დიზაინის თეორიის დაცვა თანამედროვე ფრონტ-ენდ დეველოპმენტში

გიორგი ბერიძე
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

განხილულია დიზაინის თეორია, ე.წ. **DRY** პრინციპი, თანამედროვე ფრონტ-ენდ დეველოპმენტში. ეს არის მეთოდი, რომლის მოშვეობით ხდება პროექტში არსებული ფუნციების ან სხვა დიზაინერული სტილების დაყოფა გარკვეულ კომპონენტებად და მოდულებად, რომლითაც ხდება მათი მრავალჯერადი გამოყენება და დუბლირებული კოდის თავიდან აცილება. ნაშრომში შემოთავაზებულია პროგრამული კოდის „სისუფთავისა“ და მისი სტრუქტურის დაცვის პრინციპები კონკრეტული მაგალითების საფუძვლებზე.

საკვანძო სიტყვები: პროგრამირება. დუბლირებული კოდი. კომპონენტი. ტესტირება. Dry-პრინციპი.

1. შესავალი

Dry პრინციპის გამოყენება კოდში უკვე დიდი ხანია მიღებულია როგორც ერთ ერთი ყველაზე ცნობილი სტანდარტი ვებ ინდუსტრიაში. რა არის ეს პრინციპი და რატომ გვჭირდება მისი გამოყენება თანამედროვე ვებ სივრცეში? ინგლისურიდან, სიტყვა სიტყვით რომ ვთარგმნოთ, ნიშნავს „არ გაიმეორო საკუთარი თავი“ (**D**on’t **R**epeat **Y**ourself). ეს არის პროგრამული უზრუნველყოფის აგების პრინციპი, რომელიც ფორმულირებულია *ა. ჰანტის (Andrew Hunt) და დ. ტომასის (David Thomas)* წიგნში (1964) *„პრაგმატული პროგრამისტი*” [1]. Dry კოდის ფილოსოფია მდგომარეობს იმაში, რომ პროექტში არსებული მრავალჯედარად გამოყენებად ფუნქციების წარმოდგენა უნდა მოხდეს როგორც ცალკეული კომპონენტი, რათა თავიდან ავიცილოთ მათი დუბლირება. ყოველ პროგრამისტს აქვს თავისი დიზაინის პრინციპი, რომელსაც ყოველღიურად იყენებს სხვადასხვა პროგრამულ ენებში. ეს მას ეხმარება ხარისხის, პროდუქტიულობის და კომფორტულობის გაუმჯობესებაში.

DRY პრინციპზე დაყრდნობით ხდება კოდის პროდუქტიულობის გაზრდა, კოდში ცვლილებების მარტივად შეტანა და ტესტირების ხარისხის გაუმჯობესება. პროგრამისტის ყველაზე დიდი პრობლემაა ლოგიკის ცვლილებები კოდში, თუმცა თუ არადუბლირებული კოდის შემთხვევაში ეს პრობლემა შეიძლება გადაჭრილად ჩაითვალოს, რადგან DRY ყოველთვის ახდენს რთული ალგორითმების დეკომპოზიციას მარტივ ფუნქციებად, ეს კი საგრძნობლად ამარტივებს პროგრამულ კოდს. ფუნქციების განმეორებადი გამოყენება კი უფლებას იძლევა შემცირდეს პროგრამირების დრო და ახალი ფუნქციონალის ტესტირება მარტივად და „უმტკივნეულოდ“.

DRY პრინციპებზე დაყრდნობას მივყავართ აპლიკაციების მოდულირებად არქიტექტურამდე, რითაც მარტივად ხდება ბიზნესს ლოგიკისა და პროგრამული კლასების გამორჩევა. სირთულის ყველაზე მარტივი მიდგომად ითვლება სისტემის დაყოფა მართვად ნაწილებად (კომპონენტებად). მაგალითად: ნებისმიერი CMS-ის ნაწილი არის კომპონენტი,

რომელიც კურირებს მომხმარებლების მართვაზე. ეს კომპონენტი შეიძლება დაყოფილი იყოს კიდევ რამდენიმე ქვეკომპონენტად, მაგალითად მომხმარებლის უფლებებზე წვდომა, რომელიც მუშაობს სისტემის სხვა უსაფრთხოების ქვესისტემასთან. სისტემის ასე ღრმად დაყოფით კი საბოლოოდ მივდივართ იმ მომენტამდე, როცა სისტემის ყოველი ნაწილი პასუხს აგებს მკაფიოდ განსაზღვრულ მოქმედებაზე. სწორედ ეს არის DRY პრინციპი – განმეორებადი კოდის თავიდან აცილება და მათი მრავალჯერადი გამოყენება.

