

საქართველოს ტექნიკური უნივერსიტეტი

გია სურგულაძე, გიორგი კაკაშვილი,
გიორგი მარტიაშვილი

მობილური აპლიკაციების დეველოპმენტის საფუძვლები (Java, Android)



თბილისი
2020

უაკ 0045

განხილულია მობილური აპლიკაციების აგების საფუძვლები, მათი დაპროგრამების მეთოდოლოგია და ძირითადი პრინციპები Android ოპერაციული სისტემისათვის. გადმოცემულია მობილური აპლიკაციის სასიცოცხლო ციკლის ეტაპები, სისტემის ბიზნეს-მოთხოვნების ფორმირების, დაპროექტების/დიზაინის (UI/UX), პროგრამული დეველოპმენტის (Java და Kotlin ენები), ტესტირების, დანერგვისა და თანხლების პროცესების გათვალისწინებით. წიგნის ორიგინალური მხარეა მობილური აპლიკაციების აგება და გამოყენება ტელეფონის სფეროში, კერძოდ, მობილურებში შემავალი და გამავალი ზარების და SMS/MMS-ების მონიტორინგისთვის Intent-ების გამოყენებით, აგრეთვე ლოკაციის სერვისებისა და რუქის ანიმაციისთვის და სხვ. პრაქტიკული რეალიზაციის მაგალითები წარმოდგენილია გადაუდებელი დახმარებისა და საგანგებო სიტუაციებისათვის („112“). მონოგრაფიული სახელმძღვანელო განკუთვნილია ინფორმატიკის სპეციალობის სტუდენტების და განხილული საკითხებით დაინტერესებული მკითხველისათვის.

რეცენზენტები:

პროფ. ე. თურქია (სტუ)

პროფ. დ. გულუა (ბახრეინის უნივერსიტეტი)

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახოზაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, თ. ლომინაძე, ჰ. მელაძე, თ. ოზგაძე, გ. სურგულაძე (რედაქტორი), გ. ჩაჩანიძე, ა. ცინცაძე, ზ. წვერაიძე, ო.შონია

© სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი“, 2020
ISBN 978-9941-8-2488-3

ყველა უფლება დაცულია. ამ წიგნის არც ერთი ნაწილის (იქნება ეს ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) გამოყენება არანაირი ფორმით და საშუალებით (იქნება ეს ელექტრონული თუ მექანიკური) არ შეიძლება გამოიმეცლოს წერილობითი ნებართვის გარეშე. საავტორო უფლებების დარღვევა ისჯება კანონით.

Gia Surguladze, Giorgi Kakashvili
Giorgi Martiashvili

**BASICS OF DEVELOPING MOBILE
APPLICATIONS
(Java, Android)**



© „IT-Consulting Research Center“ of Georgian Technical
University, Tbilisi, 2020

ISBN 978-9941-8-2488-3

ავტორთა შესახებ:

გია სურგულაძე – სტუ-ს „მართვის ავტომატიზებული სისტემების (პროგრამული ინჟინერიის) დეპარტამენტის უფროსი, პროფესორი, ტექნიკის მეცნიერებათა დოქტორი, გაეროსთან არსებული „ინფორმატიზაციის საერთაშორისო აკადემიის (IIA)“ ნამდვილი წევრი, სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრის“ ხელმძღვანელი, გერმანიის DAAD-ის გრანტის მრავალგზის მფლობელი, ბერლინის ჰუმბოლდტის, ნიურნბერგ-ერლანგენის და სხვა უნივერსიტეტების მიწვეული პროფესორი. 400 სამეცნიერო ნაშრომის ავტორი, მათ შორის 85 წიგნი და 62 ელ-სახელმძღვანელო გამოყენებითი პროგრამული ინჟინერიის სფეროში.

