნდა, მხოლოდ component-ების დაყენება არ ხდება, სხვა ნაწილების შექმნაცაა შესაძლებელი, როგორიცაა, მაგალითად, სერვისები.

## 2.3. PHP famework Laravel

დღეისათვის ე.წ. native წერა, როგორც front-end აგრეთვე back-end დეველოპმენტში არ არის მიზანშეწონილი, გამომდინარე იქიდან, რომ არსებობს უკვე მზა საშუალებები, რომლებიც framework-ებად იწოდება და ძალიან გვიმარტივებს აპლიკაციების შექმნას.

Back-end ენებში როგორიცაა PHP არსებობს არაერთი framework, როგორებიცაა: Laravel, CodeIgniter, Symfony, Zend და სხვ. [37]. ყველა მათგანი PHP - სთვის არის განკუთვნილი.

Laravel არის MVC არქიტექტურის მოდელზე მომუშავე open source ფრეიმვორკი, რომელსაც ბევრი დეველოპერი იყენებს დღეს. მისი გამოყენების სტატისტიკა 2012 წლიდან იზრდება, Google მონაცემებზე დაყრდნობით (ნახ.2.11).




**ნახ.2.11. Back-end ენების გამოყენების სტატისტიკა**

laravel ფრეიმვორკს აქვს შემდეგი უპირატესობები:

- ავტორიზაციის ტექნიკა
- ობიექტ ორიენტირებული ბიბლიოთეკები

- Artisan (დეველოპერებისთვის განკუთვნილი საშუალება, რომელიც გამოიყენება CMD ს საშუალებით)
- MVC
- დაცულობა
- Database Migration
- ძალიან კარგი გაკვეთილები www.laracasts.com
- Blade სისტემა

ეს ის არასრული სიაა, რისგანაც Laravel შედგება. ის ძალიან მოქნილი აგრეთვე დღესდღეისობით სწრაფად განვითარებადი და თანამედროვეობაზე მორგებული frameworkსია, რომელიც ძალიან აადვილებს ვებ-აპლიკაციების შექმნას, საჭიროა მისი სინტაქსის გარკვევა, რომელიც დოკუმენტაციიდან და ვიდეო გაკვეთილებიდან შეიძლება ისწავლოთ [38].

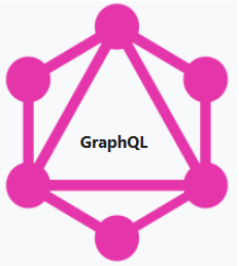
Laravel დეველოპერებს სთავაზობს გამარტივებულ საშუალებებს ვებ-პროექტებზე მუშაობისთვის, მაგალითად Artisan არის Command Line დან სამუშაო საშუალება, საიდანაც ხდება controller, model, migration გაშვება, ასევე ბაზებთან მუშაობა და სხვ. საშუალებები.

Laravel იყენებს MVC-ს, იგი იშიფრება Model - View - Controller, ის არის მოწესრიგებული საშუალება, რომელიც ახდენს კოდის ლოგიკური ნაწილის გამოყოფას მისი საპრეზენტაციო ნაწილისგან (view).

Model არის ბაზასთან სამუშაო საშუალება, View კლიენტის საპრეზენტაციო მხარე, Controller - კი გახლავთ შუამავალი ნაწილი, რომელიც ღებულობს კლიენტის მოთხოვნებს Route-ების საშუალებით და ახდენს მოთხოვნისდა მიხედვით მოდელისთვის გარკვეული დავალების დაკისრებას. Model ღებულობს ბრძანებას, ახდენს პასუხის ფორმირებას და უბრუნებს Controller მხარეს, რომელიც თავის მხრივ აბრუნებს მიღებულ შედეგს View ნაწილში.

## 2.4. GraphQL: მოთხოვნების და მონაცემთა მანიპულირების ენა

GraphQL (Graph Query language) - არის ფეისბუკის კომპანიის მიერ (2012-2015 წწ.) დამუშავებული მოთხოვნების და მონაცემთა მანიპულირების ენა აპლიკაციების პროგრამირების ინტერფეისისათვის (API – Application Programming Interface) (ნახ.2.12) [53,54].




ნახ.2.12

იგი უზრუნველყოფს ეფექტურ, მძლავრ და მოქნილ მიდგომას ვებ-აპლიკაციების ინტერფეისების (Web-API) დასამუშავებლად და ვებ-სერვისების არქიტექტურის რეალიზაციისთვის.

კლიენტს შესაძლებლობა აქვს განსაზღვროს მისთვის საჭირო მონაცემთა სტრუქტურა და ამ სახით მიიღოს იგი სერვერიდან. ეს კი გამორიცხავს მონაცემთა დიდი მოცულობების გადაცემის აუცილებლობას, რაც მნიშვნელოვნად ეფექტურს ხდის მოთხოვნების შედეგებისათვის *ვებ-კეშირების* პროცესს (web cache ინფორმაციული ტექნოლოგიაა ვებ-დოკუმენტების: ვებ-გვერდების, გამოსახულებების, მულტიმედიური სხვადასხვა ტიპების დროებით შესანახად). ეს კი ამცირებს სერვერის დატვირთვას და შესაძლებელს ხდის ვებ-კეშში მოთავსებულ დოკუმენტთა ასლების ხელმეორედ გამოყენების შესაძლებლობას სხვა მოთხოვნების მიერ. ამასთანავე, GraphQL უზრუნველყოფს მონაცემთა წაკითხვის, ჩაწერის და ცვლილების პროცედურების განხორციელებას რეალურ დროში.

GraphQL სერვერები გამოიყენება დაპროგრამების მრავალი ენისათვის, როგორებიცაა: JavaScript, PHP, Python, Ruby, Java, C#, Go, R და სხვ. [55].

# III თავი

# ReactJS - ინტერფეისების დაპროგრამების ბიბლიოთეკა

## 3.1. რეაქტიული დაპროგრამების პარადიგმა

რეაქტიული დაპროგრამება (Reactive programming) - არის დეკლარაციული პროგრამირების პარადიგმა, რომელიც ორიენტირებულია მონაცემთა ნაკადებზე და ცვლილებების გავრცელებაზე [6]. ეს ნიშნავს, რომ უნდა არსებობდეს მონაცემთა სტატიკური და დინამიკური ნაკადების ადვილად გამოსახვის შესაძლებლობა, აგრეთვე შესრულების მოდელის მიერ ცვლილებების ავტომატური გავრცელება მონაცემთა ნაკადების საფუძველზე. ანუ, შედეგი ყოველთვის ხელახლა განისაზღვრება ავტომატურად, როცა მონაცემთა საწყისი ნაკადები იცვლება.

ლიტერატურაში რეაქტიული დაპროგრამების მაგალითად განიხილავენ თანამედროვე ცხრილურ პროცესორებს (მაგალითად, Ms Excel), სადაც (მაგალითად A20) უჯრას აქვს სტრიქონული მნიშვნელობა ფორმულის სახით: “ = B5+C17”. მისი მნიშვნელობა დამოკიდებულია B5 და C17 უჯრების მნიშვნელობებზე. თუ შეიცვალა რომელიმე ამ უჯრის მნიშვნელობა, მაშინვე ავტომატურად იცვლება A20 უჯრის მნიშვნელობაც.

რეაქტიული პროგრამირება შემოთავაზებული იყო როგორც გზა მომხმარებელთა ინტერფეისების, ანიმაციების ან დროში ცვალებადი სისტემების მოდელირების ადვილად შექმნის მიზნით, რომელიც *სწრაფად მოახდენდა რეაქციას ცვლილებაზე*.

მაგალითად, MVC არქიტექტურაში რეაქტიული პროგრამირების დახმარებით შესაძლებელია Model-იდან View-ში და პირიქით, View-დან Model-ში ცვლილებების ავტომატური ასახვის რეალიზება.

წინამდებარე თავში ჩვენ განვიხილავთ მომხმარებელთა ინტერფეისების დამუშავების მიზნით JavaScript-ის ბიბლიოთეკას, რომელიც React - სახელითაა ფიქსირებული. იგი უზრუნველყოფს

მარტივი, სწრაფი და მასშტაბირებადი ერთგვერდიანი და მობილური აპლიკაციების აგებას [6].

