Mini Bank of Georgia

# გამოსაყენებელი მასალა:

<https://github.com/levani132/mini-bank-of-georgia>

<https://fomantic-ui.com/introduction/getting-started.html>

<https://app.swaggerhub.com/apis/levani132/mini-bank-of-georgia-api/1.0.1>

<https://bog-angular-course-final.herokuapp.com/>

<https://bog-angular-course-api.herokuapp.com/>

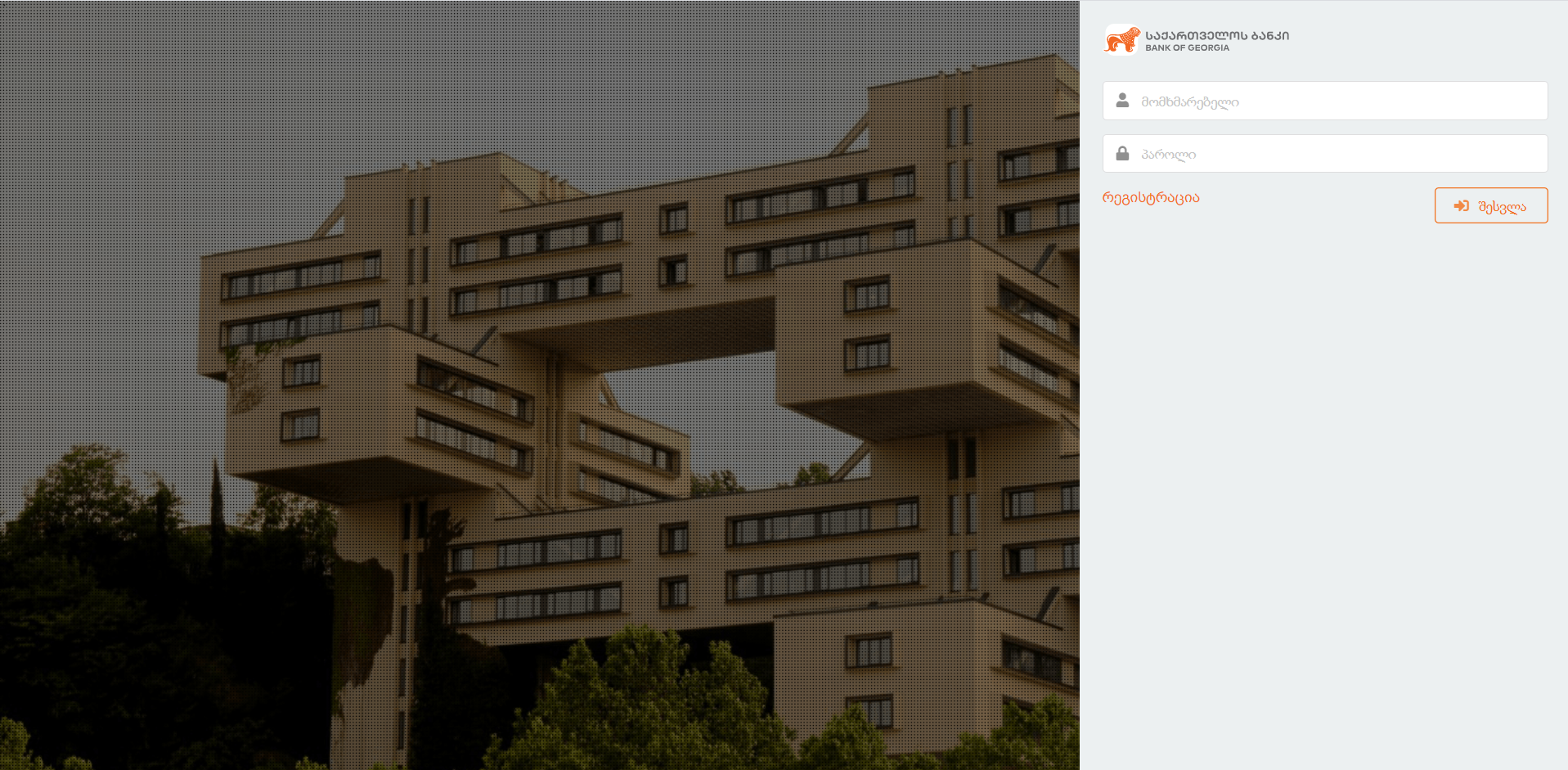
# აღწერა:

აპლიკაცია რომელიც თქვენ უნდა დაწეროთ წარმოადგენს საქართველოს ბანკის core banking სისტემების front-end-ის მინი შედარებით გამარტივებულ და გადაკეთებულ ვარიანტს (ამ აპლიკაციას ხშირად „ვაბანკსაც“ ვუწოდებთ).

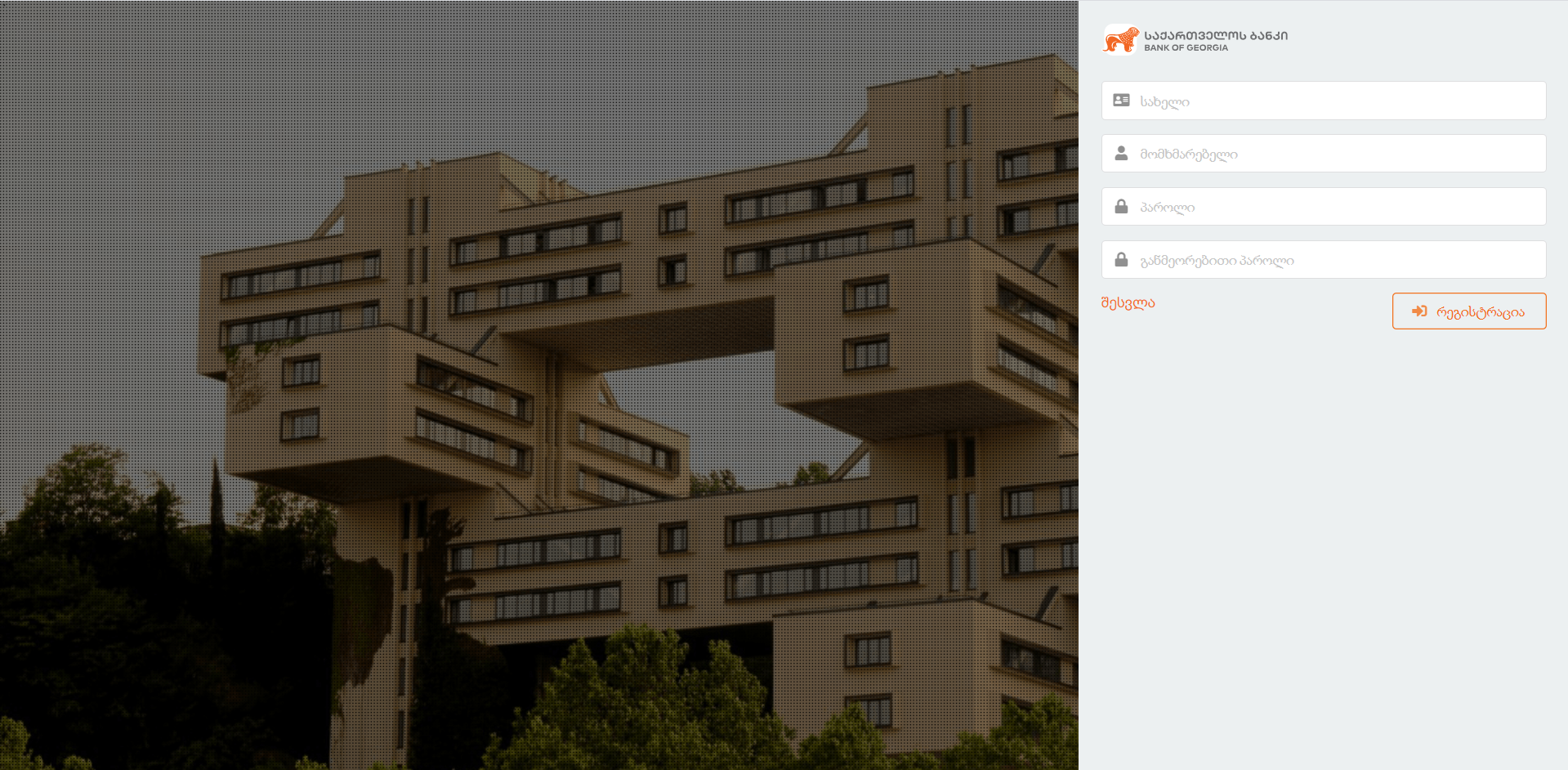
მთელი დიდი აზრი ამ გამარტივებული ვარიანტისა არის ის რომ საიტზე შესაძლებელია მომხმარებლების რეგისტრაცია/ავტორიზაცია. მომხმარებელს შეუძლია მოძებნოს კლიენტი, შექმნას კლიენტი, ნახოს კლიენტის ანგარიშები, შეუქმნას ან წაუშალოს მას ანგარიში და გადარიცხოს კლიენტის ანგარიშებიდან ნებისმიერ სხვა ანგარიშზე თანხა.

# სქრინები/გვერდები:

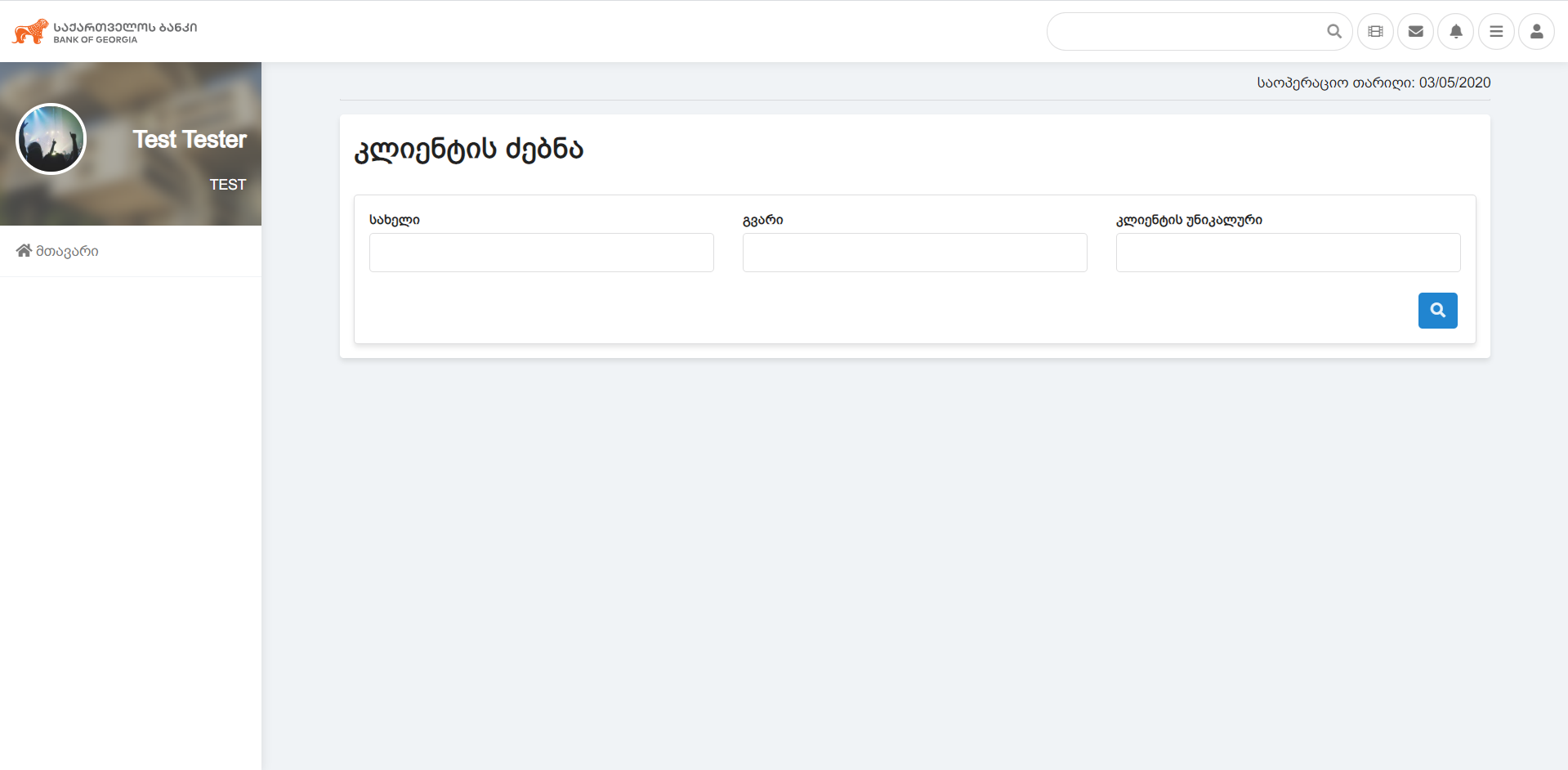
## გვერდი /auth:

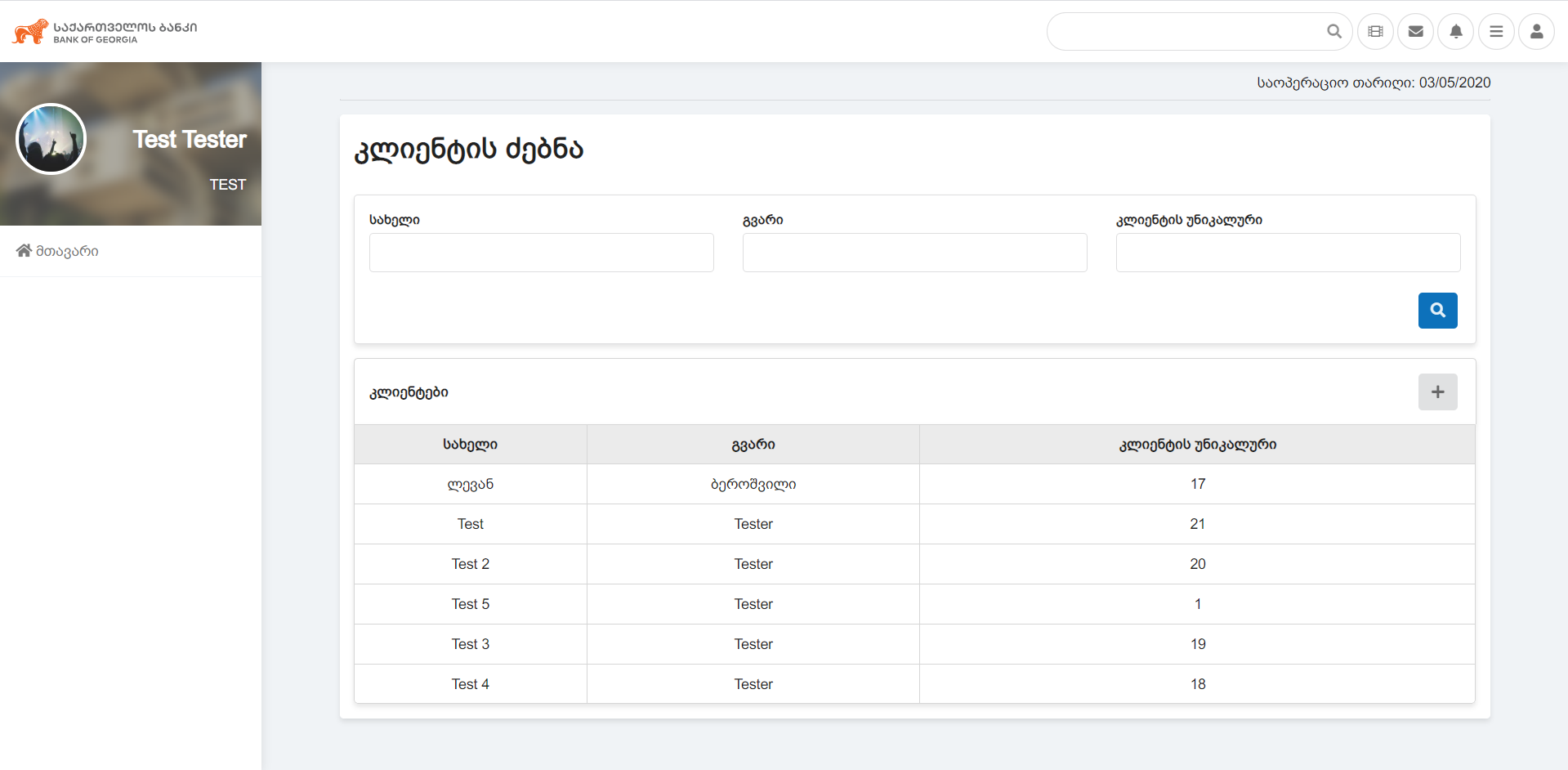


## გვერდი /auth/register:

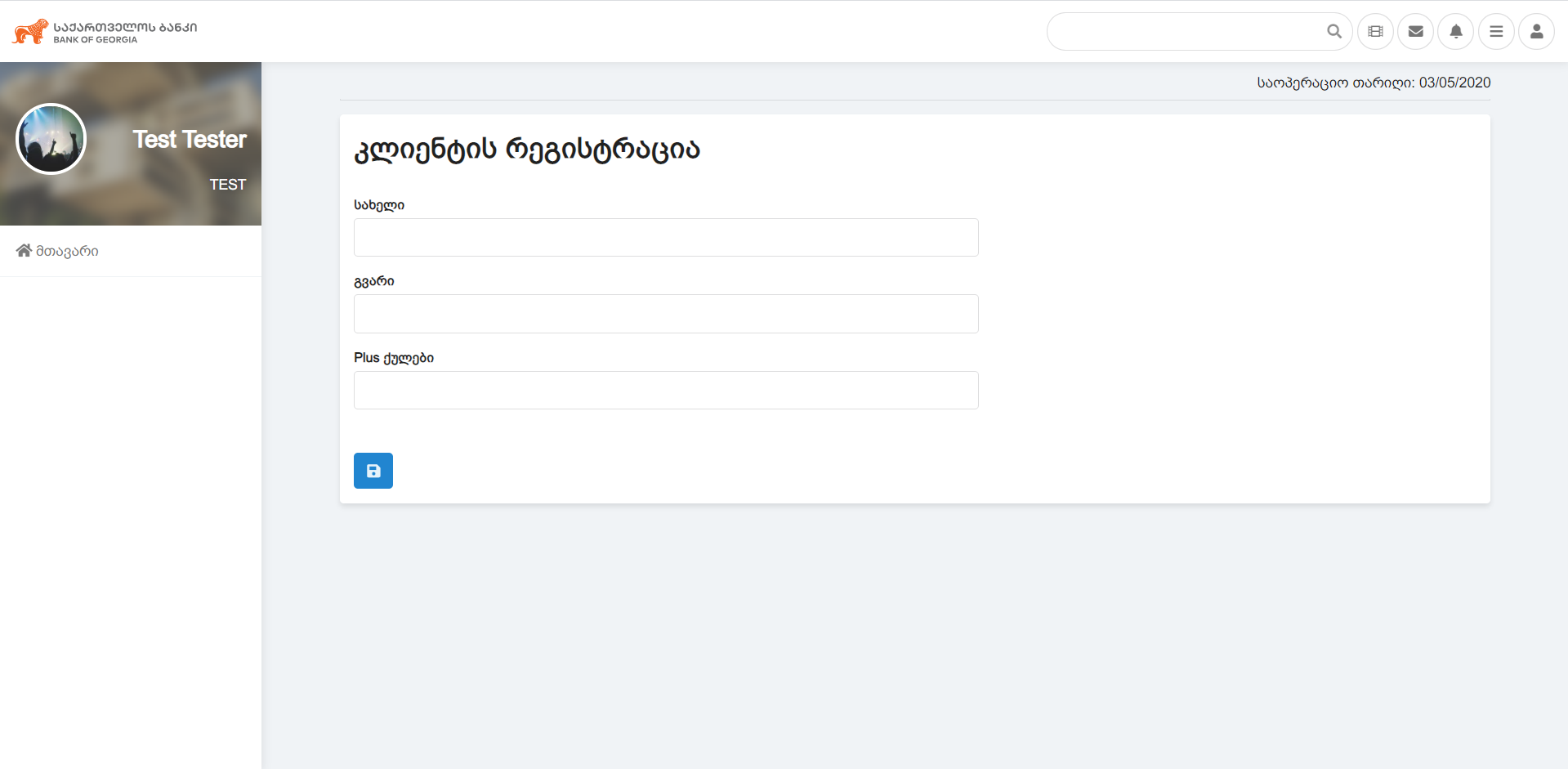


## კლიენტის ძებნა /bpm/bpm000:

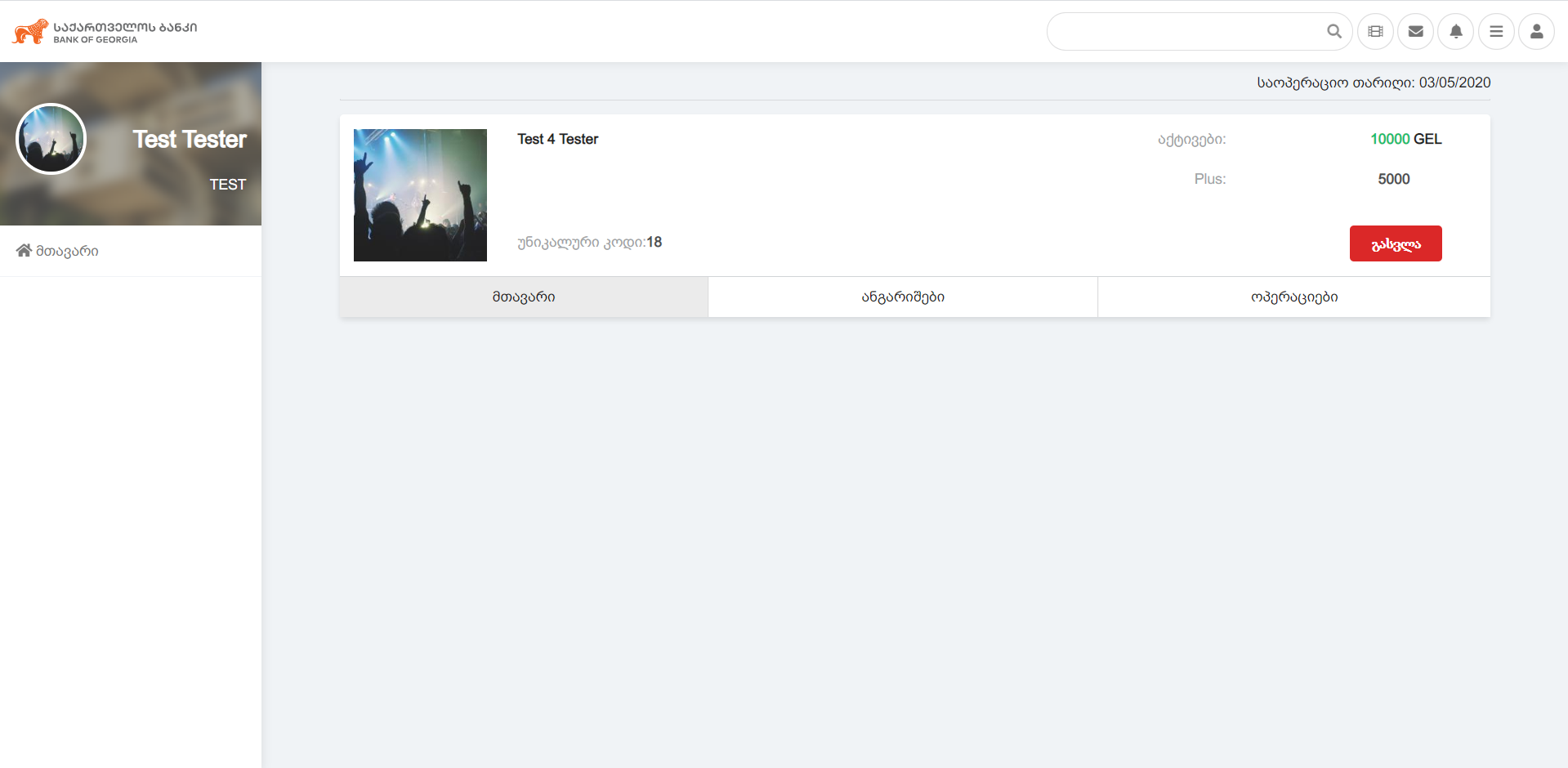