გიორგი კაკაშვილი – აკადემიური დოქტორი „ინფორმატიკაში“ (დაიცვა დისერტაცია სტუ-ში 2018 წელს), ინფორმატიკის ბაკალავრი (სტუ-2013), მაგისტი (სტუ-2015). თეოლოგიის ბაკალავრი (თბილისის სასულიერო აკადემია და სემინარია - 2018). არის „ბაბლო უზაღლესი სასწავლებლის“ IT სამსახურის უფროსი, სტუ-ს მოწვეული ასოც.პროფესორი. გამოქვეყნებული აქვს 10-ზე მეტი სამეცნიერო ნაშრომი, მონაწილეობდა 12 საერთაშორისო სამეცნიერო კონფერენციაში. ფლობს Web-დიზაინისა და აპლიკაციების ინფორმაციულ ტექნოლოგიებს, მულტიპარადიგმული და მობილური პროგრამირების ენებს.

გიორგი მარტიაშვილი – სტუ-ს „პროგრამული ინჟინერიის“ დეპარტამენტის დოქტორანტი, მობილური პროგრამული აპლიკაციების დაპროექტებისა და კონსტრუირების სპეციალისტ-დეველოპერი. მუშაობდა „112“-ში, სადაც მისი უშუალო მონაწილეობით შეიქმნა დღეს ფართოდ გამოყენებადი მობილური აპლიკაციები Android პლატფორმაზე. არის საერთაშორისო სამეცნიერო კონფერენციების მონაწილე-მომხსენებელი, გამოქვეყნებული სამეცნიერო ნაშრომების ავტორი მობილური პროგრამული აპლიკაციების სფეროში.

სარჩევი

შესავალი:	9
I თავი. მობილური აპლიკაციების პროგრამული	
პლატფორმები და ენები	11
1.1 შესავალი Android ოპერაციულ სისტემაში	11
1.1.1. რა არის Android ?	11
1.1.2. რატომ უნდა შევქმნათ აპლიკაციები Android-ზე ?	11
1.1.3. Android-ის ვერსიები	15
1.2. Android-ის პირველი პროგრამა	19
1.2.1. დეველოპმენტის პროცესი	19
1.2.2. კონცეპტუალიზაცია	20
1.2.3. მიზანშეწონილობის შეფასება	20
1.2.4. აპლიკაციის დიზაინის შემუშავება	21
1.2.5. პროგრამული უზრუნველყოფა	21
1.2.6. ტესტირება და გამოქვეყნება	22
1.3. Android Studio-ს ინსტალაცია	23
1.4. პირველი Android პროექტის შექმნა „Hello World“	31
1.5. Kotlin vs Java	36
1.6. ვირტუალურ მოწყობილობათა შექმნა და მართვა	42
1.7. მომხმარებლის ინტერფეისის შექმნა	43
1.8. Android ღილაკი და საილუსტრაციო მაგალითი	45
1.9. კოტლინის Android სია	53
1.10. Android-ის პარამეტრების მენიუ	58
1.11. Android Media Player და მისი მაგალითი	63
II თავი. ტელეფონია და SMS	73
2.1. ანდროიდის პროგრამული უზრუნველყოფა	73
2.2. განსახილველი თემები	75
2.3. ფიზიკური მოწყობილობის მხარდაჭერა	
ტელეფონიისთვის	76

2.4. ტელეფონის მონიშვნა, როგორც აუცილებელი ფიზიკური თვისება	76
2.5. ტელეფონის ფიზიკური მოწყობილობის შემოწმება..	77
2.6. ტელეფონის გამოყენება	78
2.7. ზარის წამოწყება	78
2.8. ტელეფონის Native Dialer-ის ჩანაცვლება	79
2.9. ტელეფონის პარამეტრებზე და მდგომარეობზე წვდომა	81
2.10. მოწყობილობის პარამეტრებზე წვდომა	82
2.11. სიმ-ბარათის პარამეტრების წაკითხვა	83
2.12. მონაცემთა კავშირისა და მიმოცვლის სტატუსის დეტალების წაკითხვა	85
2.13. ტელეფონის სტატუსის ცვლილების მონიტორინგი..	85
2.14. შემომავალი ზარების მონიტორინგი	87
2.15. მობილურის ადგილმდებარეობის ცვლილების ტრეკინგი	88
2.16. სერვისის ცვლილებების ტრეკინგი	89
2.17. მონაცემთა კავშირის და მონაცემთა გაცვლის სტატუსის ცვლილების მონიტორინგი	92
2.18. Intent Receiver-ების გამოყენება შემომავალი ზარების საკონტროლებლად	92
2.19. SMS და MMS	94
2.20. SMS და MMS გამოყენება თქვენს აპლიკაციაში	95
2.21. SMS და MMS გაგზავნა აპლიკაციიდან Intent-ის საშუალებით	95
2.22. SMS შეტყობინების გაგზავნა SMS Manager-ის საშუალებით	96
2.22.1. ტექსტური შეტყობინების გაგზავნა	97
2.22.2. SMS შეტყობინების ტრეკინგი	97
2.22.3. SMS შეტყობინების მაქსიმალური ზომა	100
2.22.4. ინფორმაციული შეტყობინებების გაგზავნა	101