## 3.2. ReactJS – ინტერფეისების აგების JavaScript ბიბლიოთეკა

ReactJS არის JavaScript-ის ბიბლიოთეკა ღია საწყისი კოდით, რომელიც გამოიყენება მომხმარებლის ინტერფეისების შესაქმნელად სპეციალურად ერთგვერდიანი აპლიკაციებისათვის [6,39]. იგი გამოიყენება ვებსაიტისა და მობილური აპლიკაციების წარმოდგენის (View) შრის დასამუშავებლად. React გვაძლევს ასევე საშუალებას შეიქმნას სამომხმარებლო ინტერფეისის (UI) მრავალჯერადად გამოყენებადი (reusable) კომპონენტები.

ReactJS-ის შესწავლისათვის აუცილებელია JavaScript (ასევე HTML5 და CSS) ენის ცოდნა. მიუხედავად იმისა, რომ ReactJS არ იყენებს HTML-ს, JSX არის მსგავსი, ამიტომ HTML ცოდნა იქნება ძალზე სასარგებლო. ასევე სასურველია MVC და SIngle Page Architecture-ს კონცეფციების ცოდნა. სასურველია React Native ფრეიმვორკის გაცნობა.

- **როგორ მუშაობს React ?**

React-ის კომპონენტი - ესაა კოდის ფრაგმენტი, რომელიც ასახავს გვერდის ნაწილს. ყოველი კომპონენტი JavaScript-ის ფუნქციაა, რომელიც აბრუნებს კოდის ფრაგმენტს, რაც შეესაბამება ვებ-გვერდის ფრაგმენტს. React იყენებს JSX ენას, რომელიც HTML-ის მსგავსია, ოღონდაც იგი მუშაობს JavaScript-ის შიგნით (HTML არა).

- **React Native** - არის JavaScript-ის ფრეიმვორკი საკუთარი მობილური აპლიკაციების შესაქმნელად. იგი იყენებს React პლატფორმას და იძლევა მრავალ ჩაშენებულ კომპონენტს და აპლიკაციის პროგრამულ ინტერფეისს (API - application programming interface). React-ით იგი ვებ-გვერდზე პირდაპირ ტრანსლირებას უკეთებს **DOM** ბრაუზერს. React Native-სთვის mark-

up (მონიშვნა, ფორმატირება) ტრანსლირდება ხოსტის პლატფორმის შესაბამისად Android-ისთვის დამახასიათებელი სპეციფიკაციით. React Native ფაქტობრივად მუშაობს JavaScript ფაილების ჩასაშენებლად აპლიკაციაში და ლოკალური შესრულებისათვის. *React Native API* - არის ნამდვილი მობილური აპლიკაცია. React Native არ ქმნის მობილურ ვებ-აპლიკაციებს, რადგანაც იგი იყენებს მომხმარებლის ინტერფეისის იმ ძირითად სამშენებლო ბლოკებს, რომელსაც იყენებს ჩვეულებრივი აპლიკაციები iOS-ის და Android-სთვის. ამ სამშენებლო ბლოკების შესაერთებლად აქ გამოიყენება JavaScript და React, ნაცვლად Swift, Kotlin или Java ენებისა [41].

- **DOM და VDOM ობიექტები ReactJS-ში**.

Document Object Model (DOM) არის HTML და XML დოკუმენტების პროგრამირების ინტერფეისი. იგი ასახავს გვერდს ისე, რომ პროგრამებს შეუძლია შეცვალოს მათი სტრუქტურა, სტილი და შინაარსი. DOM ასახავს დოკუმენტს კვანძებისა და ობიექტების სახით. აქ შესაძლებელია პროგრამული ენების მიერთება გვერდთან. ვებ-გვერდი - არის დოკუმენტი, რომელიც შეიძლება ასახული იყოს ბრაუზერის ფანჯარაში ან როგორც HTML-ის წყარო. ორივე ერთიდაიგივეა. DOM არის ვებ-გვერდის ობიექტ-ორიენტირებული წარმოდეგენა, რომლის ცვლილება შესაძლებელია სცენარების ენით, მაგალითად JavaScript-ით.

DOM არაა პროგრამირების ენა, მაგრამ მის გარეშე JavaScript-ს არ ექნებოდა არავითარი მოდელი ან ცნებები ვებ-გვერდის, HTML და XML დოკუმენტების და მათი შემადგენელი ელემენტების შესახებ. დოკუმენტის თითოეული ელემენტი, მთლიანი დოკუმენტი, სათაურები, ცხრილები დოკუმენტში, ტექსტები ცხრილის უჯრებში - არის დოკუმენტის ობიექტური მოდელის ნაწილი. ამიტომ მათთან მიმართვა და მანიპულაციების ჩატარება შესაძლებელია DOM-ის და JavaScript-ის საშუალებით.

Virtual Document Object Model (VDOM) – არის JavaScript-ის მსუბუქი ობიექტი, რომელიც რეალური DOM-ის უბრალო ასლია. ესაა კვანძების ხე, სია ელემენტების, მათი ატრიბუტების და შედგენილობისა, როგორც ობიექტებისა და თვისებების. React-ის რენდერის ფუნქცია აგებს ამ ხეს. DOM-ის ცვლილება ყოველთვის იწვევს მისი შესაბამისი VDOM-ის ცვლილებას.

- **React Fiber და React360.** ფეისბუკმა 2017 წ. გამოუშვა React Fiber - ახალი front-end ალგორითმი React Framework ბიბლიოთეკისთვის. იგი მომავალში უნდა გახდეს React-ის ინფრასტრუქტურის ფუნქციების დამუშავების ინსტრუმენტი {27}. React 360 შემუშავდა ინფორმაციული საზოგადოების ფართო სპექტრისთვის და მისი გამოყენება ორიენტირებულია ვირტუალური რეალობის სისტემებისაკენ [43,45].
- **React-ის მახასიათებლები** [40]:
  - **JSX –** ესაა JavaScript-ის გაფართოებული სინტაქსი. React-ში არაა აუცილებელი JSX გამოყენება, მაგრამ რეკომენდებულია;
  - **Components –** React არის ყველაფერი კომპონენტების შესახებ. კომპონენტებით აზროვნება განსაკუთრებით საყურადღებოა დიდი პროექტების დამუშავების დროს;
  - **Unidirectional data flow and Flux –** მონაცემთა ერთმიმართულებიანი ნაკადი ხორციელდება React-ში, რაც აადვილებს აპლიკაციის ანალიზს. Flux არის შაბლონი (pattern), რომელიც გვეხმარება მონაცემთა ერთმიმართულებიანი სახით შესანახად;
  - **License –** ლიცენზირება React-ისა ხდება Facebook Inc ფარგლებში. დოკუმენტაციისა კი - CC BY 4.0 -ის შესაბამისად.

❖ ***Reast-ის უპირატესობები:***

✓ იყენებს ვირტუალურ DOM-ს, რომელიც JacaScript-ის ობიექტია. ეს ამაღლებს აპლიკაციის მწარმოებლურობას, რადგან VDOM მუშაობს უფრო სწრაფად, ვიდრე DOM;

✓ შეიძლება გამოყენებულ იქნას როგორც კლიენტის მხარეს (front-end), ასევე სერვერის მხარეს (back-end). აგრეთვე სხვა პლატფორმებთან;

✓ კომპონენტებისა და მონაცემთა შაბლონები (pattern) აუმჯობესებს კოდის კითხვადობას, რაც გვეხმარება დიდი აპლიკაციების მხარდაჭერაში.

❖ **Reast-ის შეზღუდვები:**

✓ ეხება მხოლოდ წარმოდგენის დონეს (View), ამიტომ საჭირო ხდება სხვა ტექნოლოგიების გამოყენებაც აპლიკაციის დამუშავების ინსტრუმენტების დასაკომპლექტებლად;

✓ იყენებს JSX-ის ჩადგმულ შაბლონებს, რაც შეიძლება არ იყოს მოხერხებული ზოგიერთი დეველოპერისათვის.

➢ **Redux_(JavaScript_library) –** არის JavaScript-ის ბიბლიოთეკა ღია საწყისი კოდით, რომელიც მართავს აპლიკაციის მდგომარეობას. იგი ხშირად გამოიყენება React- ან Angular-თან ერთად მომხმარებელთა ინტერფეისების ასაგებად [42]. მსგავსია Flux Facebook არქიტექტურისა. შეიცავს ინსტრუმენტებს, რომლებიც განსაკუთრებით ამარტივებს საცავის მონცემების გადაცემას.

## 3.3. ReactJS-ის კომპონენტები

ვებ-გვერდების აგების პროცესში ReactJS-ით განსაკუთრებული მნისვნელობა აქვს კომპონენტების ცნებას. კომპონენტების კომბინირებით შესაძლებელია აპლიკაციების სერვისების გამარტივება [40].