DRY პრინციპზეა ასევე აგებული ისეთი პროგრამირების პარადიგმები როგორიცაა, *სტრუქტურული პროგრამირება* – პროგრამირების სტილი, რომელშიაც პროგრამა წარმოდგენილია იერარქიულად დაქვემდებარებული ბლოკების სტრუქტურის სახით და *ობიექტზე ორიენტირებული პროგრამირება* - ოოპ (ინგლისურენოვანი შემოკლებით OOP) - პროგრამირების პარადიგმა, რომელისთვისაც ცენტრალურია არა მოქმედების, არამედ ობიექტის ცნება. ობიექტი შეიძლება შეიცავდეს ინფორმაციას, რომელიც ინახება ობიექტის ველებში (ასევე ცნობილია როგორც ობიექტის ატრიბუტები); კოდი წარმოდგენილია პროცედურების, ან როგორც ხშირად მოიხსენიებენ, მეთოდების სახით. ობიექტებს აქვს უნარი შეცვალოს ინფორმაცია ობიექტის ველებში პროცედურების (მეთოდების) საშუალებით. ოოპ-ში კომპიუტერული პროგრამები იქმნება სხვადასხვა ობიექტებით და ამ ობიექტების ერთმანეთთან ურთიერთქმედებით.

DRY – ეს არის ფილოსოფია, რომელიც ლოგიკას გარდაქმნის წარმოდგენად. DRY მოითხოვს კარგ გეგმას. ყოველი პროექტის არქიტექტურის სისტემა უნდა დაიყოს მარტივ ლოგიკურ ნაწილებად. არსებობს ამ ნაწილების ორი ძირითადი წესი:

- უნდა შეიქმნას სისტემის გრაფიკული სქემა და დაიყოს ვიზულარი კომპონენტებად;
- პროექტის დაწყებამდე სქემაში უნდა მოინიშნოს ის ნაწილი, რომელზეც იწყება მუშაობა.

Dry პრინციპზე დაყრდნობით დღეს უკვე აგებულია თითქმის ყველა თანამედროვე ბიბლიოთეკა თუ ფრეიმვორკი. აღსანიშნავია რომ ყოველ პროგრამულ ენას გააჩნია თავისი ბიბლიოთეკა ან ფრეიმვორკი, რომლეთა მეშვეობით ხდება სინტაქსის გამარტივება და უფრო პროფესიონალურ დონემდე დაყვანა.

2. ძირითადი ნაწილი

არსებობს ენები რომელთა გარეშეც წარმოუდგენელია თანამედროვე ვებ საიტები და ვებ აპლიკაციები, ესეთ ენებს წარმოადგენს html, css და javascript-ი, მაგრამ ვიცით რომ მათი სინტაქსი არ ახდენს იმ პრინციპის დაცვას რის განხილვასაც ვცდილობთ ჩვენ, ამიტომაც დროთა განმავლობაში შეიქმნა მათი ე.წ. ფრეიმვორკები/ბიბლიოთეკები, რისი დახმარებითაც მოხდა dry პრინციპის წესების დაცვა [2]. განვიხილოთ თითოეული მათგანის ყველაზე პოპულარული ფრეიმვორკები/ბიბლიოთეკები და მაგალითების მიხედვით დავინახავთ თუ რაოდენ ძლიერი და მოქნილია მათი გამოყენება.

PUGJS – HTML-სთვის შექმნილი პრეპროცესორი, რომელითაც წარმოდგენა იცვლება HTML კოდის წერაზე [3]. ეს არის ახალი ხერხი – თუ როგორ შემცირდეს HTML კოდის წერის დრო და, რა თქმა უნდა, DRY პრინციპის შენარჩუნება.

HTML

```
<div>
 <h1>Ocean's Eleven</h1>
 <ul>
  <li>Comedy</li>
  <li>Thriller</li>
 </ul>
 <p>Danny Ocean and his eleven accomplices plan to rob three Las Vegas casinos simultaneously.</p>
</div>
```

PUBGJS

```
div
 h1 Ocean's Eleven
 ul
  li Comedy
  li Thriller
 p.
  Danny Ocean and his eleven accomplices plan to rob
  three Las Vegas casinos simultaneously.
```

ამ პრეპროცესორის გამოყენებით ხდება html კოდის უფრო გამარტივება და ტეგების განმეორებადი სინტაქსის თავიდან მოშორება.