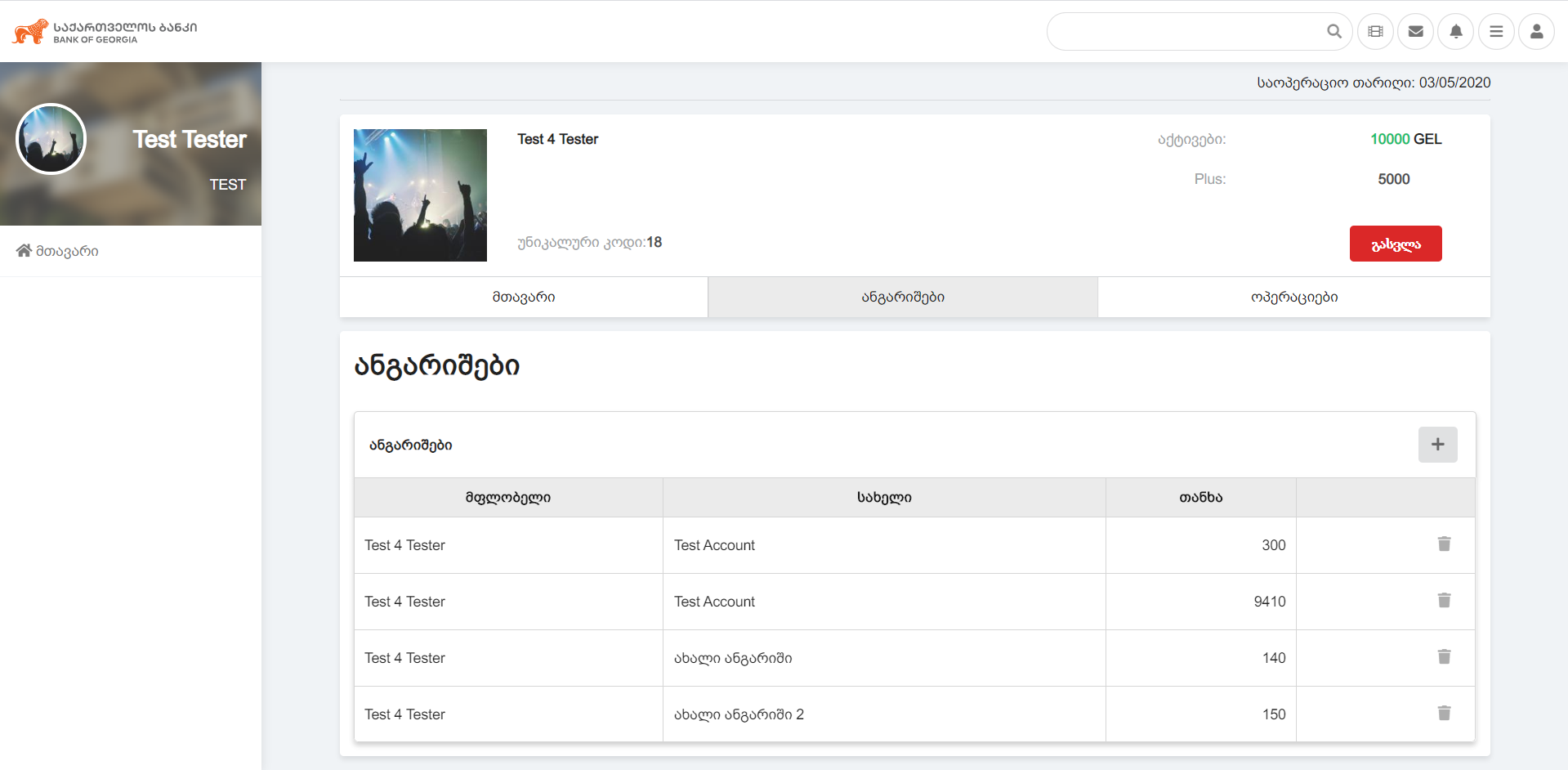
## კლიენტის რეგისტრაცია /bpm/bpm001:



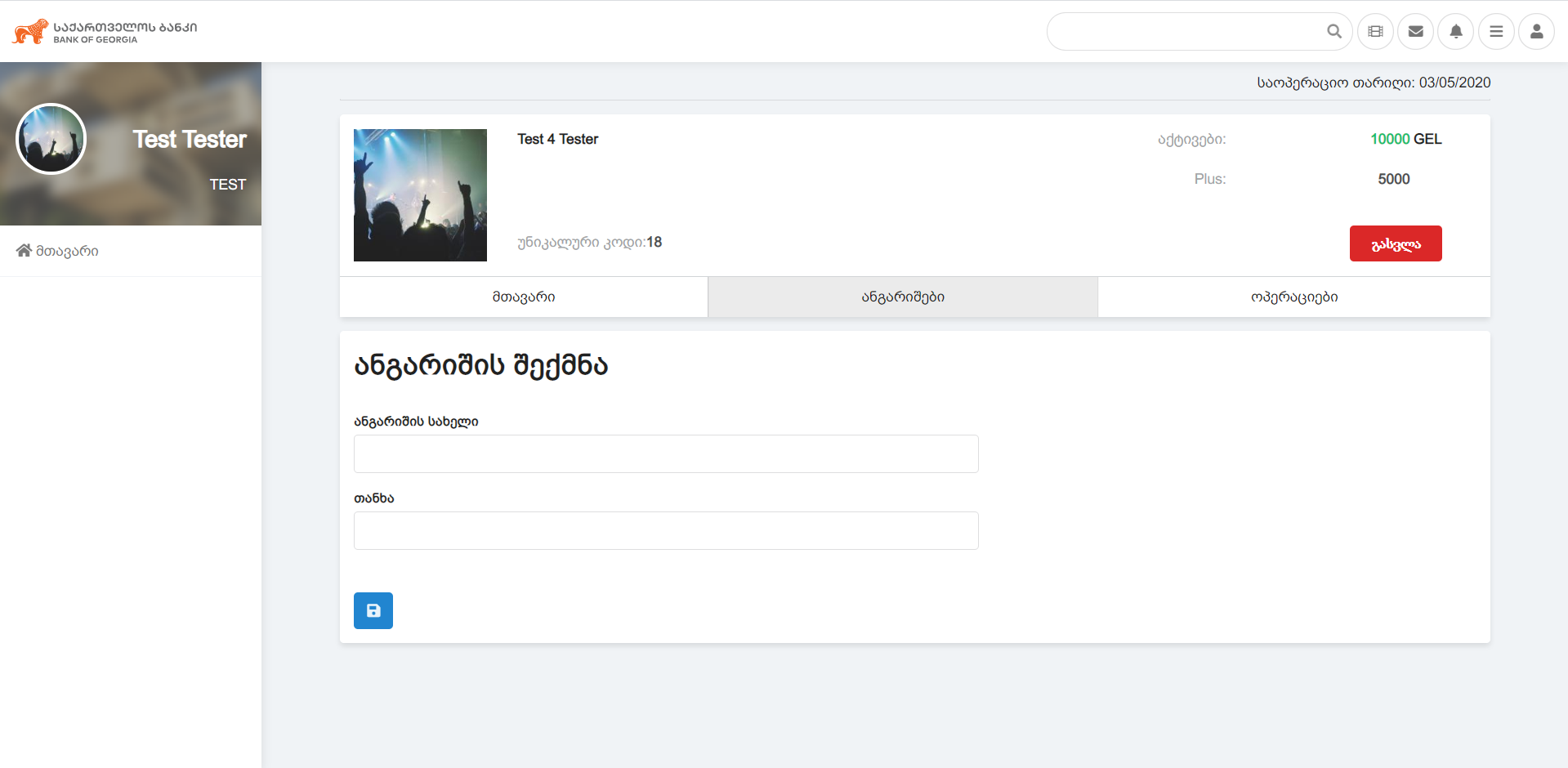
## კლიენტის გვერდი /krn/krnicp:



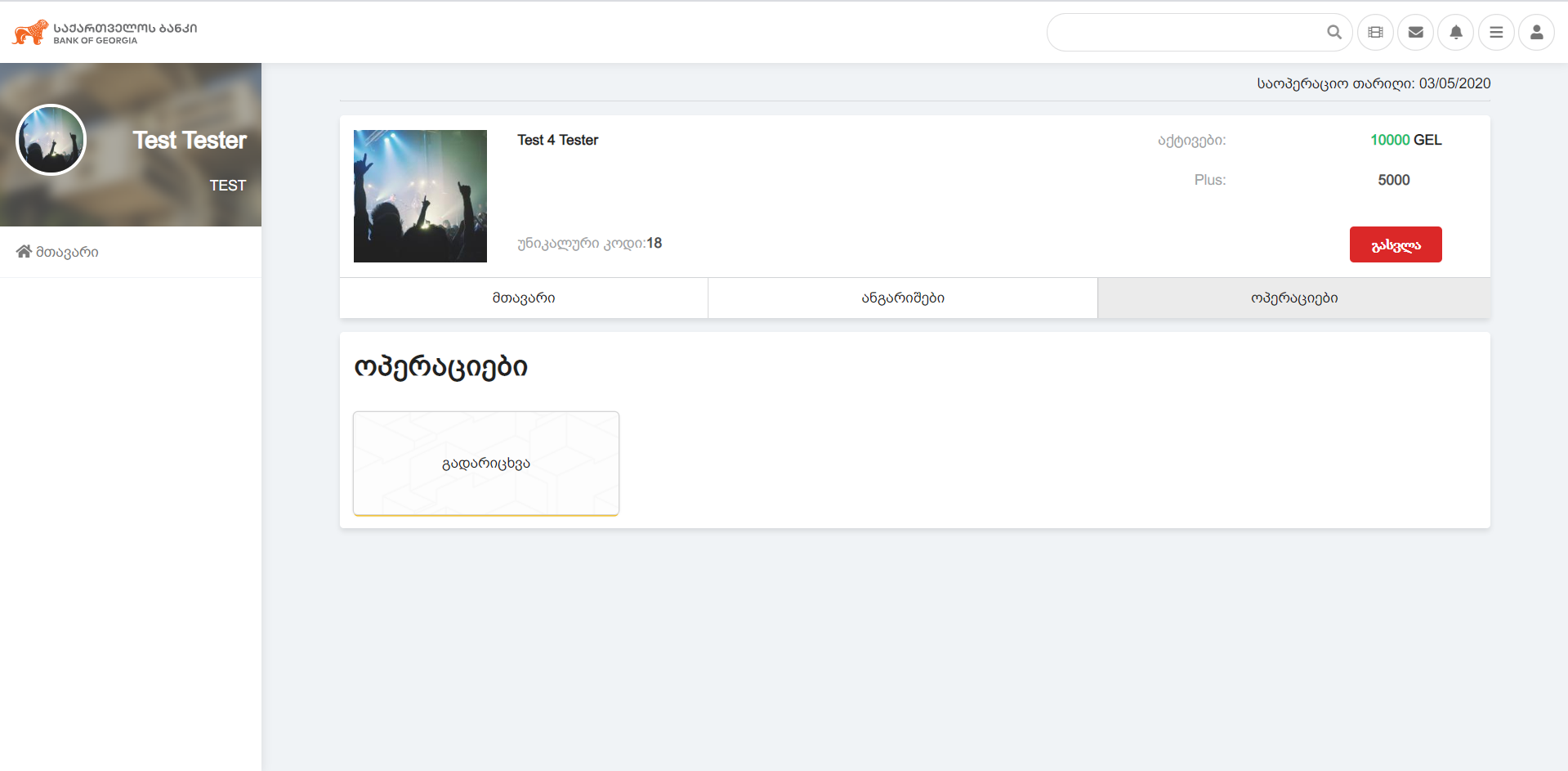
## ანგარიშები /krn/accounts:



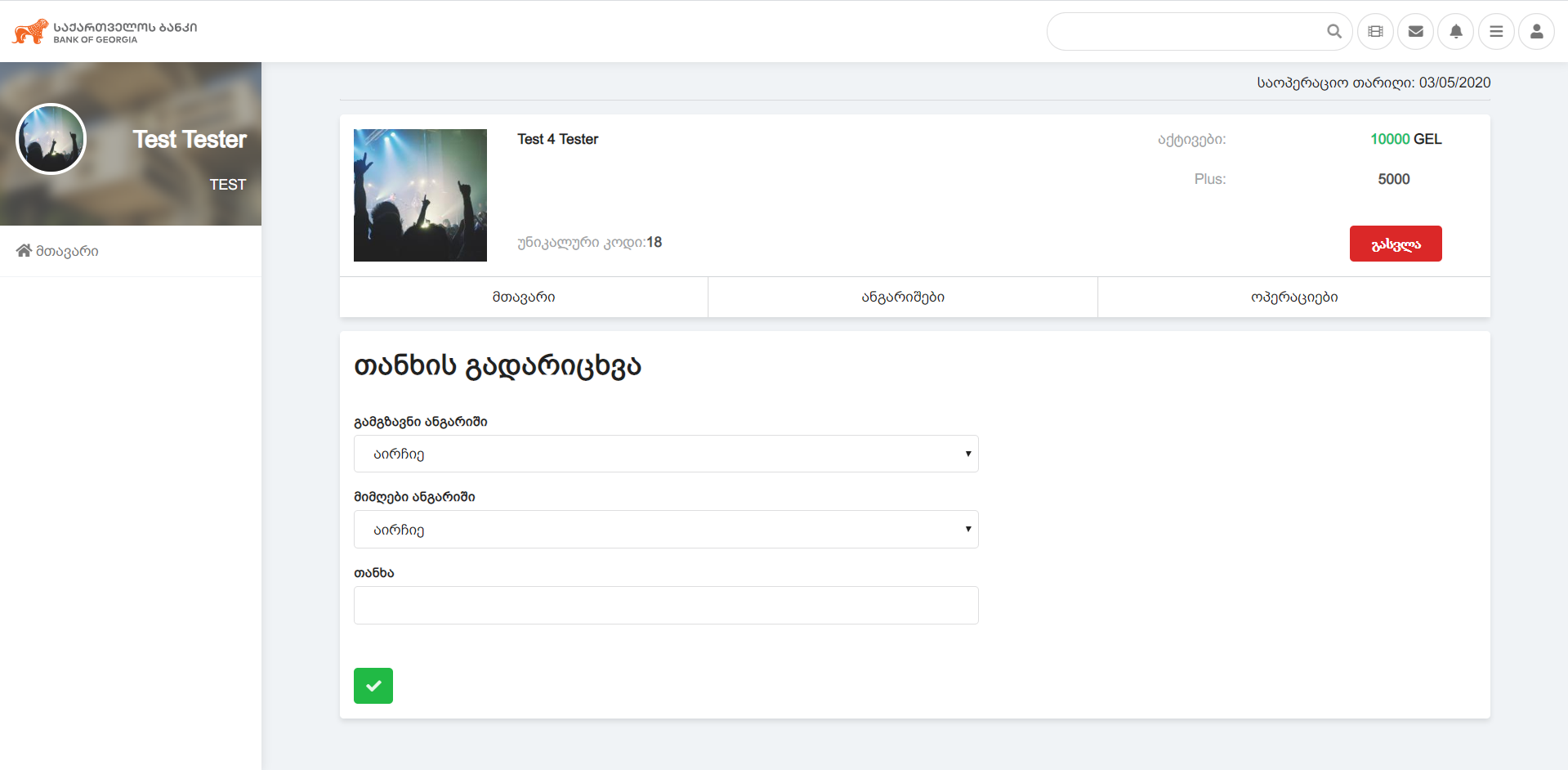
## ანგარიშის შექმნა /krn/accounts/create:



## ოპერაციები /krn/operations:

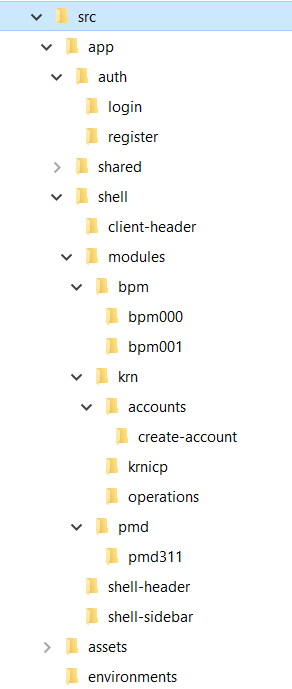


## გადარიცხვა /pmd/pmd311:



# ფოლდერების სტრუქტურა:

საბოლოო ფოლდერების სტრუქტურა წესით უნდა მიიღოთ ასეთი:

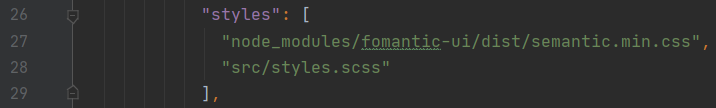


რაც შეეხება ავტორიზაციის გვერდს შეგიძლიათ თქვენ გადაწყვიტოთ ან ცალცალკე ფოლდერებად და შესაბამისად კომპონენტებად დაყოთ შესვლა და რეგისტრაცია ან ერთ კომპონენტში წეროთ ლოგიკები (გირჩევდით ცალცალკე გაკეთებას, მეორე შემთხვევაში გაგირთულდებათ). მოცემული ფაილური სტრუქტურის არჩევა აუცილებელია, ვინაიდან ეს სტრუქტურა მაქსიმალურად მიმსგავსებულია იმ სტრუქტურაზე რასაც რეალურად ვიყენებთ ბანკის ანგულარის პროექტზე სამუშაოდ. 1. გვაქვს განცალკევებული shell და auth კომპონენტები, რომლებიც ერთმანეთთან არაფერ შუაში არ არიან და დამოუკიდებლად იტვირთებიან იმის მიხედვით არის თუ არა კლიენტი ავტორიზებული. 2. Shell-ს გააჩნია sidebar და header

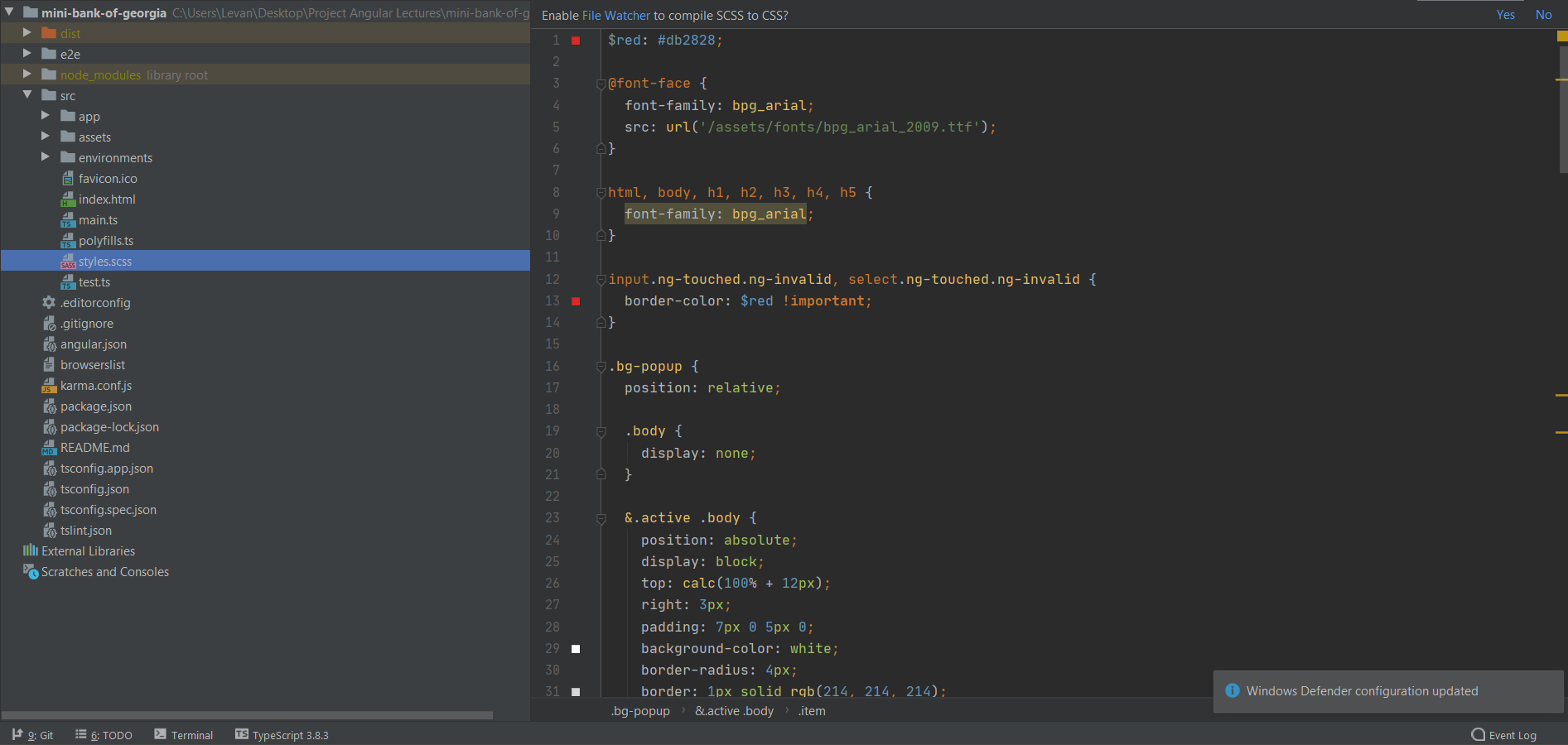
# საწყისი პროექტი და სტილები

საწყისი პროექტი შედგება დაყენებული და გამზადებული semantic ბიბლიოთეკისგან (1), ნაწილობრივ გამზადებული სტილებისგან (2), ასევე, საწყისი ავტორიზაციის (3) და გარსის (shell) გვერდებისგან (4).