განვიხილოთ კომპონენტი - „აპლიკაცია“ (App). ამ კომპონენტს გააჩნია სათაური და შინაარსი (Header and Content). ეს სათაური და შინაარსი შეიქმნება ცალკე და შემდეგ ისინი მარტივად ჩაემატება ჩვენი აპლიკაციის კომპონენტის JSX ხეში. მხოლოდ აპლიკაციის კომპონენტი უნდა იქნას ექსპორტირებული. ქვემოთ ნაჩვენებია პროგრამის ლისტინგი.

ამჯერად განიხილება Stateless Example, ანუ მაგალითი „მდგომარეობის შენახვის გარეშე“.

App.jsx

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <Header/>
        <Content/>
      </div>
    );
  }
}
class Header extends React.Component {
  render() {
    return (
      <div>
        <h1>Header</h1>
      </div>
    );
  }
}
class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>Content</h2>
        <p>The content text!!!</p>
      </div>
```

```
    );
    }
}
export default App;
```

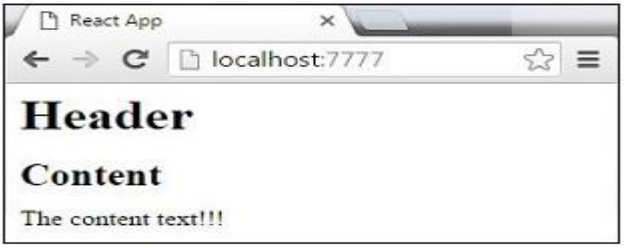
იმისათვის, რომ შედეგი ავსახოთ (ვიზუალურად) გვერდზე, საჭიროა მისი იმპორტირება main.js ფაილში და reactDOM.render() -ის გამოძახება.

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));
```

შედეგი ნაჩვენებია 3.1 ნახაზზე.




**ნახ.3.1. reactDOM.render() -ის შედეგი**

ახლა განვიხილოთ Stateful Example, ანუ მაგალითი „მდგომარეობის შენახვით“. აქ საჭიროა მდგომარეობის განსაზღვრა (დაყენება) მფლობელი-კომპონენტისთვის (App). Header კომპონენტი ემატება წინა შემთხვევის მსგავსად, რადგან მას არ ჭირდება არავითარი მდგომარეობა. content-ტეგის ნაცვლად ჩვენ ვქმნით table და tbody ელემენტებს, რომლებშიც დინამიკურად ჩავსვამთ TableRow-ს ყოველი ობიექტისათვის მონაცემთა მასივიდან.

პროგრამის ლისტინგიდან ჩანს, რომ გამოიყენება EcmaScript 2015-ის ისრის სინტაქსი (=>), ნაცვლად JavaScript-ის ძველი

სინტაქსისა. ეს უზრუნველყოფს ელემენტების შექმნას კოდის ნაკლები სტრიქონებით. განსაკუთრებით ეს სასარგებლოა დიდი რაოდენობის ელემენტების მქონე სიის შესაქმნელად.

App.jsx

```
import React from 'react';
class App extends React.Component {
   constructor() {
      super();
      this.state = {
     data:
     [
       {
         "id":1,
         "name":"Foo",
         "age":"20"
       },
       {
         "id":2,
         "name":"Bar",
         "age":"30"
       },
       {
         "id":3,
         "name":"Baz",
         "age":"40"
       }
           ]
          }
        }
    render() {
         return (
```

```
            <div>
              <Header/>
              <table>
               <tbody>
                {this.state.data.map((person, i) => <TableRow key = {i}
                      data = {person} />)}
               </tbody>
              </table>
            </div>
          );
        }
      }
      class Header extends React.Component {
        render() {
          return (
            <div>
              <h1>Header</h1>
            </div>
          );
        }
}
class TableRow extends React.Component {
  render() {
    return (
      <tr>
        <td>{this.props.data.id}</td>
        <td>{this.props.data.name}</td>
        <td>{this.props.data.age}</td>
      </tr>
    );
  }
```

```
}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App/>, document.getElementById('app'));
```

შენიშვნა: ჩვენ აქ ვიყენებთ key=[i] -ს map() ფუნქციის შიგნით. ეს ეხმარება React-ს განაახლოს მხოლოდ აუცილებელი ელემენტები და არა მთლიანი სია რენდერინგის ხელმეორედ გამოყენებით, როცა რამე შეიცვლება. ეს საგრძნობლად ამაღლებს აპლიკაციის მწარმოებლურობას დიდი რაოდენობის დინამიკური ელემენტების სემნის დროს.

შედეგი ნაჩვენებია 3.2 ნახაზზე.


**ნახ.3.2. ReactDOM.render ახალი შედეგი**

## 3.4 React-ის მდგომარეობა და რეკვიზიტები

➢ *მდგომარეობა (State)* - ესაა ადგილი, საიდანაც მოედინება მონაცემები. ჩვენი მიზანია შევქმნათ რაც შეიძლება მარტივი მდგომარეობა და „მდგომარეობის შენახვის“ მქონე კომპონენტების (stateful components) მინიმალური რაოდენობით. თუ, მაგალითად, ჩვენ გვაქვს 10 კომპონენტი, რომლებსაც ჭირდება მონაცემები მდგომარეობიდან, ჩვენ უნდა შევქმნათ ერთი კონტეინერული კომპონენტი, რომელიც შეინახავს მდგომარეობას ყველასთვის.

მომდევნო კოდის მაგალითში ნაჩვენებია თუ როგორ უნდა შევქმნათ stateful-ანი კომპონენტი EcmaScript2016 სინტაქსის გამოყენებით.

App.jsx

```
import React from 'react';

class App extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      header: "Header from state...",
      content: "Content from state..."
    }
  }
  render() {
    return (
      <div>
        <h1>{this.state.header}</h1>
        <h2>{this.state.content}</h2>
      </div>
    );
  }
```

```
}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(<App />, document.getElementById('app'));
```

შედეგი ნაჩვენებია 3.3 ნახაზზე..


**ნახ.3.3. stateful-ანი კომპონენტის შექმნა**

➢ ძირითადი განსხვავება მდგომარეობასა (state) და რეკვიზიტებს (props) შორის ისაა, რომ რეკვიზიტები უცვლელია. ამიტომაც კონტეინერის კომპონენტმა უნდა განსაზღვროს მდგომარეობა, რომელიც შეიძლება განახლდეს და შეიცვალოს. ამავდროულად, მისმა შვილობილმა კომპონენტებმა მხოლოდ უნდა გადასცეს მონაცემები მდგომარეობიდან, რეკვიზიტების გამოყენებით.

როდესაც ჩვენ გვჭირდება უცვლელი მონაცემები კომპონენტში, ჩვენ შეგვიძლია main.js კოდში უბრალოდ დავამატოთ props-ი reactDOM.render() ფუნქციას და გამოვიყენოთ იგი ჩვენი კომპონენტის შიგნით.

App.jsx

```
import React from 'react';
class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
ReactDOM.render(<App headerProp = "Header from props..."
contentProp = "Content from props..."/>,
document.getElementById('app'));

export default App;
```

შედეგი ნაჩვენებია 3.4 ნახაზზე.


ნახ.3.4. props-ი reactDOM.render() -ში

### ➢ მდგომარეობა და რეკვიზიტები

განვიხილოთ მაგალითი თუ როგორ შეიძლება მდგომარეობა (state) და რეკვიზიტები (props) გაერთიანდეს აპლიკაციაში. ჩვენ მოვათავსეთ მდგომარეობა მშობელ კომპონენტში და გადავცემთ მას ქვევით კომპონენტების ხით რეკვიზიტების საშუალებით. რენდერინგის ფუნქციის შიგნით ჩვენ ვაყენებთ headerProp და contentProp -ს, რომლებიც გამოიყენება შვილობილ კომპონენტებში.