2.25. შემომავალი SMS შეტყობინებების მოსმენა	101
2.26. მეორე თავის დასკვნა	102
III თავი. ლოკაცია და რუკა	103
3.1. Android Location API	103
3.2. GPS (Global Positioning System) გამოყენება	104
3.3. GPS-ის გამოყენება აპლიკაციაში	104
3.4. მოწყობილობის ლოკაციის დადგენა	105
3.5. ლოკაციის სერვისების გამოყენება Emulator-ში	109
3.6. GeoCoding	109
3.7. მეტის გაკეთება Location-Based სერვისებისგან	113
3.8. Google Location Services API	113
3.9. Fused Location Provider	114
3.10. მეტის გაკეთება Google Location Services-ით	115
3.10.1. მოქმედების ამოცნობა	115
3.10.2. Geofencing API	116
3.11. Google Maps Android API v.2	116
3.11.1. ლოკაციის მაპინგი	117
3.11.2. Mapping Intents	117
3.11.3. რუკის API KEY-ს მოპოვება	119
3.11.4. Map Fragments	121
3.12. ლოკაციის მარკირება	124
3.13. პოზიცია და რუკის ანიმაცია	124
3.14. მესამე თავის დასკვნა	126
IV თავი. მობილური აპლიკაციის აგება ექსტრემალური სიტუაციებისთვის 112-ის მაგალითზე	127
4.1. გადაუდებელი საჭიროების (დახმარების) სიტუაციებში 112-თან დაკავშირების არხები	127
4.1.1. 112-თან დაკავშირება ხმოვანი ზარის საშუალებით..	128
4.1.2. SMS და ვიდეო ზარი	131
4.1.3. მობილური აპლიკაცია	132

4.1.3.1. პრობლემის აღწერა	132
4.1.3.2. მთავარი მიზანი	132
4.1.3.3. სასურველი შედეგები	134
4.1.3.4. აპლიკაციის ფუნქციები	134
4.1.3.5. აპლიკაციის უპირატესობანი და ნაკლევანებები ...	149
4.2. მუშაობა background არხებში	149
4.2.1. სერვისები	150
4.2.2. სერვისის შექმნა და მათი კონტროლი	151
4.2.3. სერვისის ამუშავება და მისი გადატვირთვის კონტროლი	153
4.2.4. სერვისის ჩართვა და გათიშვა	155
4.2.5. თვითშეჩერებადი სერვისები	156
4.2.6. სერვისის და Activity-ს დაკავშირება	157
4.2.7. Foreground სერვისების შექმნა	160
4.2.8. Background Thread-ების გამოყენება	162
4.2.9. AsyncTask ასინქრონული დავალებების შესასრულებლად	163
4.2.10. AsyncronousTask-ის შექმნა	164
4.2.11. ასინქრონული ოპერაციების გაშვება	166
4.2.12. Intent სერვისი	166
4.2.13. Alarm-ის გამოყენება	168
4.2.14. Alarm-ის შექმნა კონფიგურირება და გაუქმება	168
4.3. მეოთხე თავის დასკვნა	170
ლიტერატურა	171

შესავალი

XXI საუკუნე, მსოფლიო გლობალიზაციის ფონზე, სამეცნიერო-ტექნიკური პროგრესის სწრაფი ტემპებით ვითარდება, რაც განპირობებულია როგორც ახალი ტექნიკისა და ტექნოლოგიების შექმნითა და დანერგვით სხვადასხვა სფეროში, ასევე ინტერნეტული და მობილური სისტემების გამოყენების არეალის გაფართოებით.