## semantic.min.css



## styles.scss



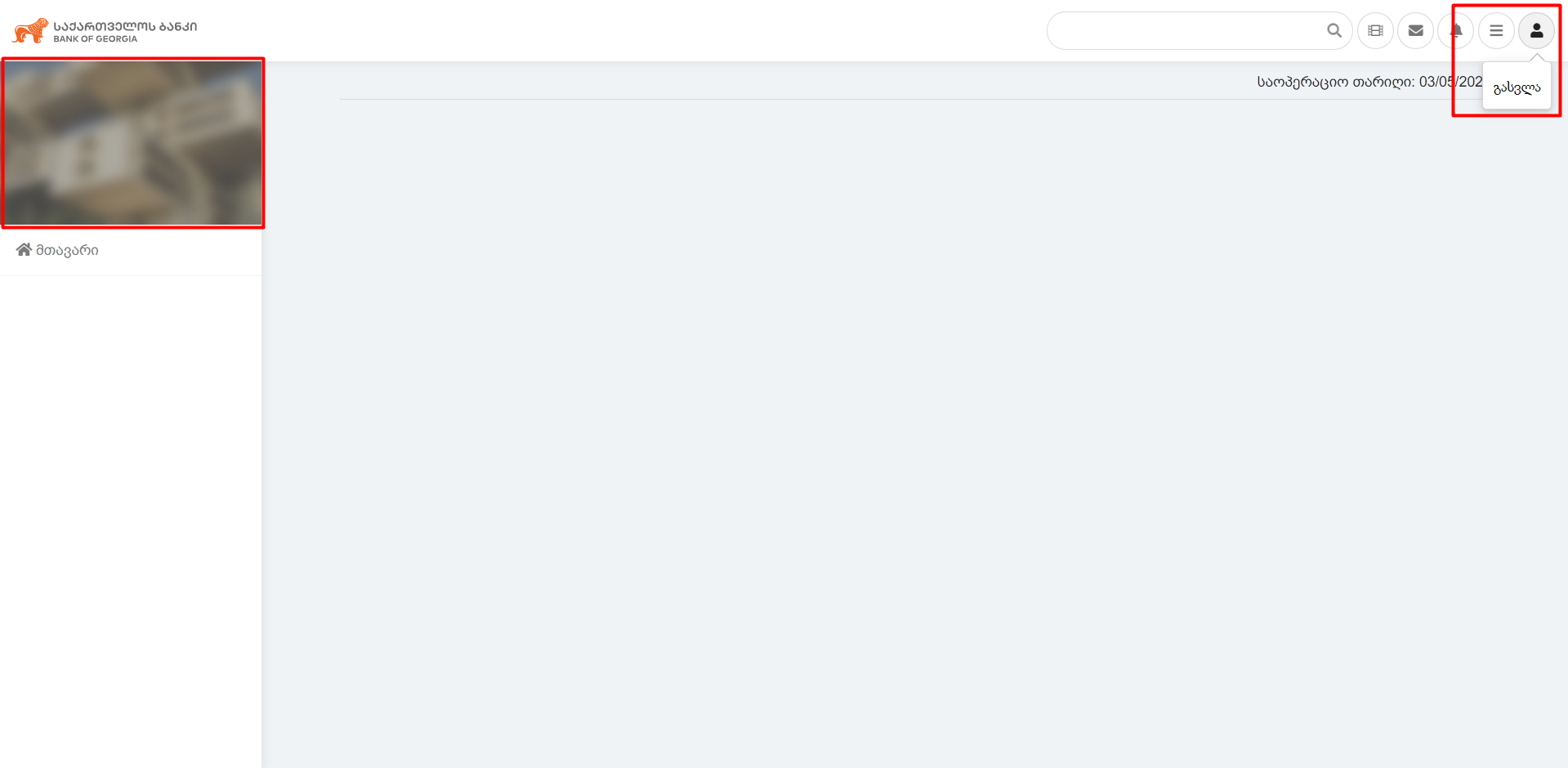
როგორც ხედავთ პირველ და მეორე ეტაპზე ვეცადე გამემზადებინა სტილები რათა ამ საკითხზე ბევრი ფიქრი არ მოგიწიოთ. პირველი რაც აღსანიშნავია გამოიყენეთ semantic-ის ნებისმიერი კლასი თუ საჭიროდ ჩათვლით, თუმცა თითოეული ნაწილისთვის მიწერილი მექნება რომელი კლასების გამოყენება იქნება რეკომენდირებული. (ერთადერთი გამონაკლისი იუზერის ინფო და კლიენტის ჰედერია, რომელშიც მოგიწევთ თქვენი css შესაძლებლობები გამოავლინოთ). მთლიანობაში ცხადია ყველაზე ნაკლები მნიშვნელობა სტილებს მიექცევა, თუმცა მაინც მნიშვნელოვანია საიტი ვიზუალურად თითქმის იდეალურად ასახავდეს სქრინებზე მოცემულს.

## /auth



განსხვავებით პირველი (1) და მეორე (2) ნაწილებისგან ამ გვერდზე მოგიწევთ უკვე თქვენ იმუშაოთ, შექმნათ ისეთივე ფორმები როგორიც ზემოთ სქრინებზეა ნაჩვენები. ამისთვის გამოგადგებათ: (ui form უკვე მზად არის ამიტომ გამოვტოვებ) field, ui left icon input, ui orange text, ui basic orange right floated button. ყველა ეს სტილი semantic-ის ნაწილია და შესაძლებელია ინფორმაცია ინახოს როგორც semantic-ის ასევე fomantic-ის საიტზე.

## /shell



ამ მზა კოდში თქვენი შესაცვლელი იქნება მხოლოდ იუზერის ინფორმაციის გამოჩენის ნაწილი, სადაც მცირედი scss-ის დაწერა მოგიწევთ, რათა გამოაჩინოთ მომხმარებლის ფოტო, სრული სახელი და ქვევით მიწერილი მომხმარებლის სახელი და ზედა მარჯვენა კუთხეში არსებული გასვლის popup გადასაკეთებელი გექნებათ დირექტივის საშუალებით, თუ დააკვირდებით ღილაკს active კლასი აქვს ამ კლასის გაქრობით ან გაჩენით popup გაქრება ან გაჩნდება. ამით მორჩა მზა კოდი და მათში ცვლილებები, შეგვიძლია გადავიდეთ თქვენს დასაწერ კომპონენტებზე.

# დასაწერი კომპონენტები დეტალურად

## Auth კომპონენტები

ეს ნაწილი როგორც უკვე ვახსენეთ შეგიძლიათ დაყოთ ორ ნაწილად ან შეგიძლიათ ერთ ფორმაში ეცადოთ მოაქციოთ, როგორც რეგისტრაციის, ასევე შესვლის გვერდი, თუმცა გირჩევდით პირველი გზით წასულიყავით.

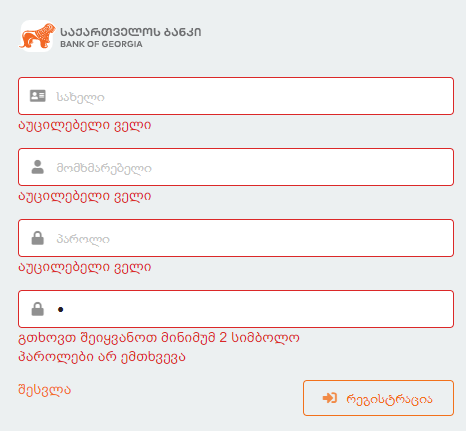
პირველ რიგში აუცილებელია ორივე გვერდს ქონდეს input ველები, რომლებსაც აუცილებელია ქონდეთ პატარა icon-ები რომლებიც შეგიძლიათ ზემოთ მითითებულ fomantic-ის საიტზე ნახოთ და არის ძალიან მარტივი გამოსაყენებელი. მეორე ორივე გვერდს აუცილებლად უნდა ქონდეს საპირისპირო გვერდზე გადასვლის საშუალება (შესვლის გვერდს რეგისტრაციის გვერდზე, ხოლო რეგისტრაციის გვერდს შევლის გვერდზე). და მთავარი ღილაკი რომლითაც გვერდიდან გამომდინარე ან დარეგისტრირდება ან შევა მომხმარებელი სისტემაში. კიდევ ერთხელ ამ ყველაფრისთვის საკმარისი იქნება მხოლოდ ეს კლასები (თუ მეტის გამოყენება მოგინდათ ან დაგჭირდათ რა თქმა უნდა არ არის პრობლემა): field, ui left icon input, ui orange text, ui basic orange right floated button.

აუცილებელია ფორმის ველების ვალიდაცია მოხდეს თქვენით შექმნილი ან ჩაშენებული ვალიდატორებით, მაგრამ მესიჯები ორივე შემთხვევაში ქართულად უნდა გამოუტანოთ (გირჩევდით თქვენით შექმნათ, გაიმარტივებდით საქმეს და სხვა ფორმებშიც გამოიყენებდით). აუცილებელი ვალიდაციები API გვერდზეც წერია, თუმცა აქაც დავაკონკრეტებ (საერთო ველებისთვის მახასიათებლები იგივეა):

* 1. username – მინიმუმ 2 სიმბოლო, მაქსიმუმ 30 სიმბოლო, დაუშვებელია space-ები, ველი აუცილებელია.
  2. password/confirmPassword - მინიმუმ 2 სიმბოლო, მაქსიმუმ 30 სიმბოლო, ველი აუცილებელია.
  3. Name - მინიმუმ 2 სიმბოლო, მაქსიმუმ 30 სიმბოლო, ველი აუცილებელია.

ყველა ველს, არასწორად შეყვანის შემთხვევაში, უნდა გაუაქტიურდეს ქვევით წარწერა რომელიც მიუთითებს შეცდომაზე. ველის ჩარჩოს გაწითლების ლოგიკა გლობალური scss-ის საშუალებით ავტომატურად დაიჰენდლება. ველის ვალიდაციის მესიჯი უნდა გამოჩნდეს მხოლოდ იმ შემთხვევაში თუ ველი touched არის, ანუ თუ ველში ერთხელ მაინც ჩადგა და გამოვიდა. *(აქვე პატარა რჩევა, შეგიძლიათ გააკეთოთ კომპონენტი რომელსაც validation-ს ან validation-message-ს დაარქმევთ, გადაცემთ FormControl Input-ს და ამ FormControl-დან წაიკითხავს ერორებს და გამოაჩენს, ერორების განახლების შემთხვევაში განაახლებს ან წაშლის ამ ერორებს გვერდიდან, კიდევ ერთხელ ამას თუ გააკეთებთ ერთხელ რთული/ჩახლართული გრძელი კოდი მოშორებული გექნებათ სხვა ველებიდან და სხვა ფორმებიდანაც)*

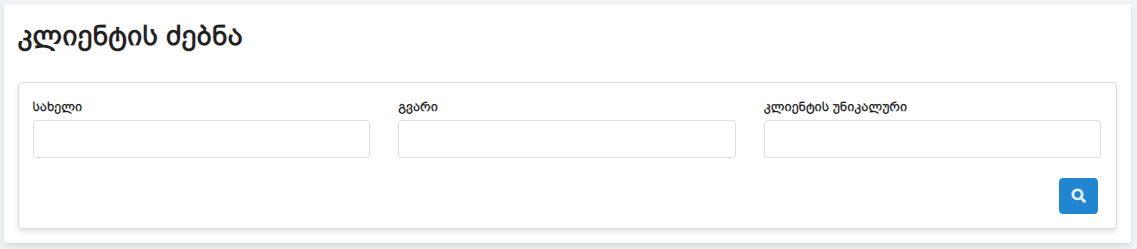
თუ მომხმარებელმა ყველა ვალიდაცია სწორად გაიარა შეგიძლიათ გააგზავნოთ Request სერვერზე, რომელიც დაგიბრუნებთ, ახლად შექმნილი მომხმარებლის შესახებ ყველა საჭირო ინფორმაციას, მათ შორის token-ს და token-ის ვადის გასვლის თარიღსაც მილიწამებში, რის შემდეგაც თქვენი მისიაა ისე შეინახოთ ეს მონაცემები რომ გვერდის დარეფრეშება არ იწვევდეს log out-ს. (სერვერის API-ზე უფრო დეტალურად ვისაუბრებთ ცალკე სექციაში).



## კლიენტის ძებნა

იუზერის და-login-ების შემდეგ, ვიწყებთ ნამდვილად დამოუკიდებელი კომპონენტის შექმნას, რომელიც აუცილებელია გაითვალისწინოთ რომ მოცემული სტრუქტურის მიხედვით უნდა იყოს შექმნილი. ანუ ვქმნით, src/app/shell/modules ფოლდერში კიდევ ერთ ფოლდერს და მოდულს bpm-ს რაც „ვაბანკში“ ერთ-ერთი გლობალური მოდულის სახელია. ამ მოდულის bpm000 ფორმა წარმოადგენს იმ გვერდს რომლიდანაც შესაძლებელია კლიენტის მოძებნა რათა შემდეგ მასში შევიდეთ. ეს გვერდი უნდა იყოს დალოგინებული მომხმარებლის მთავარი გვერდი, პირველი გვერდი რომელიც საიტზე შესვლის შემდეგ გამოუჩნდება.