App.jsx

```
import React from 'react';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      header: "Header from props...",
      content: "Content from props..."
    }
  }
  render() {
    return (
      <div>
        <Header headerProp = {this.state.header}/>
        <Content contentProp = {this.state.content}/>
      </div>
    );
  }
}
class Header extends React.Component {
  render() {
    return (
```

```
      <div>
<h1>{this.props.headerProp}</h1>
      </div>
    );
  }
}
class Content extends React.Component {
  render() {
    return (
      <div>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}
export default App;
```

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';
ReactDOM.render(<App/>, document.getElementById('app'));
```

შედეგად, მდგომარეობის მონაცემების განახლება შვილობილებშიც განაახლებს მონაცემებს.

## 3.5. ReactJS-ის კომპონენტების სასიცოცხლო ციკლი

წინამდებარე პარაგრაფში ვიხილავთ კომპონენტების სასიცოცხლო ციკლის მეთოდებს.

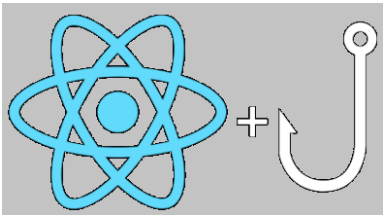
- componentWillMount – სრულდება რენდერინგის წინ როგორც სერვერის, ასევე კლიენტის მხარეს;.

- componentDidMount – სრულდება პირველი რენდერინგის შემდეგ მხოლოდ კლიენტის მხარეს. სწორედ აქ უნდა შესრულდეს Ajax-ის მოთხოვნები და DOM-ის ან მდგომარეობის განახლება. ეს მეთოდი გამოიყენება აგრეთვე JavaScript-ის სხვა პლატფორმებთან ინტეგრაციისთვის და სხვა ფუნქციებთან;
- componentWillReceiveProps – გამოიძახება, როგორც კი რეკვიზიტები განახლდება მომდევნო რენდერის გამოძახების წინ;
- shouldComponentUpdate – დააბრუნებს true ან false მნიშვნელობას. ეს განსაზღვრავს განახლდება თუ არა კომპონენტი. სტანდარტულად დაყენებულია true;
- componentWillUpdate – გამოიძახება უშუალოდ რენდერინგის წინ;
- componentDidUpdate – გამოიძახება რენდერინგის შემდეგ;.
- componentWillUnmount – გამოიძახება კომპონენტის დაშლის შემდეგ DOM-იდან. კომპონენტი დემონტირდება main.js -ში.

## 3.6. Hooks და Hooking ტექნოლოგია

კომპიუტერულ პროგრამირებაში ტერმინი hooking („ჩაჭრა“, „перехват“) მოიცავს სხვადასხვა მეთოდს, რომლებიც გამოიყენება ოპერაციული სისტემის, აპლიკაციების ან სხვა პროგრამული კომპონენტების ქცევის შეცვლის ან დამატების მიზნით,

პროგრამული უზრუნველყოფის კომპონენტებს შორის გადაცემული ფუნქციების (შეტყობინებების ან მოვლენების) „დაჭერის" გზით. კოდი, რომელიც ასრულებს ასეთი დაჭერილი ფუნქციების (შეტყობინებების ან მოვლენების) დამუშავებას, hook-ს უწოდებენ („ანკესი“, „მახე“) (ნახ.3.5) [46,47].



**ნახ.3.5. ReactHooks: ერთ-ერთი ლოგო**

ამგვარად, hooking - ტექნოლოგიაა, რომელიც ცვლის ინფორმაციული სისტემის ამა თუ იმ კომპონენტის სტანდარტულ ყოფაქცევას.

Hook არის მექანიზმი, რომლის დახმარებითაც აპლიკაციას შეუძლია „დაიჭიროს“ ისეთი მოვლენები, როგორიცაა შეტყობინება, მაუსის მოქმედება, კლავიატურაზე დაკლიკვა [48]. ფუნქცია, რომელიც დაიჭერს განსაზღვრული ტიპის მოვლენას, უწოდებენ hook-პროცედურას. მას შეუძლია ზემოქმედება ყოველ მიღებულ მოვლენაზე, შემდეგ კი შეცვალოს ეს მოვლენა ან გააუქმოს.

Hooks-ის კონცეფციის გამოყენების მაგალითებია:

- შეტყობინებათა მონიტორინგი პროგრამული სისტემის გამართვის პროცესში;
- პროგრამულ სისტემაში მაკროსების ჩაწერის და რეპროდუქციის უზრუნველყოფა;
- F1-ღილაკის (help key) მხარდაჭერის უზრუნველყოფა;
- მაუსისა და კლავიატურის იმიტაცია;
- კომპიუტერზე ბაზირებული სწავლების (CBT - computer-based training) აპლიკაციის რეალიზაცია.

*შენიშვნა:* Hook-ები, როგორც წესი, ანელებს სისტემის მუშაობის სწრაფქმედებას, რადგანაც მატულობს თითოეული შეტყობინების დამუშავების მოცულობა. საჭიროა, რომ Hook-ი დაყენდეს მხოლოდ საჭიროების შემთხვევაში და მისი გამოყენების შემდეგ - სასწრაფოდ წაიშალოს.

არსებობს Hook-ების სხვადასხვა ტიპი, რომლებიც უზრუნველყოფს შეტყობინებებზე (ან მოვლენებზე) წვდომის და დამუშავების განსხვავებულ ასპექტებს. მაგალითად, WH_MOUSE ჰუკი აპლიკაციის მიერ გამოიყენება მაუსიდან შემოსული შეტყობინებათა ტრაფიკის მონიტორინგისათვის [48].

Hook-ების ყოველი ტიპისათვის სისტემა უზრუნველყოფს ჰუკების ცალკე მწკრივის (Hook Chains) მხარდაჭერას. ესაა მაჩვენებლების სია (pointers list) აპლიკაციით განსაზღვრულ,

სპეციალურ ფუნქციებზე, რომლებსაც Hook Procedures-ს უწოდებენ.

როდესაც მოსულია შეტყობინება (ან მოხდა მოვლენა), რომელიც დაკავშირებულია განსაზღვრული ტიპის hook-თნ („მახესთან“), სისტემა გადასცემს შეტყობინებას ერთიმეორის მიმდევრობით ყოველ პროცედურას, რომლებიც მითითებულია Hook Chains-ში. ჰუკ-პროცედურებს შეუძლია სხვადასხვა ფუნქციის შესრულება მათი ტიპების შესაბამისად. მაგალითად, ზოგ პროცედურას შეუძლია შეტყობინების მხოლოდ მონიტორინგი, ზოგს - შეტყობინების შეცვლა ან მათი შეჩერება მწკრივში ისე, რომ მათ ვერ მიაღწიოს შემდეგ ჰუკ-პროცედურამდე ან დანიშნულების ფანჯრამდე.

რომელიმე ტიპის ჰუკისათვის უპირატესობის მისანიჭებლად დეველოპერს შეუძლია პროცედურულ ჰუკს SetWindowsHookEx ფუნქციით განუსაზღვროს ადგილი ჰუკების მწკრივში. ჰუკ-პროცედურას, მაგალითად, ექნება ასეთი სინტაქსი:

```
LRESULT CALLBACK HookProc(
  int nCode,
  WPARAM wParam,
  LPARAM lParam
)
{
  // process event
  ...