3. SASS – css-ის პრეპროცესორი/ბიბლიოთეკა

დაახლოებით 10-15 წლის წინ რომ გვეთქვა css-ში შესაძლებელია ცვლადების გამოყენება და ფუნქციების წერა, ეს იქნებოდა ყველაზე დიდი შეცდომა ვებ სივრცეში. მაგრამ დღესდღეობით არსებობს სტილების შექმნის ახალი ინსტრუმენტები, რომლებიც ამის საშუალებას იძლევიან. ერთ-ერთი და ყველაზე პოპულარული ინსტრუმენტია - SASS [4]. მისი შესაძლებლობების დანახვისათვის განვიხილოთ მაგალითი:

```
$blue: #3bbfce;
$margin: 16px;
.content_navigation {
 border-color: $blue;
 color: darken($blue, 10%);
}
.border {
 padding: $margin / 2;
 margin: $margin / 2;
 border-color: $blue;
}
```

```
@mixin left($dist) {
  float: left;
  margin-left: $dist;
}
```

ამ მაგალითში ჩანს, რომ უკვე css სინტაქსში შესაძლებელია ცვლადების გამოყენება, მათემატიკური გამოთვლები, ფერების პროცენტებით გაზრდა შემცირება და ა.შ.

4. REACTJS – javascript-ის ბიბლიოთეკა

რა თქმა უნდა javascript-თვის არსებობს სხვა მრავალი ბიბლიოთეკა/ფრეიმვორკი, როგორიცაა, angular2, vueJS, knokoutJS, reactJS უფრო მეტი უპირატესობა აქვს სიმარტივისა და კოდის სისუფთავის მხრივ [5]. თუმცა აღსანიშნავია ის რომ ყველა თანამედროვე javascript ბიბლიოთეკები ცდილობენ dry პრინციპზე მორგებას და მისი მეშვეობით ვებ აპლიკაციის კომპონენტებად და მოდულებად დაყოფას.

დავიწყოთ იქედან, რომ ადრე ვებ აპლიკაციის შესაქმნელად გამოიყენებოდა ერთადერთი გზა: მოთხოვნის გაგზავნა სერვერზე და, შემდეგ ლოდინი თუ როდის მივიღებდით პასუხს (პასუხი მზა ვებ გვერდის სახით). ძალიან რთულია იმის გარკვევა თუ რა ხდება ამ დროს ბრაუზერში. ასეთი მიდგომის რეალიზაცია ხდება ისეთ პროგრამულ ენეზე როგორიცაა PHP. მასში ასევე შესაძლებელია ფუნქციონალური კომპონენტების შექმნა და მათი მრავალჯერ გამოყენება. სწორედ ასეთი სიმარტივის გამო php გარდაიქმნა ყველაზე გავრცელებულ პროგრამულ ენად. მაგრამ ასეთი მარტივი აპლიკაციით მუშაობა, ძალიან მოუხერხებელი და მოსაწყენი გამოდგა, რადგან მომხმარებელი ყოველ მოთხოვნაზე ელოდება სერვერს და ამავდროულად კარგავს კონტენტის დიდ ნაწილს მის ბოლომდე ჩამოტვირთვამდე, რაც, რა თქმა უნდა, დროის გარკვეულ ნაწილს მოითხოვს.

ამ პრობლემის გადასაჭრელად სხვადასხვა დეველოპერები ქმნიდნენ სხვადასხვა ბიბლიოთეკებს javascript-ის მეშვეობით, რომლითაც აპლიკაციის რენდერინგი ხდებოდა პირდაპირ ბრაუზერში. გამოიყენებოდა DOM-ელემენტების მანიპულაციის სხვადასხვა ხერხები, დაწყებული მარტივი HTML შაბლონიზატორებიდან, დამთავრებული სისტემებით, რომლებიც აკონტროლებდნენ სრულ აპლიკაციას.

დროთა განმავლობაში ეს ყველაფერი ნელ-ნელა გადრაიქმნა „მონსტრად", რომლებიც თითქოს მუშაობდნენ ისე, როგორც საჭირო იყო, თუმცა ისევ გაუგებარი იყო მისი მუშაობის პრინციპი (PHP-სგან განსხვავებით). ხოლო ეს ყველაფერი შეიცვალა როცა გამოჩნდა ReactJS [4]. საინტერესოა ის რომ reactJS წარმოიქმნა php ფრეიმვორკის ერთ-ერთი პორტისგან (XHP), ასე შეარქვეს მას facebook-ის დეველოპერებმა, რომლებმაც შექმნეს თვით reactjs. React-მა არ იცის ajax მოთხოვნების, როუტინგის და ინფორმაციის შენახვის შესახებ. ის არ არის MVC (Model-View-Controller) [6].