UNESCO-ს რეკომენდაციების და ევროპული ქვეყნების უნივერსიტეტების საგანმანათლებლო პროგრამების ანალიზის საფუძველზე შეიძლება ითქვას, რომ *ინფორმაციული საზოგადოების* ფორმირება და მისი განვითარება გარდაუვალი პროცესია. მისი ერთ-ერთი დამაჩქარებელი კატალიზატორი გახდა „კოვიდ-19“-პანდემიის მოვლენები, რომლის შედეგია მსოფლიოში მრავალი დაწესებულების (სკოლები, უნივერსიტეტები, სახელმწიფო და კერძო სტრუქტურათა ორგანიზაციები) გადასვლა „ონლაინ“-რეჟიმში სასწავლად ან სამუშაოდ.

ასეთმა პროცესებმა გამოიწვია ინფორმაციული და კომუნიკაციური ტექნოლოგიების სფეროში დასაქმებული ადამიანების რაოდენობის მკვეთრი ზრდა, როგორც ინფორმაციული სისტემებისა და ტექნოლოგიების სხვადასხვა სფეროში გამოყენების მიზნით, ასევე მათი შექმნისა და განვითარების თვალსაზრისით. ამიტომაც დიდი ყურადღება ექცევა ინტერნეტისა და მობილური ტექნოლოგიების ახალი სასწავლო პროგრამების შექმნას ამ მიმართულებით, რაც განსაკუთრებით აქტუალური და მოთხოვნადია მოზარდი

თაობისა და სტუდენტების წრეებში. ჩნდება ახალი პროფესიები და სამუშაო ადგილები, იქმნება კვალიფიკაციის ამაღლებისა და გადამზადების ცენტრები და პროგრამები.

წინამდებარე წიგნი, როგორც მონოგრაფიული სახელმძღვანელო, გამიზნულია ინფორმატიკის სპეციალობის სტუდენტებისათვის მობილური აპლიკაციების პროგრამირების საფუძვლების შესასწავლად.

პირველ თავში განხილულია მობილური დაპროგრამების მეთოდოლოგია და ძირითადი პრინციპები Android ოპერაციული სისტემისათვის. გადმოცემულია მობილური აპლიკაციის სასიცოცხლო ციკლის ეტაპები, სისტემის ბიზნეს-მოთხოვნების ფორმირების, დაპროექტების/დიზაინის (UI/UX), პროგრამული დეველოპმენტის (Java და Kotlin ენები), ტესტირების, დანერგვისა და თანხლების პროცესების გათვალისწინებით.

მეორე თავი ეხება მობილური აპლიკაციების აგებისა და გამოყენების საკითხებს ტელეფონის სფეროში, კერძოდ, მობილურებში შემავალი და გამავალი ზარების და SMS/MMS-ების მონიტორინგისთვის Intent-ების გამოყენებას.

მესამე თავში აღწერილია ანდროიდის სისტემის რესურსების გამოყენება ლოკაციის სერვისებისა და რუქის ანიმაციისათვის, GPS-ის და Google Maps Android API-ის გამოყენება მობილურ აპლიკაციაში.

მეოთხე თავი წიგნის ორიგინალური მხარეა, რომელშიც ასახულია მობილური აპლიკაციის პრაქტიკული რეალიზაცია კონკრეტულ სფეროში, კერძოდ, „112“-ში, გადაუდებელი დახმარებისა და საგანგებო სიტუაციების მართვის პროცესების სრულყოფის მიზნით.

I თავი. მობილური აპლიკაციების პროგრამული პლატფორმები და ენები

1.1. შესავალი Android ოპერაციულ სისტემაში

1.1.1. რა არის Android ?