  return CallNextHookEx(NULL, nCode, wParam, lParam);
}
```

HookProc არის ადგილი სახელის მისათითებლად, რომელსაც განსაზღვრავს აპლიკაცია.

nCode პარამეტრი არის hook-კოდი, რომელსაც იყენებს hook-პროცედურა, რათა განსაზღვროს შესასრულებელი ქმედება. Hook-კოდის მნიშვნელობა დამოკიდებულია ჰუკის ტიპზე. ყოველ ტიპს აქვს hook -კოდების საკუთარი ერთობლიობა.

wParam და lParam პარამეტრების მნიშვნელობები დამოკიდებულია Hook-კოდზე, მაგრამ, ჩვეულებისამებრ, ისინი შეიცავს ინფორმაციას, იმ შეტყობინების შესახებ, რომელიც იქნა გაგზავნილი ან მიღებული.

SetWindowsHookEx ფუნქცია ყოველთვის აყენებს hook-პროცედურას hook-მწკრივის დასაწყისში. როდესაც ხდება მოვლენა, რომელსაც თვალყურს ადევნებს განსაზღვრული ტიპის ჰუკი, სისტემა გამოიძახებს hook-მწკრივს, რომელიც დაკავშირებულია ამ ჰუკთან. ყოველი ჰუკ-პროცედურა მწკრივში განსაზღვრავს - გადასცეს თუ არა მოვლენა შემდეგ პროცედურას. ამისათვის იგი გამოძახებით იყენებს CallNextHookEx ფუნქციას.

ქვემოთ მოცემულია HookTypes:

- WH_CALLWNDPROC და WH_CALLWNDPROCRET – შეტყობინებათა გაგზავნის მონიტორინგი;
- WH_CBT – computer-based training (CBT) applications;
- WH_DEBUG – განისაზღროს სისტემისათვის ჰუკ-პროცედურის გამოძახების ნებართვა;
- WH_FOREGROUNDIDLE – დაბალი პრიორიტეტის ჰუკ-პროცედურის გამოძახება;
- WH_GETMESSAGE – კლავიატურის ან მაუსის შეტყობინებების მონიტორინგი;
- WH_JOURNALPLAYBACK – შეტყობინებათა დაყენება სისტემურ შეტყობინებათა რიგში;
- WH_JOURNALRECORD – შემავალი მოვლენების კონტროლი და ჩაწერა;
- WH_KEYBOARD_LL – კლავიატურიდან შეტანილი მოვლენების მონიტორინგი;

- WH_KEYBOARD – შეტყობინებათა ტრაფიკის მონიტორინგი (კლავიატურიდან შეტანილი);
- WH_MOUSE_LL – მაუსიდან მიღებული მოვლენების მონიტორინგი
- WH_MOUSE – მაუსიდან მიღებული შეტყობინებების თვალყურისდევნება;
- WH_MSGFILTER და WH_SYSMSGFILTER – შეტყობინებათა თვალყურისდევნება მენიუს და დიალოგური ფანჯრებისთვის;
- WH_SHELL - გარსის აპლიკაციის გააქტიურება.

## 3.7. React Hooks: Memoization

ჩვენი ნაშრომის ძირითადი მიზანია თანამედროვე ვებ-ტექნოლოგიების ანალიზი, front-end და back-end აპლიკაციების აგების თვალსაზრისით. განსაკუთრებული ყურადღება გამახვილებული იყო JavaScript ენის საფუძველზე შექმნილი ინსტრუმენტებისა და ახალი ტექნოლოგიების ინტეგრაციის საკითხებზე.

უნდა გამოვყოთ AngularJS, Angular, NodeJS, Ajax, ReactJS და სხვა ფრეიმვორკები და ბიბლიოთეკები, რომელთა საფუძველზე მნიშვნელოვნად გაუმჯობესდა front-end და back-end ვებ-აპლიკაციების როგორც ხარისხობრივი, ასევე მწარმოებლურობის (სწრაფქმედების) მახასიათებლები.

ამჯერად, წინამდებარე პარაგრაფში გვინდა წარმოვადგინოთ ჩვენ მიერ შესწავლილი, გამოკვლეული და ჩამოყალიბებული ამოცანის გადაწყვეის საკითხი – როგორიცაა React Hooks: memoization პრობლემა. ამოცანა მიეკუთვნება Front End Development -ის სფეროს და მომხმარებელთა აპლიკაციების ფუნქციონირების სრულყოფის მიზნით ხორციელდება, კერძოდ, იგი ეხება სისტემის მწარმოებლურობის ამაღლებას [49].

ამასთანავე, წარმოდგენილია პრობლემის გადაწყვეტის გზები, კონკრეტული საილუსტრაციო მაგალითები და რეკომენდა-

ციები ფრონტ-ენდ აპლიკაციების ასაგებად React Hook-ის გამოყენებით.

აღნიშნული სტატიის ელექტრონული ვერსია წიგნის თანაავტორის, სანდრო დოლიძის მიერ გამოქვეყნებულია აშშ-ის, კერძოდ კალიფორნიის სან ფრანცისკოს ელექტრონულ ჟურნალში (ბლოგი MEDIUM) [49].

წიგნის 1-დანართში წარმოდგენილია აღნიშნული სტატიის ორიგინალი ვერსია.

## 3.8. React Hooks-ის აისბერგი

React Hooks-ები, კლასთა კომპონენტებისაგან განსხვავებით, დაბალი დონის „სამშენებლო ბლოკებია", აპლიკაციების ასაგებად და ოპტიმიზაციისათვის მინიმალური შაბლონური კოდებით (boilerplate). ეს უკანასკნელი, კომპიუტერულ პროგრამირებაში გამოიყენება და იგი კოდის ის ნაწილია, რომელიც პროგრამის ბევრ ადგილას თავსდება უცვლელად. დაპროგრამების ენებში, რომლებიც ითვლება მრავალსიტყვიანად (verbose), დეველოპერეს უხდება დიდი მოცულობის კოდის წერა შედარებით უმნიშვნელო ზომის ფუნქციონალის შესასრულებლად. ასეთ კოდს უწოდებენ შაბლონურს (boilerplate) [60].

შაბლონური კოდების გამოყენების მოთხოვნილება შეიძლება შემცირდეს მაღალი დონის მექანიზმების საშუალებით, როგორიცაა, მაგალითად, *მეტაპროგრამირება* (აქ კომპიუტერი თვითონ წერს ავტომატურად შაბლონურ კოდა ან სვამს მას პროგრამის კომპლიაციის დროს), *შეთანხმება კონფიგურაციის შესახებ* (რომელიც ავტომატურად უზრუნველყოფს კარგ მნიშვნელობებს, რათა შემცირდეს პროგრამისათვის დეტალების მითითების აუცილებლობა ყოველ პროექტში) და *მოდელურ-ორიენტირებული დაპროექტება* (რომელშიც გამოიყენება მოდელები და მოდელი->კოდის გენერატორები, რაც გამორიცხავს ხელოვნური კოდის შაბლონის აუცილებლობას).

დანართში_2 მოცემულია ავტორის, ს. დოლიძის მიერ შესრულებული და კალიფორნიის (აშშ) Medium ბლოგზე გამოქვეყნებული სტატია „The Iceberg of React Hooks“, სადაც დეტალურადაა განხილული აღნიშნული პრობლემა და მისი გადაწყვეტის ვერსია [52].

ამგვარად, აღნიშნულ კვლევით ნაშრომში, რომელიც 12 ნაწილისაგან შედგება, ილუსტრირებულია ზოგადი პრობლემები და მათი გადაწყვეტის ხერხები.

აგრეთცე მოცემულია React Hooks Radar და React Hooks Checklist მცირე რეკომენდაციები მოკლე ცნობარის სახით.

## დასკვნა

Web-დეველოპმენტის სფეროში ერთ-ერთი მნიშვნელოვანი მიმართულებაა მასშტაბირებადი ვებ-აპლიკაციების დამუშავება. back-end და front-end ტექნოლოგიები მუდმივად განიცდის განახლებას და სრულყოფას.

წარმოდგენილია ის თანამედროვე ტექნოლოგიები, რომლებიც მათ შესაქმნელად გამოიყენება და აქტუალურია, როგორიცაა: HTML, CSS, PHP, BEM მეთოდოლოგია. ხოლო ყველაზე მთავარი რასაც ჩვენი ნაშრომი მოიცავს ეს გახლავთ Angular და ReactJS სისტემების განხილვა, რომლებმაც ბაზარზე საკმაო პოპულარობა მოიპოვა ვებ-დეველოპერებში. ისინი დაცული და მრავალი ფუნქციით აღჭურვილი საშუალებებია, რომლებიც ვებ-აპლიკაციის შექმნას საგრძნობლად ამარტივებს და აუმჯობესებს მწარმოებლურობის ამაღლების თვალსაზრისით.

React Hooks-ის ინსტრუმენტის გამოყენება მასშტაბირებადი და ეფექტური ვებ-აპლიკაციების ასაგებად მეტად მნიშვნელოვანია. იგი გვეხმარება სუსტი ადგილების გამოვლენისა და მწარმოებლურობის ამაღლების პრობლემების გადაწყვეტაში.

## გამოყენებული ლიტერატურა:


1. ჩოგოვაძე გ., ფრანგიშვილი ა., სურგულაძე გ. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. სტუ. თბ., 2017. 1001გვ. http://gtu.ge/book/monacemta_menejmenti.pdf
2. Melnik I. How to Build Scalable Web Applications. 2018. https://merehead.com/blog/how-build-scalable-web-applications/
3. Hugo Di Francesco. In simple terms: backend code, frontend code and how they interact. 2017. https://tproger.ru/translations/frontend-backend-interaction/
4. Node.js. https://ru.wikipedia.org/wiki/Node.js
5. https://en.wikipedia.org/wiki/Event-driven_programming
6. https://en.wikipedia.org/wiki/Reactive_programming
7. https://ru.wikipedia.org/wiki/NW.js
8. http://getbem.com/introduction/
9. Social media addiction. https://en.wikipedia.org/wiki/Social _media_addiction
10. Mobile app development. https://en.wikipedia.org/wiki/Mobile _app _development
11. Android software development. https://en.wikipedia.org/wiki/Android_software_development