კლიენტის ძებნის საწყისი ვიზუალი საკმაოდ მარტივია: გვაქვს ერთი „სეგმენტი“ რომლის შიგნითაც გვაქვს სიგრძეზე გასწორებული სამი input და ერთი ძებნის ღილაკი. ამის ასაწყობად დაგჭირდებათ (როგორც გითხარით სტილებზე ბევრი ფიქრი არ არის მთავარი ამოცანა ამ პროექტში) .ui.segments კლასი, რომლის შიგნითად შეგიძლიათ ორი ცალი .ui.segment ელემენტი გქონდეთ, პირველში გექნებათ h1 ტაგი და ჩაწერთ ამ გვერდის დასახელებას. რაც შეეხება მეორეს, აქ კიდევ ერთ სეგმენტს გააკეთებთ რომელსაც უკვე bordered კლასს დაუმატებთ (ეს კლასი სხვა შემთხვევებშიც როცა დაგჭირდებათ ეტყვის სემანტიკს რომ ამ სეგმენტს უნდა ქონდეს ჩარჩოები). შიდა სეგმენტში შეგიძლიათ შექმნათ .ui.three.column.grid რომელშიც უკვე თითოეულ .column კლასში უკვე გამოყენებული .field-ები თავის input-ებითა და label-ებით შეგიძლიათ ჩააგდოთ. ბოლო დეტალი კი .ui.right.floated.blue.icon.button არის, რომელიც შეგიძლიათ ახალ .grid ელემენტში ჩააგდოდ და შედეგად მიიღებთ ასეთ ვიზუალს.

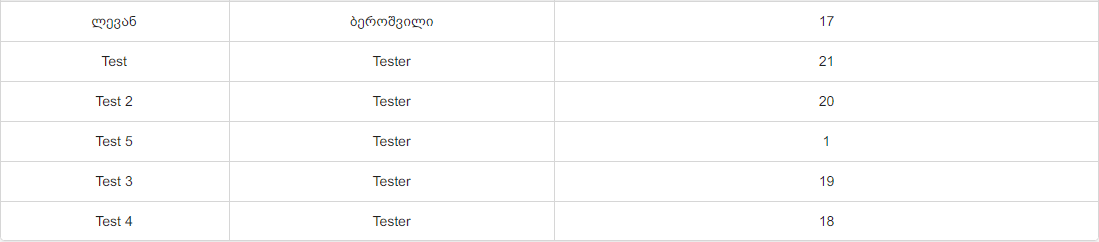


ამ ფორმას არ ჭირდება არანაირი ვალიდაცია, მომხმარებელს შეუძლია შეიყვანოს რაც უნდა თქვენ კი უნდა გააგზავნოთ მოთხოვნა სერვერზე (API-ზე) საიდანაც დაგიბრუნდებათ ცარიელი ან კლიენტების არსებობის შემთხვევაში კლიენტებით სავსე მასივი, რომელიც უნდა გამოიყენოთ ამავე გვერდზე შემდეგი ვიზუალის გამოსატანად.

შემდეგი ვიზუალი ერთი შეხედვით საკმაოდ კომპლექსური ჩანს თუმცა ყველა სტილი გამზადებულია იმისთვის რომ ააგოთ სტანდარტული html-ის table ტაგი რომლის შვილებიც არიან thead და tbody ტაგები, პირველის შვილები არიან th ტაგები რომლითაც შეგიძლიათ ჩამოწეროთ თითეული ველის სათაური.



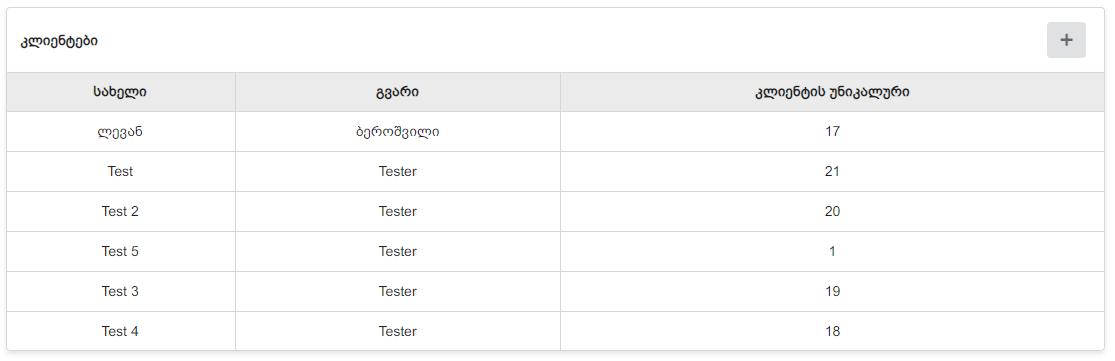
მეორე tbody შეიცავს რამდენი ჩანაწერიც არის იმდენ tr-ს და თითოეულისთვის th-ების შესაბამის td-ებს ანუ ველებს.



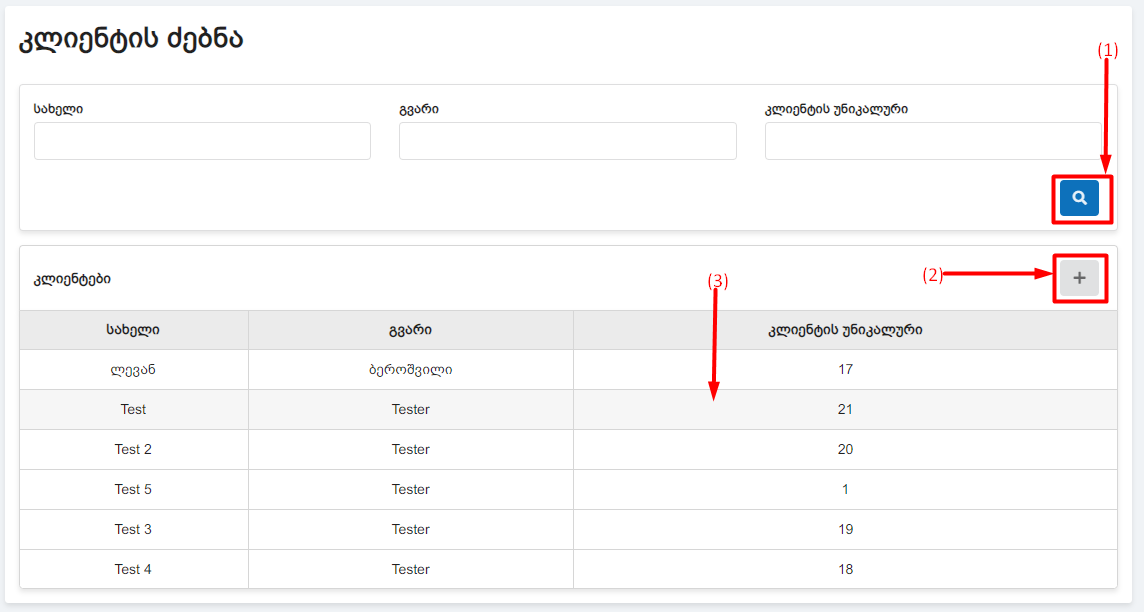
ერთადერთი განსხვავებული ელემენტი იქნება ამ თეიბლის თავზე .table-header კლასის მქონე ელემენტი, რომელშიც span ელემენტით ამ თეიბლის სათაური და ერთი .ui.icon.button უნდა დავაგდოთ დამატების იკონით, რომელიც შემდეგ კლიენტის დამატების ფორმაზე გადაგვიყვანენ. ამ ელემენტების სწორად დალაგებასაც უკვე არსებული სტილები უზრუნველყოფენ.



შედეგად მივიღებთ სრულყოფილ თეიბლს რომელიც გამოიყურება ასე:



მთლიანობაში ამ გვერდზე გვექნება შემდეგ ქმედებები:



1. ღილაკზე დაჭერით მოგვაქვს კლიენტები და ვაჩენთ.
2. ღილაკზე დაჭერით გადავდივართ კლიენტის რეგისტრაციის ფორმაში.
3. ქმედებით, ანუ მთელ სტრიქონზე ნებისმიერ ადგილზე დაჭერით შევდივართ ამ კლიენტში.

## კლიენტის დამატება

კლიენტის რეგისტრაციის ვიზუალი კიდევ უფრო მარტივია ვიდრე წინა ფორმები, რაც მთავარია ზუსტად ასეთივეს დაწერა მოგიწევთ როგორც ამ პროექტის ანგარიშის დამატებისა და ფულის გადარიცხვის ისე რეალურად „ვაბანკის“ პროექტზე მუშაობის შემთხვევაში უმეტესწილად. როუტი ამ შემთხვევაშიც ჩვენი სტანდარტების მიხედვით უნდა იყოს /bpm/bpm001. დეტალურად აღარ ავხსნი თუმცა აქაც გვჭირდება .ui.segments, სამი ცალი .ui.segment-ით, პირველში სათაური წავა, მეორეშეი input field-ები და მესამეში ღილაკი. რაც შეეხება თვითონ field-ებს აქ ცოტა ახალი ლოგიკა შემოდის „ვაბანკის“ ფორმების უმეტესობა (ძველ დაწერილ ფორმებს თუ არ ჩავთვლით) გამოიყურება ასე .ui.form ტაგში თითოეული სტრიქონისთვის რომელშიც მხოლოდ ერთი input უნდა არსებობდეს ვქმნით .fields კლასის ელემენტს, რომელიც სემანტიკის წესების მიხედვით შეგვიძლია დავყოთ 16 თანაბარ ნაწილად, აქედან სტანდარტულად 9-ს იშვიათ შემთხვევებში 10 ნაწილს ვიყენებთ უშუალოდ input ელემენტისთვის. ამას ვაკეთებთ .nine.wide.field კლასის საშუალებით (10 ნაწილის დასაკავებლად ვწერთ .ten.wide.field-ს). ამ კლასის შიგნით არსებულ input და label ტაგებს უკვე ექნებათ შესაბამისი სტილები რათა ფორმა ისეთი გამოჩნდეს როგორიც დავალებაშია მოცემული.

ვიზუალური ნაწილის გარდა ამ შემთხვევაშიც აუცილებელია ვალიდაციები, ვალიდაციების ლოგიკა ქვევით არის მოცემული, რაც შეეხება გამოჩენას და ლოგიკას, რჩება იგივე რაც აქამდე იყო და როგორც აქამდე ვთქვი თუ გექნებოდათ შექმნილი ცალკე კომპონენტი რომელიც ვალიდაციის ლოგიკას დაჰენდლავდა იგივე კომპონენტის გამოყენებით ვალიდაციის ლოგიკები ამ ფორმაში ძალიან მარტივი ჩასადები იქნებოდა. ვალიდაციები რომლებიც უნდა გამოიყენოთ:

1. firstName - მინიმუმ 2 სიმბოლო, მაქსიმუმ 30 სიმბოლო, ველი აუცილებელია.
2. lastName - მინიმუმ 2 სიმბოლო, მაქსიმუმ 30 სიმბოლო, ველი აუცილებელია.
3. plusPoints - რიცხვითი მნიშვნელობა (ეს არ არის საჭირო ვალიდაციით გაკეთდეს, უბრალოდ input type=”number” შეგიძლიათ დაწეროთ), მინ მნიშვნელობა 0.

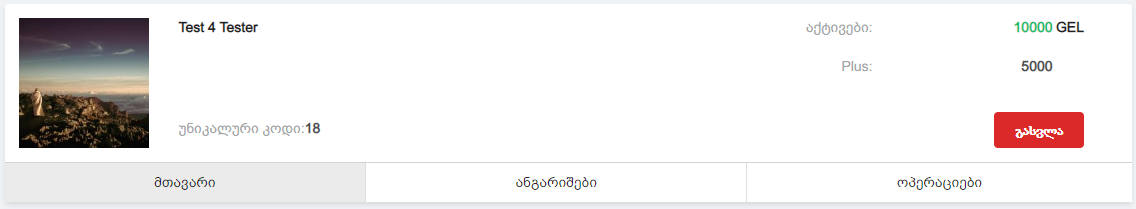
შენახვაზე დაჭერამ უნდა გამოიძახოს კლიენტის ჩამატების API ხოლო წარმატებით დაბრუნებული პასუხის შემთხვევაში დაგიბრუნდებათ კლიენტის ყველა ის მონაცემი რომელიც კლიენტის ძებნაში გექნებოდათ, შესაბამისად იგივე ლოგიკით როგოც წინა გვერდის ცხრილში კლიენტის არჩევით უნდა გადაიყვანოთ მომხმარებელი კლიენტის პროფილში.