ამ აბრევიატურიდან მასთან კავშირშია მხოლოდ V (view), ანუ reactJS მუშაობს მხოლოდ კლიენტის მხარეს, ამიტომაც არის შესაძლებელი მისი გამოყენება სხვადასხვა

სისტემებში. Javascript-ის ყველაზე დიდი პრობლემაა ვებ გვერდის ყოველ შენახულ მოქმედებაზე, თავიდან გამოჩენა, რაც ძლიერ მოქმედებს ვებ აპლიკაციის სისწრაფეზე, განსხვავებით reactJS-სა, რომელსაც შეუძლია სწრაფი სისტემური რენდერინგი.

Reactjs-ი მუშაობს როგორ 3D თამაშების ძრავა, ის შექმნილია რენდერინგის ფუნქციების ირგვლივ, რომლებიც იღებენ „სამყაროს“ და გარდაქმნიან ვირტუალურად. როცა react-ი გებულობს მოქმედების შეცვლის შესახებ, ის თავიდან აგზავნის ფუნქციას, რომლითაც განსაზღვრავს ვებ გვერდის ვირტუალური DOM-ის მდგომარეობას, შემდეგ ავტომატურად აკეთებს შედეგის ტრანსლაციას უკვე არსებულ DOM-ში და ცვლის მხოლოდ იმას რაც უნდა შეცვლილიყო ვებ აპლიკაციის კონტენტში(მხოლოდ შეცვლილ ელემენტს და არა სრულად ვებ აპლიკაციას) ერთი შეხედვით ეს თითქოს ნელა უნდა მუშაობდეს, განსხვავებით javascript-ისა, მაგრამ react-ს აქვს ჩაშენებული ალგორითმი, რომლითაც ის განსაზღვრავს განსხვავებას მიმდინარე და ახალ გვერდს შორის.

3. დასკვნა

დღესდღეობით პროგრამირებაში პროდუქტიულობის მისაღწევად ძალიან ბევრი ინსტრუმენტი თუ ფრეიმვორკი/ბიბლიოთეკა არსებობს, თუმცა პრობლემა ისევ და ისევ ვლინდება გამართული კოდის ქონაში. ეს ინსტრუმენტები გვიმარტივებენ „ცხოვრებას”, თუმცა მხოლოდ მათი დახმარებით პრობლემის მოგვარება ვერ ხერხდება, რადგანაც უნდა შემუშავდეს სისტემა, რომელიც იძლევა საშუალებას მისი რაიმე პატარა ნაწილის ცვლილების შემთხვევაში არ მოხდეს სხვა ნაწილზე შეხება, ამიტომაც DRY პრინციპი არის ყველაზე კარგი მეთოდი ამის განსახორციელებლად, ხოლო ზემოთ ჩამოთვლილი ინსტრუმენტებით კი უბრალოდ მარტივდება სისტემის სწორად ჩამოყალიბება.

ლიტერატურა - References – Литература:

1. Hunt A., Thomas D. The Pragmatic Programmer, US, 2000
2. https://habr.com/post/144611/ - უკანასკნელად იქნა გადამოწმებული - 27.10.2018
3. https://github.com/pugjs/pug - უკანასკნელად იქნა გადამოწმებული - 12.10.2018
4. https://sass-lang.com/ უკანასკნელად იქნა გადამოწმებული - 29.11.2018
5. Banks A., Learning React: Functional Web Development with React and Redux, US, 2017
6. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller - უკანასკნელად იქნა გადამოწმებული - 26.11.2018

DESIGNING THEORY IN THE MODERN *FRONT-END* DEVELOPMENT

Giorgi Beridze

Georgian Technical University

Summary

The design theory, named by DRY – don't repeat yourself, in modern front-end development. This is a method, what make some functions and design styles make like a separate components or modules, through which can use it for multiple times and avoid duplicate code. In this article are offered the principles of programing code "cleanliness" and its structure based on specific examples.

ТЕОРИЯ ПРОЕКТИРОВАНИЯ В СОВРЕМЕННОМ *FRONT-END* РАЗРАБОТКЕ

Беридзе Г.

Грузинский Технический Университет

Резюме

Обсуждается теория дизайна в современном фронт-енд девелопменте, так называемый принцип DRY. Это метод, который делает некоторые функции и стили дизайна похожими на отдельные компоненты или модули, с помощью которых можно использовать его несколько раз и избежать дублирования кода. В данной статье предлагаются принципы «чистоты» программного кода и его структура на основе конкретных примеров.