12. iOS mobile operating system. https://en.wikipedia.org/wiki/IOS_12
13. ვებ-საიტი. https://ka.wikipedia.org/wiki/საიტი
14. web application or web app. https://en.wikipedia.org/wiki/Web_application
15. TypeScript Language. https://ru.wikipedia.org/wiki/TypeScript
16. Anders Hejlsberg. https://en.wikipedia.org/wiki/Anders_Hejlsberg

17. HyperText Markup Language. https://en.wikipedia.org/wiki/ HTML

18. https://www.w3schools.com/html/html5_new_elements.asp

19. https://en.wikipedia.org/wiki/Cascading_Style_Sheets

20. What Is CSS3 And Why Is It Used ? 2012. https://www. webunlimited .com/css3-used/

21. CSS Flexible Box Layout. https://en.wikipedia.org/wiki/CSS _Flexible _Box_ Layout

22. Bootstrap (front-end framework). https://en.wikipedia.org/ wiki/Bootstrap_ (front-end_framework)

23. Bootstrap. http://getbootstrap.com

24. Block, Element, Modifier. https://de.wikipedia.org/wiki/ Block,_Element,Modifier

25. ვებ-აპლიკაციასთან სამუშაო პროგრამების (html, css, javascript, bootstrap, php, sql) მასალების ერთობლიობა: https://www.w3schools.com/ გადამოწმ. 18.12.2018

26. JavaScript-ფრეიმვორკი. https://en.wikipedia.org/wiki/Java Script_framework

27. React_JavaScript ბიბლიოთეკა მომხმარებელთა იმტერფეი-სების დამუშავება. https://en.wikipedia.org/wiki/React _(JavaScript _library)

28. ღვინეფაძე გ. WEB-დაპროგრამება jQuery. „ტექნიკური უნივერსიტეტი“, 2013. http://gtu.ge/View/index.html#http://gtu.ge/ book/JQuery.pdf

29. ღვინეფაძე გ. WEB-დაპროგრამება: WEB2.0, XML, AJAX. „ტექნიკური უნივ.“. http://gtu.ge/View/index.html#http://gtu.ge/book/ WEB_2_XML_ AJAX_gvinefadze.pdf

30. Ajax ტექნოლოგია. https://simple.wikipedia.org/wiki/Ajax _programming

31. AngularJS. https://ru.wikipedia.org/wiki/AngularJS

32. ღვინეფაძე გ. WEB-დაპროგრამება: AngularJS. „ტექნ.უნი-ვერსიტ.“, http://gtu.ge/book/AngularJS_g_gvinepadze.pdf

33. Angular Framework. https://en.wikipedia.org/wiki/Angular _(web_framework)

34. TypeScript ენა. https://en.wikipedia.org/wiki/TypeScript

35. Macquin G.B. SPA vs Multi page application. 2018. https://medium.com/@goldybenedict/single-page-applications-vs-multi-ple -page-applications-do-you-really -need-an-spa-cf60825232a3

36. Freeman A. Pro AngularJS (Expert's Voice in Web Development). 2014. http://www.hainanfuli.com/uploadfile/201412091 418121977.pdf.

37. Laravel ფრეიმვორკი. https://en.wikipedia.org/wiki/Laravel

38. Laravel framework-ის შემსწავლელი ვიდეო გაკვეთილები. 2019. https://laracasts.com/

39. Пацианский М. Основы React (текстовый учебник, 2-е издание). 2018. https://legacy.gitbook.com/book/maxfarseer/react-cour-se-ru-v2/details

40. ReactJS Tutorial. https://www.tutorialspoint.com/reactjs/

41. Stäbler R. Kotlin for Swift Developers. Medium programming. 2018. https://medium.com/@raphaelstaebler/kotlin-for-swift-developers-40e846fb7813

42. Redux ბიბლიოთეკა. https://en.wikipedia.org/wiki/Redux _(JavaScript_library). გადამოწმ. 15.01.19

43. Barrett L. Virtual Reality with React360. 2018. https://hacker-noon.com/virtual-reality-with-react-360-ce24b611f0f5

44. Gagliardi V. React Hooks Tutorial for Beginners: Getting Started with React Hooks. 2019. https://www.valentinog. Com/blog /hooks/

45. სურგულაძე გ., წაწიშვილი დ. ვირტუალური რეალობა და თანამედროვე საინფორმაციო ტექნოლოგიები. ISBN 978-9941-8-0626-1. სტუ, „IT კონსალტინგ ცენტრი". თბ., 2018. http://gtu.ge/book /Surgul_VirtualReality.pdf

46. Hooking. https://en.wikipedia.org/wiki/Hooking

47. Okeh K. How to Get Started With React Hooks: Controlled Forms. 2019. https://medium.freecodecamp.org/how-to-get-started-with-react-hooks-controlled -forms-826c99943b92

48. Kennedy J., Satran M. Hooks. Microsoft. 2018. https://docs. microsoft .com/en-us /windows/desktop/winmsg/hooks

49. Dolodze S. React Hooks: Memoization. 2019. A MEDIUM Co. San Francisco, California, US. https://medium.com/@sdolidze/react-hooks-memoization-99a9a91c8853

50. Vue.js. https://en.wikipedia.org/wiki/Vue.js

51. The Progressive JavaScript Framework. https://vuejs.org/

52. Dolodze S. The Iceberg of React Hooks. 2019. A MEDIUM Co. San Francisco, California, US. https://medium.com/@sdolidze/the-iceberg-of-react-hooks-af0b588f43fb

53. GraphQL. https://en.wikipedia.org/wiki/GraphQL

54. GraphQL: A data query language. https://code.fb.com/core-data/graphql-a-data-query-language/

55. Getting Started With GraphQL.js. http://graphql.org/graphql-js/

56. Maldonado L. A Beginner's Guide to GraphQL. 2019. https:// medium.freecodecamp.org/a-beginners-guide-to-graphql-86f849ce1bec

57. ხვედელიძე ნ. ვებ-სისტემების ხარისხის შეფასებისა და მართვის მეთოდების შემუშავება. სტუ-ს შრ.კრ. „მას“, N3(26), „ტექნიკური უნივ.“. თბ., გვ.288-291

58. გოგშელიძე დ. კვლევის მიზნები და მეთოდოლოგია “ტექსტური ინფორმაციის შექნისა და საგამომცემლო მარკეტინგის ერთიანი სისტემის” დამუშავებისას. სტუ-ს შრ.კრ. „მას“, N3(26), „ტექნიკური უნივ.“. თბ., გვ.261-264

59. Gogselidze D. Lorem: System for creating, searching and using books, articles, scientific papers and other textual information. Studio Master thesis. The supervisor. G Surguladze. Tbilisi. 2017.

60. Boilerplate. https://en.wikipedia.org/wiki/Boilerplate_code

დანართი_1

# REACT HOOKS: MEMOIZATION

https://medium.com/@sdolidze/react-hooks-memoization-99a9a91c8853

Sandro Dolidze. Mar 5, 2019

React Hooks make our life so much better in almost every way. But the minute performance becomes an issue, things get slightly tricky. You can write blazing fast applications using Hooks, but before you do that, there are a thing or two that you should be aware of.

## Should you memoize?

React is plenty fast for most use cases. If your application is fast enough and you don't have any performance problems, this article is not for you. Solving imaginary performance problems is a real thing, so before you start optimizing, make sure you are familiar with React Profiler (Figure 1).

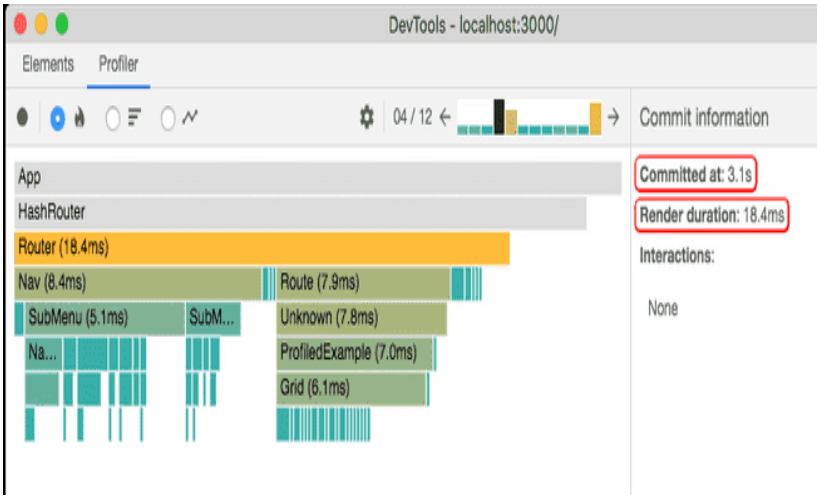



Fig.1. React Profiler: Flame Chart

If you've identi ed scenarios where rendering is slow, memoization is probably the best bet.

React.memo is a performance optimization tool, a higher order component. It's similar to React.PureComponent but for function components instead of classes. If your function component renders the same result given the same props, React will memoize, skip rendering the component, and reuse the last rendered result.

By default it will only shallowly compare complex objects in the props object. If you want control over the comparison, you can also provide a custom comparison function as the second argument.

- **No memoization**

Let's consider an example where we don't use memoization and why that might cause performance problems.