Android არის ოპერაციული სისტემა და პროგრამირების პლატფორმა, რომელიც შექმნილია კომპანია Google-ის ფინანსური მხარდაჭერით, ძირითადად სენსორული მობილური მოწყობილობებისთვის, როგორიცაა სმარტფონები და ტაბლეტები. მას შეუძლია ფუნქციონირება სხვადასხვა მწარმოებლის განსხვავებული ტიპის მოწყობილობებზე. იგი შეიცავს პროგრამული უზრუნველყოფის დამუშავების საშუალებებს შესაბამისი კოდისა და აპლიკაციის შესაქმნელად, Android ოპერაციული სისტემის მომხმარებლებისათვის [1-3].

Android აპლიკაციების შექმნისას პროგრამირების ენა Java ერთგვარად შეუცვლელია, რადგან ის ბუნებრივ ენად ითვლება Android ოპერაციულ სისტემაზე მომუშავე მოწყობილობებისათვის. აქედან გამომდინარე, Java ენაზე Android აპლიკაციის შექმნისას, შესაძლებელია ყველა იმ ფუნქციონალობის სრულფასოვნად გამოყენება, რომელსაც Android ოპერაციული სისტემა იძლევა. დღეს ანდროიდის ყველა აპლიკაცია თავმოყრილია Google Play-ზე [1].

1.1.2. რატომ უნდა შევქმნათ აპლიკაციები Android-ზე ?

მიმდინარე ტექნოლოგიური განვითარების შეუქცევადმა პროცესმა, თანამედროვე მსოფლიოში, საზოგადოება ახალი გამოწვევების წინაშე დააყენა. დღეისათვის ინფორმაციული ტექნოლოგიების ეფექტიანად გამოყენების

გარეშე, ნებისმიერ სფეროში, მნიშვნელოვანი პროგრესის მიღწევა პრაქტიკულად წარმოუდგენელია.

ყველა ის სიკეთე რაც ტექნოლოგიურმა განვითარებამ მოგვითანა, მოწყობილობები, პროგრამები, შემუშავებულია და გამოიყენება სხვადასხვა მიზნის განსახორციელებლად და პრობლემის გადასაწყვეტად, როგორცაა, მაგალითად: ბიზნესის მოთხოვნების მოგვარება, ტურიზმის განვითარება, ახალი სერვისების შექმნა, თამაშების, სხვადასხვა ტიპის შინაარსის აპლიკაციების მიწოდება მომხმარებლებისთვის და სხვ. [1]

მოგეხსენებათ, რომ ელექტრონული კომერციის ბაზარი ელვის სისწრაფით ვითარდება, იქმნება ახალი კომპანიები, მათ შორის სტარტაპები, რის შედეგადაც ბაზარზე კონკურენციამ ახალ სიმაღლეებს მიაღწია. აქედან გამომდინარე, ყველა კომპანიისთვის გახდა აუცილებელი, რომ ჩაერთონ ინტერნეტში და სწრაფად მიაწვდინონ ხმა უფრო და უფრო მეტ მომხმარებელს.

დღეს თითქმის ყველა ადამიანი ფლობს სმარტფონს. აქედან გამომდინარე, მობილური პროგრამები გახდა მოსახერხებელი გზა, დაუკავშირდე უფრო მეტ ადამიანს და განავითარო ურთიერთობები ვირტუალურ მომხმარებლებთან და კლიენტებთან.

როდესაც საუბარია აპლიკაციის შექმნაზე, ამ დროს განიხილება ორი შესაძლებლობა:

- Android და
- iOS.

ეს ორი უალტერნატივო გადაწყვეტა ხდება მათი პოპულარობის, ეფექტიანობისა და მომხმარებლის კმაყოფილების გათვალისწინებით. თუმცა, საერთო დილემა, რომლის წინაშეც უმეტესი ადამიანი დგება, არის ის, თუ რომელი ოპერაციული სისტემა უნდა აირჩეს – Android თუ iOS.

პირველი მარტივი გამოსავალი, რომელზეც შეიძლება იფიქროთ, არის ორივე პლატფორმის ერთდროულად შემუშავება, მაგრამ ამისთვის ყველას საკმარისი ბიუჯეტი არ აქვს. ასევე არსებობს ამ პრობლემის ცალსახა გადაწყვეტა: „ეს ყველაფერი დამოკიდებულია არჩევანზე (Choices)” [1,3].