## კლიენტის Header

ეს ნაწილი სტილების მხრივ ცოტათი tricky იქნება. როგორც უკვე გითხარით ეს ის შემთხვევაა როცა ყველა სტილი გამზადებული არ გექნებათ და თქვენით მოგიწევთ ფიქრი როგორ უნდა დააგდოთ მარცხენა მხარეს კლიენტის ფოტო (რომელიც ავტომატურად გენერირდება და ყველა კლიენტს როგორც რეგისტრაციის დროს ისე კლიენტის დამატების დროს მოყვებათ ხოლმე), სახელი და უნიკალური კოდი, ხოლო მარჯვენა მხარეს აქტივები (რომელიც ანგარიშებზე არსებული თანხების ჯამია და ასევე მოყვებათ კლიენტებს), პლიუს ქულები და გამოსვლის ღილაკი ისე როგორც სურათზეა მოცემული. ვინაიდან როგორც თავიდანაც ვთქვი ამ დავალების და კურსის მიზანი არ არის თქვენი სტილებში გამოცდა და შესაბამისად ყველაზე ნაკლები წონა აქვს დავალებაში სტილებს, ამ ნაწილის მიზანი კი მხოლოდ ის არის რომ მე გავიგო რამდენად იცით ან არ იცით თქვენ თვითონ css ნებისმიერი კითხვა რომელიც გაგიჩნდებათ (სტილებთან დაკავშირებული) შეგიძლიათ მკითხოთ და ყველაფერზე გიპასუხებთ, თუ საჭირო გახდა კოდსაც დაგიწერთ, მაგრამ კარგი იქნება თუ თქვენით შეძლებთ ამ ყველაფრის გაკეთებას. სანამ შიგნით მომხმარებლის ინფორმაციის დალაგებაზე გადახვალთ, აქვე გეტყვით რომ კლიენტის ჰედერი უნდა იყოს .ui.segments, რომელსაც ეყოლება ერთი .ui.segment შვილი სადაც კლიენტის ინფორმაცია გამოჩნდება და მეორე უნდა იყოს .ui.horizontal.segments ელემენტი რომელშიც სამი .ui.segment წავა და თითოეული შეესაბამება ნავიგაციის ერთ ლინკს. ცხადია მოსაფიქრებელი გაქვთ როგორ უნდა გამოაჩინოთ აქტიური როუტი.

ლოგიკის მხრივ, კლიენტის ჰედერის ლოგიკა მარტივია. პირველ რიგში კლიენტში შესვლა უნდა გავდეს იუზერზე დალოგინებას, ანუ უნდა შევინახოთ კლიენტის მონაცემები გლობალურად და გვერდის დარეფრეშების შემთხვევაში ისევ შესული უნდა დავრჩეთ კლიენტში. მეორე კლიენტის ჰედერი უნდა ჩანდეს ყველა გვერდზე შელის ფარგლებში, თუნდაც კლიენტის ძებნის გვერდზე, თუ კლიენტში შევალთ და შემდეგ ლინკის საშუალებით გადავალთ კლიენტის ძებნის გვერდზე. მესამე უნდა გაითვალისწინოთ რომ ყოველთვის როცა ახალი ანგარიშის გახსნა ან გადარიცხვის ოპერაცია მოხდება, ჰედერში აქტივები უნდა განახლდეს. და მეოთხე არ დაგავიწყდეთ რომ თუ მომხარებელი (და არა კლიენტი) გავა საერთოდ სისტემიდან და ამ დროს კლიენტი იყო გახსნილი, კლიენტიდან გასვლაც ავტომატურად უნდა მოხდეს.

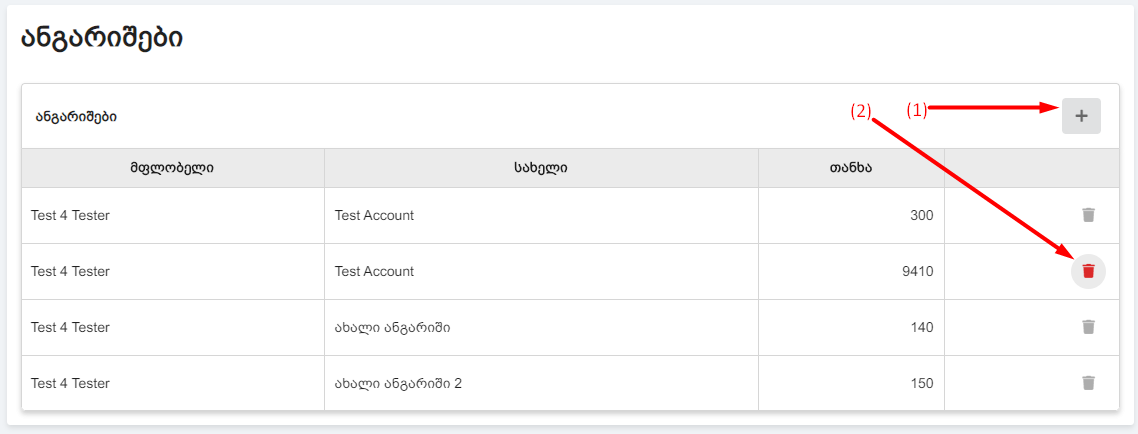


აქვე ვთქვათ რომ, პირველი გვერდი კლიენტში შესვლის შემდეგ რაც გამოჩნდება არის /krn/krnicp, ასეთი კომპონენტის შექმნაც მოგიწევთ თუმცა ეს კომპონენტი სრულიად ცარიელი უნდა იყოს, ვინაიდან ამ გვერდზე მხოლოდ კლიენტის ჰედერი უნდა ჩანდეს. ყველა krn-ის ქვე როუტს უნდა ქონდეს client-ის არსებობის შემმოწმებელი guard. Krn მოდული „ვაბანკში“ kernel-ადაც შეიძლება მოვიხსენიოთ, ანუ ყველა მოდული ნაწილობრივ იყრის აქ თავს.

## ანგარიშები

ანგარიშების გვერდიც ვიზუალურად მარტივი გასაკეთებელია, ახალი არაფერი არ არის, გვაქვს .ui.segments ერთი სათაურის სეგმენტით და ერთი ძირითადი ნაწილის სეგმენტით. ძირითად ნაწილში გვაქვს ცხრილი რომელიც იგივე ლოგიკით აიგება რა ლოგიკითაც აიგო კლიენტის ძებნაში კლიენტების ცხრილი, ერთადერთი ის განსხვავება იქნება რომ ამ ცხრილს ექნება ერთი ბოლო ცარიელი th, რომლის ქვეშაც იქნება წაშლის იკონკები, წაშლის იკონკისთვის უნდა გამოიყენოთ semantic-ის სტანდარტული .trash.button.icon რომელსაც გარკვეული სტილები წინასწარ ექნება გამზადებული.

ქმედებები მარტივია, გვაქვს მხოლოდ ახალი ანგარიშის გახსნის გვერდზე გადასვლის ღილაკი (1) და ანგარიშის წაშლის ღილაკი (2), რომელიც დაჭერისთანავე იძახებს წაშლას და წარმატების შემთხვევაში ანახლებს სიას.



## ანგარიშის შექმნა

ანგარიშების გვერდი ასევე krn მოდულში უნდა იყოს, როუტების ხეში კი ანგარიშების ქვეშ, ანუ მისი როუტი იქნება /krn/accounts/create, რაც იმას ნიშნავს რომ ამ გვერდზე ყოფნისას ჰედერში ანგარიშების ღილაკი აქტიურად უნდა იყოს მონიშნული.

ანგარიშის შექმნა ვიზუალურად ზუსტად იგივენაირი გვერდია როგორიც კლიენტის შექმნის, მხოლოდ ველებია განსხვავებული, ამიტომ ვიზუალზე ბევრი ლაპარაკის გარეშე პირდაპირ გადავალ ველებზე და ვალიდაციებზე:

1. accountName - მინიმუმ 2 სიმბოლო, მაქსიმუმ 30 სიმბოლო, ველი აუცილებელია.
2. amount - რიცხვითი მნიშვნელობა (ეს არ არის საჭირო ვალიდაციით გაკეთდეს, უბრალოდ input type=”number” შეგიძლიათ დაწეროთ), მინ მნიშვნელობა 0.

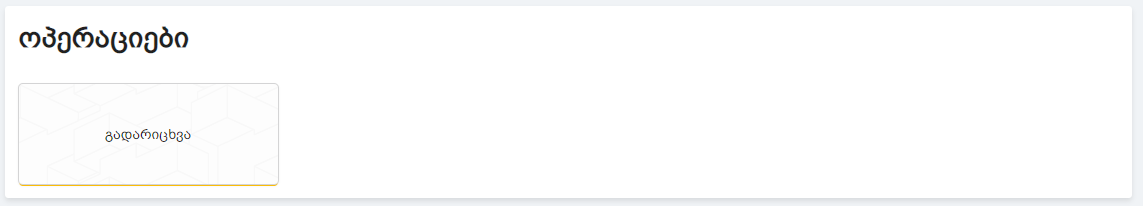
ერთადერთი ქმედება აქვს ფორმას: შენახვა, რომელზე დაჭერის შემდეგაც ნავიგაცია უნდა მოხდეს ანგარიშების გვერდზე, აქ გასათვალისწინებელია რომ შექმნის შემდეგ ჰედერში აქტივები უნდა განახლდეს.

## ოპერაციები

ოპერაციების გვერდიც krn მოდულშია. მისი როუტია /krn/operations და „ვაბანკში“ ამ გვერდიდან საკმაოდ ბევრ ოპერაციის გვერდზე შეიძლება გადასვლა, ჩვენს მინი საქართველოს ბანკში კი მხოლოდ გადარიცხვების გვერდზე ნავიგაცია იქნება შესაძლებელი.

ვიზუალურად ამ გვერდზე საჭიროა .ui.card კომპონენტები ეყაროს, ერთ მწკრივში უნდა ეტეოდეს 4 ასეთი card რის გამოც ძირითადი სეგმენტის შვილი შეგვიძლია გავხადოთ .ui.four.cards ხოლო მასში ჩავდოთ .ui.raised.yellow.link.card რისი საშუალებითაც ჩვენი ქარდი გახდება ისეთი როგორიც სქრინებშია. არასავალდებულოა, მაგრამ სასურველია card-ის შიგნით კიდევ ერთი absolute position-ით გასწორებული background-image-იც ჩააგდოთ რაც საშუალებას მოგცემთ კიდევ უფრო დაამსგავსოთ თქვენი card „ვაბანკში“ გამოყენებულ card-ს.

ამ ერთადერთ ელემენტზე დაკლიკებით, მომხმარებელი უნდა გადადიოდეს გადარიცხვების გვერდზე.



## გადარიცხვები

ამ გვერდის ვიზუალზეც არ ვისაუბრებ ბევრს, რადგან ძირითადად ჩვეულებრივი ფორმაა, როგორიც წინა ორი იყო (კლიენტის რეგისტრაცია და ანგარიშის შექმნა). თუმცა ერთი პატარა განსხვავება მაინც არის input-ების მაგივრად გამგზავნ და მიმღებ ანგარიშში უნდა გამოიყენოთ select ტაგი, რომლის შიგნითად შეგიძლიათ ჩაწეროთ option ტაგები და რამდენსაც ჩაწერთ იმდენი ასარჩევი ექნება select-ს. Option ტაგს აქვს value ატრიბუტი რომელშიც რასაც ჩაწერთ დაესეტება form control-ს რომელსაც selects-ს მიაბამთ, ხოლო რასაც შიგნით ჩაწერთ ის გამოჩნდება. (აქვე რჩევა, სავარაუდოდ ინფორმაციას ngFor-ით გადაყვებით, ngFor-მდე კი დაწერეთ ერთი ზედმეტი option რომელსაც value-დ დაუსეტავთ ცარიელ სტრინგს ხოლო შიგნით ჩაწერთ „აირჩიეს“.) ვიზუალურად გამგზავნ ანგარიშში უნდა გამოაჩინოთ {{accountKey}} – {{amount}}GEL ({{accountName}}) და ეს მონაცემები უნდა იყოს შემოსული კლიენტის, ხოლო მიმღებ ანგარიშში უნდა გამოჩნდეს {{accountKey}} – {{amount}}GEL ({{clientFirstName}} {{clientLastName}}) და ეს მონაცემები უნდა იყოს ყველაკლიენტის რომელიც სისტემაში არსებობს, ყველა მოცემული მონაცემი და ველი accounts-ის სერვისიდან მოგივათ.