```
function List({ items }) {
  log('renderList');
  return items.map((item, key) => (
    <div key={key}>item: {item.text}</div>
  ));
}
export default function App() {
  log('renderApp');
  const [count, setCount] = useState(0);
  const [items, setItems] = useState(getInitialItems(10));
  return (
    <div>
```

```
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>
        inc
      </button>
      <List items={items} />
    </div>
  );
}
```

Example 1: Live Demo

Every time inc is clicked, both renderApp and renderList are logged, even though nothing has changed for List . If the tree is big enough, it can easily become performance bottleneck. We need to reduce number of renders.

- **Simple memoization**

```
const List = React.memo(({ items }) => {
  log('renderList');
  return items.map((item, key) => (
    <div key={key}>item: {item.text}</div>
  ));
});
export default function App() {
  log('renderApp');
  const [count, setCount] = useState(0);
  const [items, setItems] = useState(getInitialItems(10));
  return (
    <div>
      <h1>{count}</h1>
      <button onClick={() => setCount(count + 1)}>
        inc
      </button>
```

```
    <List items={items} />
   </div>
  );
}
```

Example 2: Live Demo

In this example **memoization works** properly and reduces number of renders. During mount renderApp and renderList are logged, but when inc is clicked, only renderApp is logged.

- **Memoization & callback**

Let's make a small modification and add inc button to all List items. Beware, passing callback to memoized component can cause subtle bugs.

```
function App() {
log('renderApp');

const [count, setCount] = useState(0);
const [items, setItems] = useState(getInitialItems(10));

return (
<div>
<div style={{ display: 'flex' }}>
<h1>{count}</h1>
<button onClick={() => setCount(count + 1)}>
inc
</button>
</div>
<List
items={items}
```

```
inc={() => setCount(count + 1)}
/>
</div>
);
}
```

**Example 3:** Live Demo

In this example, our **memoization fails**. Since we are using inline lambda, new reference is created for each render, making React.memo useless. We need a way to memoize the function itself, before we can memoize the component.

- **useCallback**

Luckily, React has two built-in hooks for that purpose: useMemo and useCallback. useMemo is useful for expensive calculations, useCallback is useful for passing callbacks needed for optimized child components.

```
function App() {
log('renderApp');

const [count, setCount] = useState(0);
const [items, setItems] = useState(getInitialItems(10));

const inc = useCallback(() => setCount(count + 1));

return (
<div>
<div style={{ display: 'flex' }}>
<h1>{count}</h1>
<button onClick={inc}>inc</button>
```

```
</div>
<List items={items} inc={inc} />
</div>
);
}
```

**Example 4:** Live Demo

In this example, our **memoization fails** again.renderList is called every time inc is pressed. Default behavior of useCallback is to compute new value whenever new function instance is passed. Since inline lambdas create new instance during every render, useCallback with default config is useless here.

- **useCallback with input**

```
const inc = useCallback(() => setCount(count + 1), [count]);
```

**Example 5:** Live Demo

useCallback takes second argument, an array of inputs and only if those inputs change will useCallback return new value. In this example, useCallback will return new reference every time count changes. Since count changes during each render, useCallback will return new value during each render. This code **does not memoize** as well.

- **useCallback with input of empty array**

```
const inc = useCallback(() => setCount(count + 1), []);
```

**Example 6:** Live Demo

useCallback can take an empty array as input, which will call inner lambda only once and memoize the reference for future calls. This code **does memoize**, one renderApp will be called when clicking any button, main inc button will work correctly, but inner inc buttons will **stop working** correctly.

Counter will increment from 0 to 1 and it will stop after that. Lambda is created once, but called multiple times. Since count is 0 when lambda is created, it behaves exactly as the code below:

```
const inc = useCallback(() => setCount(1), []);
```

The root cause of our problem is that we are trying to read and write from and to the state at the same time. We need API designed for that purpose. Fortunately for us, React provides two ways for solving the problem:

- **useState with functional updates**

```
const inc = useCallback(() => setCount(c => c + 1), []);
```

**Example 7:** Live Demo

Setters returned by useState can take function as an argument, where you can read previous value of a given state. In this example, **memoization works** correctly, without any bugs.

- useReducer

```
const [count, dispatch] = useReducer(c => c + 1, 0);
```

**Example 8:** Live Demo

useReducer **memoization works** exactly as useState in this case. Since dispatch is guaranteed to have same reference across renders, useCallback is not needed, which makes code less error-prone to memoization related bugs.

- **useReducer vs useState**

useReducer is more suited for managing state objects that contain multiple sub-values or when the next state depends on the previous one. Common pattern of using useReducer is with useContext to avoid explicitly passing callbacks in a large component tree.

The rule of thumb I recommend is to mostly use useState for data that does not leave the component, but if non-trivial two-way data exchange is needed between parent and descendants, useReducer is a better choice.

**Summary**

To sum things up, React.memo and useReducer are best friends, React.memo and useState are siblings that sometimes fight and cause problems, useCallback is the next-door neighbor you should always be cautious about.

## THE ICEBERG OF REACT HOOKS

https://medium.com/@sdolidze/the-iceberg-of-react-hooks-af0b588f43fb

Sandro Dolidze. Mar 26, 2019

React Hooks, unlike Class Components, provide low-level building blocks for optimizing and composing applications with minimal boilerplate.

Without in-depth knowledge, performance problems can arise and code complexity can increase due to subtle bugs and leaky abstractions.

I've created 12 part case study to demonstrate common problems and ways to x them. I've also compiled React Hooks Radar and React Hooks Checklist for small recommendations and quick reference.

. . .

## ➢ Case Study: Implementing Interval

The goal is to implement counter that starts from 0 and increases every 500ms . Three control buttons should be provided: start, stop and clear .


### ❖ Level 0: Hello World

```
export default function Level00() {
 console.log('renderLevel00');
 const [count, setCount] = useState(0);
 return (
  <div>
   count => {count}
   <button onClick={() => setCount(count + 1)}>+</button>
   <button onClick={() => setCount(count - 1)}>-</button>
  </div>
 );
}
```

This is a simple, **correctly implemented** counter, which increments or decrements on user click.

### ❖ Level 1: setInterval

```
export default function Level01() {
 console.log('renderLevel01');

 const [count, setCount] = useState(0);
```

```
  setInterval(() => {
    setCount(count + 1);
  }, 500);
  return <div>count => {count}</div>;
}
```

Intention of this code is to increase counter every *500ms*. This code has a huge **resource leak** and is **implemented incorrectly.** It will easily crash browser tab. Since *Level01* function is called every time render happens, this component creates new interval every time render is triggered.

Mutations, subscriptions, timers, logging, and other side effects are not allowed inside the main body of a function component (referred to as React's render phase). Doing so will lead to confusing bugs and inconsistencies in the UI.

📎 Hooks API Reference: useEffect

❖ **Level 2: useEffect**

```
export default function Level02() {
  console.log('renderLevel02');

  const [count, setCount] = useState(0);

  useEffect(() => {
    setInterval(() => {
      setCount(count + 1);
    }, 500);
  });

  return <div>Level 2: count => {count}</div>;
}
```

Most side-effects happen inside *useEffect.* This code also has a huge **resource leak** and is **implemented incorrectly**. Default behavior of *useEffect* is to run after every render, so new interval will be created every time count changes.

Hooks API Reference: useEffect, Timing of Effects.

❖ **Level 3: run only once**

```
export default function Level03() {
    console.log('renderLevel03');

    const [count, setCount] = useState(0);

    useEffect(() => {
        setInterval(() => {
            setCount(count + 1);
        }, 300);
    }, []);
  return <div>count => {count}</div>;
}
```

Giving *[]* as second argument to *useEffect* will call function once, after *mount.* Even though *setInterval* is called only once, this code is **implemented incorrectly**.

*count* will increase from *0* to *1* and stay that way. Arrow function will be created once and when that happens, *count* will be *0*.

This code has **subtle resource leak**. Even after component unmounts, *setCount* will still be called.

📎 Hooks API Reference: useEffect, Conditionally firing an effect.

❖ **Level 4: cleanup**