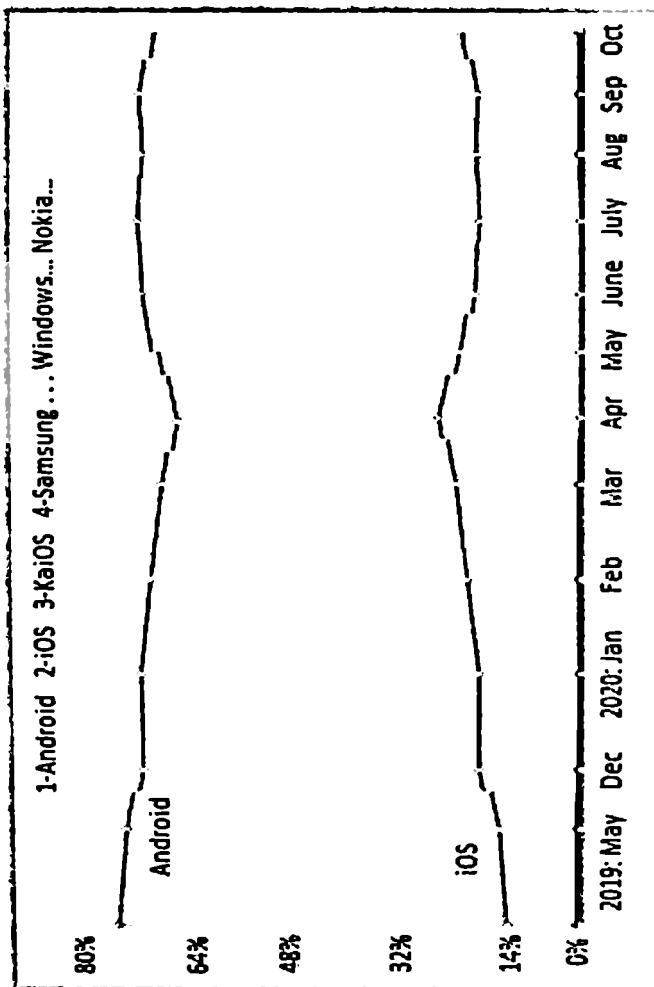
დღეისათვის მრავალი პლატფორმის გამოყენებაა შესაძლებელი, თუმცა დეველოპერები ირჩევენ Android-ს იმისათვის, რომ შეძლონ მობილურ მოწყობილობათა უმრავლესობაზე წვდომა.

სანამ ამ საკითხს უფრო მეტად ჩავუღრმავდებით, პირველ რიგში მივიღოთ პასუხი - რატომ Android?

Android-მა შეძლო გლობალური ბაზრის თითქმის 75%-ის დაპყრობა, შესაბამისად მის კონკურენტს iOS სისტემას გაცილებით მცირე წილი უჭირავს (ნახ.1.1) [2].

ასევე, Android-ის სხვა მრავალ მახასიათებელთან ერთად, მის წარმატებას ხელს უწყობს ამ სისტემის გონივრულობა და ეფექტიანობა.

Android დღეს ყველაზე წარმატებული და ძლიერი მობილური ოპერაციული სისტემაა ბაზარზე. ცხადია, ის ლიდერობს, როდესაც საქმე დიდ მასშტაბებს ეხება.



ნახ. 1.1 მობილური ოს-ის გამოყენების სტატისტიკა
შოფლიოში (11.2020) [2]

1.1.3. Android ვერსიები

დღეის მონაცემებით Google-ს აქვს შექმნილი Android-ის სხვადასხვა ვერსია. აქამდე კომპანია დესერტების სახელებს არჩევდა თავის Android ვერსიებისთვის. მაგრამ ეს შეიცვალა და ბოლო ვერსიას მხოლოდ Android-10 დაარქვა.

შეგახსენებთ, რომ ეს ყველაფერი Cupcake-ით დაიწყო და მას, შემდეგ ბევრი სხვაც მოჰყვა - Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream, Jellybean, KitKat, Lollipop, Marshmallow, Nougat, Oreo და ბოლოს ეს იყო, Pie (ცხრილი.1.1) [5].