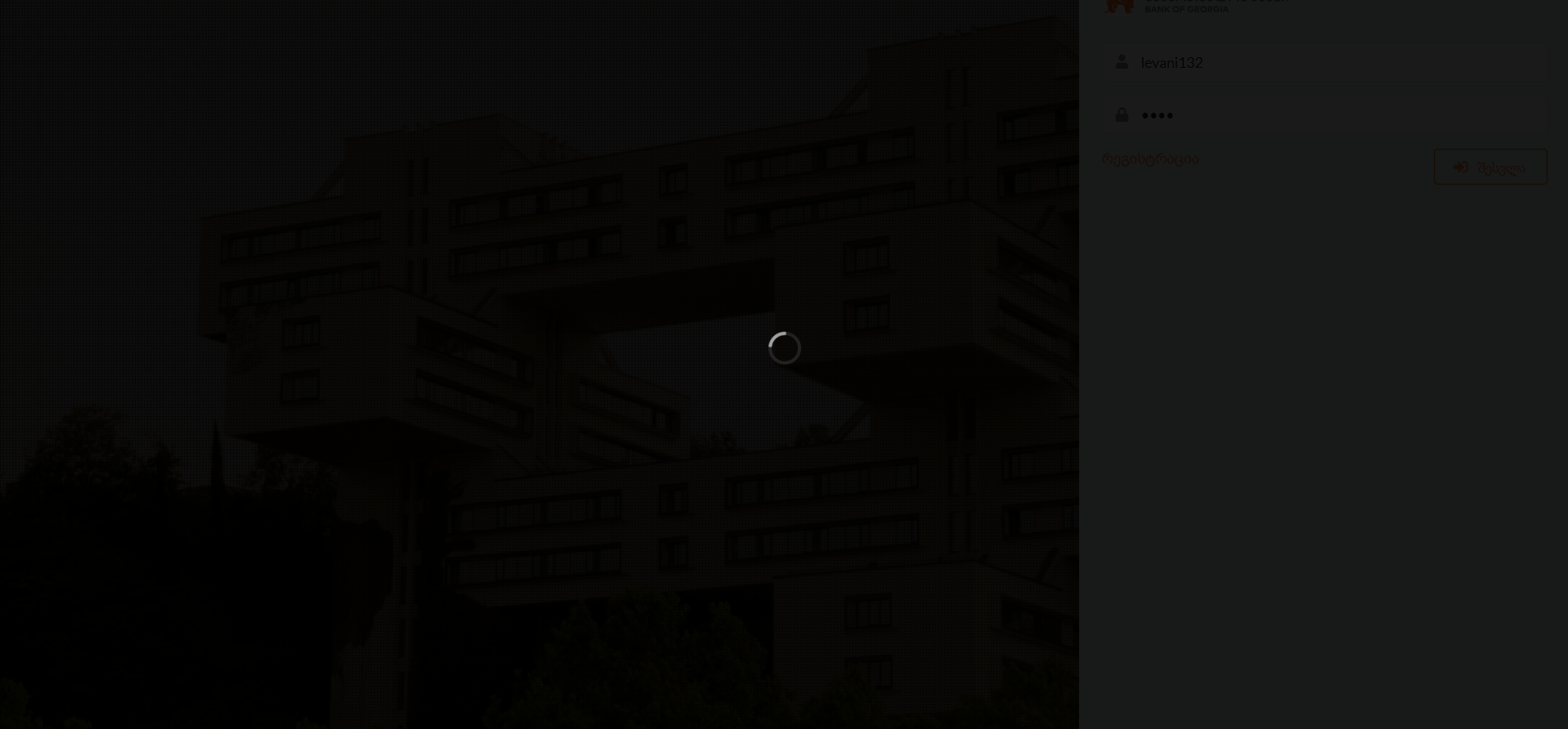
ფორმის ლოგიკა ყველა მხრივ იგივე რჩება, ვალიდატორებიც გამოიყენეთ ისე როგორც აქამდე, ველებს კი ეს ვალიდაციები ჭირდებათ:

1. senderAccountKey - ველი აუცილებელია.
2. receiverAccountKey - ველი აუცილებელია.
3. amount - რიცხვითი მნიშვნელობა (ეს არ არის საჭირო ვალიდაციით გაკეთდეს, უბრალოდ input type=”number” შეგიძლიათ დაწეროთ), მინ მნიშვნელობა 0.

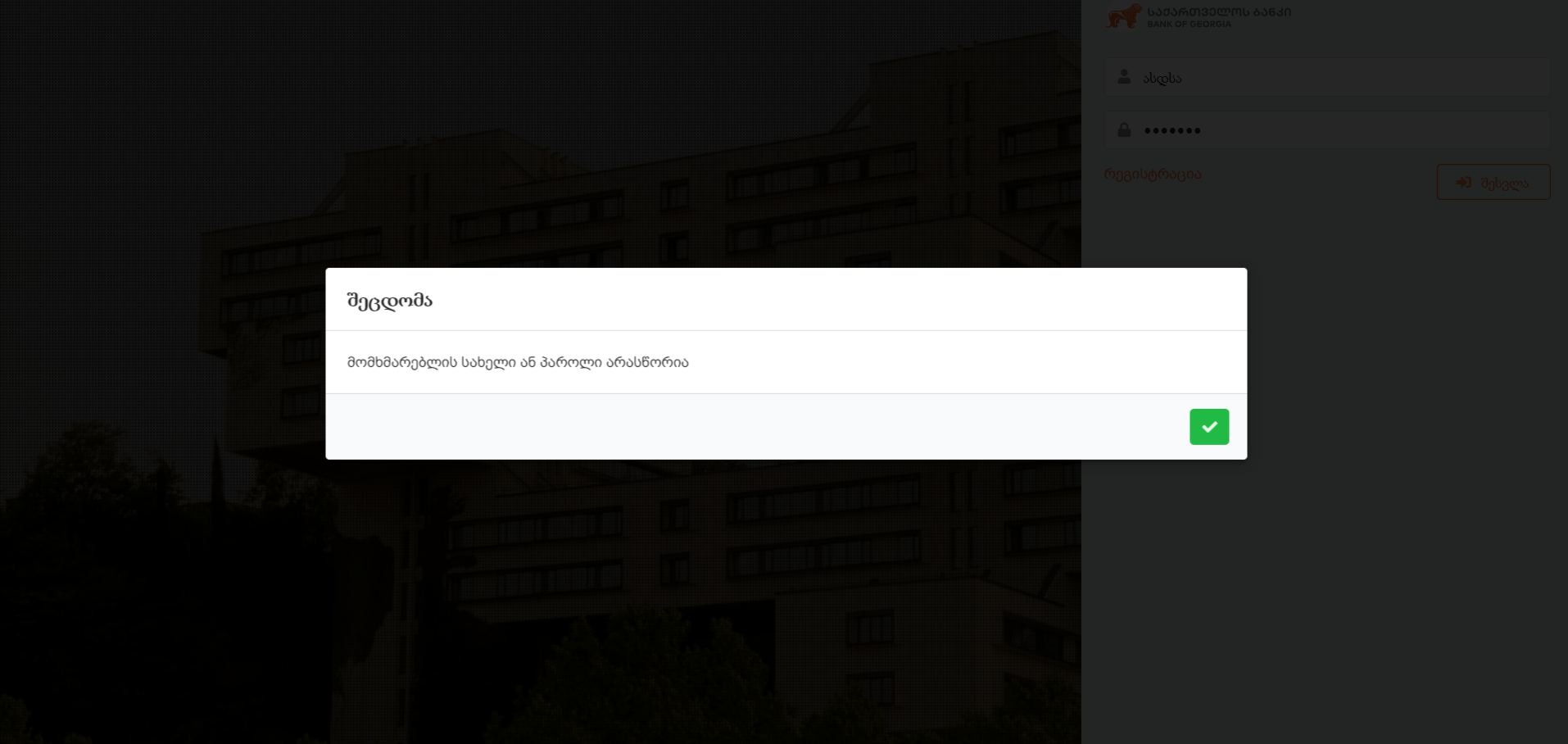
შენახვაზე დაჭერის შემდეგ უნდა გაეშვას /transfer სერვისი და მომხმარებელი გადაყვანილი იქნას ანგარიშების გვერდზე. ამ დროს აუცილებელია ისევ გაითვალისწინოთ კლიენტის ჰედერში ქტივების განახლების ლოგიკა.

# დამატებითი დავალებები

1. კიდევ ერთი ანგულარის შესაძლებლობის გამოსაყენებლად აუცილებელია ყველა თანხობრივი ველი გამოაჩინოთ ასწილადის სახით, ანუ მთელ რიცხვებსაც .00 უნდა მიუწეროთ. ასეთი ველები არის ერთი ჰედერში აქტივების ველი, მეორე ანგარიშების ცხრილში თანხის ველი და მესამე გადარიცხვების გვერდში select-ის option-ების ვიზუალში თანხის ველი.
2. ყველა სერვისის გაშვების წინ და დასრულების შემდეგ უნდა გქონდეთ loader ანუ მთელი ეკრანი უნდა შავდებოდეს ისე რომ გამჭირვალე სიშავის უკან გვერდი მაინც ჩანდეს და წინ spinner იყოს. ამისთვის შეგიძლიათ გამოიყენოთ კლასები ui active dimmer და ui loader. ვიზუალურად უნდა გამოიყურებოდეს ასე:



1. სერვისების მიერ დაბრუნებული ყველა error უნდა გამოდიოდეს მოდალში, რომელიც სურვილისამებრ ან დინამიურად უნდა ჩატვირთოთ ან უბრალოდ ngIf-ით. (აქვე გაითვალისწინეთ, რომ თუ ყველაფერს სწორად დაწერთ მაშინ სერვისიდან ერორი მხოლოდ არასწორად შეყვანილი პაროლის შემთხვევაში უნდა დაგიბრუნდეთ.) გამოიყენეთ semantic-ის კლასები: ui modal, header, content, actions. ვიზუალურად უნდა გამოიყურებოდეს ასე:



# API

როგორც გამოსაყენებელ მასალაში წერია, სერვერზე გამოძახებების გასაკეთებლად swagger-ზე აღწერილი API უნდა გამოიყენოთ. ვინც არ იცით API ნიშნავს Application Programming Interface-ს. ანუ აქ მოცემული აღწერა არის ერთგვარი interface, ანუ request-ების ჩამონათვალი მისი პარამეტრებით და რას აკეთებს და რას აბრუნებს იმის განმარტებით. წესით swagger-ზეც გარკვევით წერია თუმცა, რომ დავაზუსტოთ თქვენი request-ები უნდა გავიდეს <https://bog-angular-course-api.herokuapp.com/> ამ ლინკზე დახრილი ხაზის იქით რა უნდა დაიწეროს რა მეთოდით, პარამეტრებით და result-ებით აქ იქნება აღწერილი ყველაფერი. მგონი კითხვები არ უნდა გაჩნდეს და მეტი განმარტება არ უნდა გჭრდებოდეთ ამასთან დაკავშირებით, თუ მაინც გაჩნდა მეც აქ ვარ და დაგეხმარებით.

დასრულებული საიტი უნდა გამოიყურებოდეს ამ საიტის მსგავსად: <https://bog-angular-course-final.herokuapp.com/> .

სულ ეს იყო, წარმატებები.