```
useEffect(() => {
   const interval = setInterval(() => {
       setCount(count + 1);
    }, 300);
    return () => clearInterval(interval);
}, []);
```

To prevent **resource leaks**, everything must be disposed when lifecycle of a hook ends. In this case returned function will be called after component unmounts.

This code **does not have resource leaks**, but is **implemented incorrectly,** just like previous one.

📎 Hooks API Reference: Cleaning up an effect.

❖ **Level 5: use count as dependency**

```
useEffect(() => {
    const interval = setInterval(() => {
       setCount(count + 1);
    }, 500);
    return () => clearInterval(interval);
}, [count]);
```

Giving array of *dependencies* to *useEffect* will change its lifecycle. In this example *useEffect* will be called once after *mount*

and every time *count* changes. Cleanup function will be called every time count changes to dispose previous resource.

This code **works correctly**, without any bugs, but it's slightly **misleading**. *setInterval* is created and disposed every *500ms*. Each setInterval is always called once.

📎 Hooks API Reference: useEffect, Conditionally firing an effect.

❖ **Level 6: setTimeout**

```
useEffect(() => {
   const timeout = setTimeout(() => {
      setCount(count + 1);
   }, 500);
   return () => clearTimeout(timeout);
}, [count]);
```

This code and the code above work correctly. Since *useEffect* is called every time *count* changes, using *setTimeout* has same effect as calling *setInterval*.

This example is inefficient, new *setTimeout* is created every time render happens. *React* has a better way for fixing the problem.

❖ **Level 7: functional updates for useState**

```
useEffect(() => {
   const interval = setInterval(() => {
      setCount(c => c + 1);
   }, 500);
```

```
    return () => clearInterval(interval);
}, []);
```

In previous example we ran *useEffect* on each *count* change. The was necessary because we needed to have always up-to-date current value.

*useState* provides API to update previous state without capturing the current value. To do that, all we need to do is provide lambda to *setState*.

This code **works correctly** and more efficiently. We are using a single *setInterval* during the lifecycle of a component. *clearInterval* will only be called once after component is **unmounted.**

Hooks API Reference: useState, Functional updates.

❖ **Level 8: local variable**

```
export default function Level08() {
  console.log('renderLevel08');
  const [count, setCount] = useState(0);
  let interval = null;

  const start = () => {
      interval = setInterval(() => {
          setCount(c => c + 1);
      }, 500);
  };

  const stop = () => {
      clearInterval(interval);
  };
```

```
    return (
      <div>
        count => {count}
        <button onClick={start}>start</button>
        <button onClick={stop}>stop</button>
      </div>
    );
}
```

We've added *start* and *stop* buttons. This code is implemented incorrectly, *stop* button does not work. New reference is created during each *render*, so *stop* will have reference to *null.*

📎 Hooks API Reference: Is there something like instance variables?

❖ **Level 9: useRef**

```
export default function Level09() {
  console.log('renderLevel09');
  const [count, setCount] = useState(0);

  const intervalRef = useRef(null);

  const start = () => {
    intervalRef.current = setInterval(() => {
        setCount(c => c + 1);
    }, 500);
  };

  const stop = () => {
    clearInterval(intervalRef.current);
  };
```

```
    return (
      <div>
          count => {count}
          <button onClick={start}>start</button>
          <button onClick={stop}>stop</button>
      </div>
    );
}
```

*useRef* is the go-to hook if mutable variable is needed. Unlike local variables, *React* makes sure same reference is returned during each render.

This code seems correct, but has a **subtle bug**. If *start* is called multiple times, *setInterval* will be called multiple times triggering resource leak.

📎 Hooks API Reference: useRef

❖ **Level 10: useCallback**

```
export default function Level10() {
 console.log('renderLevel10');

 const [count, setCount] = useState(0);

 const intervalRef = useRef(null);

 const start = () => {
     if (intervalRef.current !== null) {
         return;
     }
```

```
      intervalRef.current = setInterval(() => {
          setCount(c => c + 1);
        }, 500);
    };

    const stop = ( ) => {
        if (intervalRef.current === null) {
     return;
   }

   clearInterval(intervalRef.current);
   intervalRef.current = null;
 };

 return (
   <div>
       count => {count}
       <button onClick={start}>start</button>
       <button onClick={stop}>stop</button>
   </div>
 );
}
```

To avoid resource leak, we simply ignore calls if *interval* is already started. Although calling *clearInterval(null)* does not trigger any errors, it's still good practice to dispose resource only once.

This code has **no resource leaks**, is **implemented correctly**, but might have a **performance problem**.

*memoization* is main performance optimization tool in *React*. *React.memo* does shallow comparison and if references are same, render is skipped.

If *start* and *stop* were passed to a *memoized* component, the whole optimization would fail, because new reference is returned after each render.

## 🖇 React Hooks: Memoization

❖ **Level 11: useCallback**

```
export default function Level11() {
  console.log('renderLevel11');

  const [count, setCount] = useState(0);

  const intervalRef = useRef(null);

  const start = useCallback(() => {
    if (intervalRef.current !== null) {
      return;
    }

    intervalRef.current = setInterval(() => {
      setCount(c => c + 1);
    }, 500);
  }, []);

 const stop = useCallback(() => {
    if (intervalRef.current === null) {
      return;
    }

    clearInterval(intervalRef.current);
    intervalRef.current = null;
  }, []);
```

```
return (
   <div>
      count => {count}
      <button onClick={start}>start</button>
      <button onClick={stop}>stop</button>
   </div>
  );
}
```

To enable *React.memo* to do its job properly, all we need to do it to *memoize* functions, using useCallback hook. This way, same reference will be provided after each render.

This code has **no resource leaks**, is **implemented correctly**, has **no performance problem**, but code is **quite complex**, even for a simple counter.

Hooks API Reference: useCallback

❖ **Level 12: custom hook**

```
function useCounter(initialValue, ms) {
  const [count, setCount] = useState(initialValue);
  const intervalRef = useRef(null);

  const start = useCallback(() => {
    if (intervalRef.current !== null) {
      return;
    }
    intervalRef.current = setInterval(() => {
      setCount(c => c + 1);
    }, ms);
  }, []);
```

```
  const stop = useCallback(() => {
      if (intervalRef.current === null) {
        return;
      }

      clearInterval(intervalRef.current);
      intervalRef.current = null;
  }, []);

  const reset = useCallback(() => {
      setCount(0);
  }, []);

  return { count, start, stop, reset };
}
```

To **simplify code**, we need to encapsulate all complexity inside *useCounter* custom hook and expose clean api: { *count, start, stop, reset* } .

```
export default function Level12() {
  console.log('renderLevel12');

  const { count, start, stop, reset } = useCounter(0, 500);

  return (
    <div>
        count => {count}
        <button onClick={start}>start</button>
        <button onClick={stop}>stop</button>
        <button onClick={reset}>reset</button>
    </div>
  );
}
```

🔗 Hooks API Reference: Using a Custom Hook

## ➢ React Hooks Radar

All *React Hooks* are equal, but some hooks are **more equal** than others.

### ✅ Green

Green hooks are main building blocks of modern *React* applications. They are safe to use almost everywhere without much thinking.

1. useReducer
2. useState
3. useContext

### 🛞 Yellow

Yellow hooks provide useful performance optimizations by using memoization. Managing lifecycle and inputs should be done with caution.

1. useCallback
2. useMemo

### ⚫ Red

Red hooks interact with mutable world using side effects. They are most powerful and should be used with extreme caution. Customs hooks are recommended for all non-trivial use-cases.

1. useRef
2. useEffect
3. useLayoutEffect

## ➢React Hooks Checklist

1. Obey Rules of Hooks.
2. Don’t do any side-effects in main render function.
3. Unsubscribe/dispose/destroy all used resources.
4. Prefer useReducer or functional updates for useState to prevent reading and writing same value in a hook.
5. Don’t use mutable variables inside render function, use useRef instead.

6. If what you save in useRef has smaller lifecycle than the component itself, don't forget to unset the value when disposing the resource.
7. Be cautions with infinite recursion and resource starvation.
8. Memoize functions and objects when needed to improve performance.
9. Correctly capture input dependencies (undefined => every render, [a, b] => when a or b change, [] => only once).
10. Use customs hooks for non-trivial use-cases.

გადაეცა წარმოებას 28.03.2019. ხელმოწერილია დასაბეჭდად 1.04.2019. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 6. ტირაჟი 100 ეგზ.

სტუ-ს „IT კონსალტინგის ცენტრი“

(თბილისი, მ.კოსტავას 77)

ISBN 978-9941-8-0625-4