ჩნდება კითხვა, რატომ გადაწყვიტა Google-მა Android-ის დასახელების პროცესის რესტრუქტურისა?

კომპანიამ ეს გადაწყვეტილება დაბნეულობის თავიდან ასაცილებლად მიიღო. Google-ში სჯერათ, რომ Android 10-ის სახელი იქნება უფრო მეტად „წმინდა და ნათელი“ ყველასთვის. ოფიციალურად მათ ეს გადაწყვეტილება ასე განმარტეს: „როგორც გლობალური ოპერაციული სისტემა, მნიშვნელოვანია, რომ ეს სახელები იყოს ნათელი და რელევანტური ყველასთვის მთელ მსოფლიოში. ასე რომ, Android-ის შემდეგ გამოშვების სახელში უბრალოდ გამოიყენება ვერსიის ნომერი და დაერქმევა Android 10 [6].

ვფიქრობთ, ეს ცვლილება გაამარტივებს სახელს და უფრო მეტად გასაგები და ინტუიციური იქნება ჩვენი მსოფლიო საზოგადოებისთვის. მიუხედავად იმისა, რომ მრავალი „მაცდური და გემრიელი დესერტის“ სახელი იყო გამოყენებული ჩვენი ვერსიების დასახელებებში, სხვა

ქვეყანებში რთულად გასაგები ხდებოდა. სწორედ ამიტომ მივიღეთ გადაწყვეტილება, მე-10 ვერსიაში აქტიური 2.5 მილიარდი მოწყობილობისთვის ეს ცვლილება განგვეხორციელებინა” [4,6].

Android ვერსიები

ცხრ.1.1

კოდური სახელი	ვერსიის საშუალო	გამოშვების თარიღი	API დონე
No codename	1.0	September 23, 2008	1
	1.1	February 9, 2009	2
Cupcake	1.5	April 27, 2009	3
Donut	1.6	September 15, 2009	4
Eclair	2.0 – 2.1	October 26, 2009	5 – 7
Froyo	2.2 – 2.2.3	May 20, 2010	8
Gingerbread	2.3 – 2.3.7	December 6, 2010	9 – 10
Honeycomb	3.0 – 3.2.6	February 22, 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	October 18, 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	July 9, 2012	16 – 18
KitKat	4.4 – 4.4.4	October 31, 2013	19 – 20
Lollipop	5.0 – 5.1.1	November 12, 2014	21 – 22
Marshmallow	6.0 – 6.0.1	October 5, 2015	23
Nougat	7.0 – 7.1.2	August 22, 2016	24 – 25
Oreo	8.0 – 8.1	August 21, 2017	26 – 27
Pie	9.0	August 6, 2018	28
Android 10	10.0	September 3, 2019	29

გარდა ამისა, Google-მა ასევე შეცვალა Android-ის ლოგოს სახე. კომპანია განმარტავს ბრენდის ლოგოს შეცვლის მიზეზს და ამბობს: „ლოგოს დიზაინი შთაგონებას იღებდა საზოგადოების ყველაზე ცნობადი არაადამიანური წევრიდან, Android რობოტისგან. რობოტი საზოგადოებაში ყველას ეკუთვნის. ის დიდი ხანი იყო Android-ის მხიარულების და ცნობისმოყვარეობის სიმბოლო. ახლა კი მას ჩვენს ლოგოში განსაკუთრებული ადგილი უკავია. ჩვენ ასევე შევცვალეთ ლოგო მწვანედან შავში“.

Android-10 აგებულია სამი მნიშვნელოვანი იდეის საფუძველზე [7,8]:

1) Android-10 აყალიბებს მობილური ინოვაციის წამყვან ზღვარს თანამედროვე მანქანათმცოდნეობით და მხარს უჭერს განვითარებად მოწყობილობებს, როგორიცაა დასაკეცი და 5G ტელეფონები (ნახ.1.2);

2) ძირითადი ყურადღება გადატანილი აქვს კონფიდენციალურობასა და უსაფრთხოებაზე. შემუშავებული თითქმის 50 მახასიათებელი, რაც მომხმარებლებს უფრო მეტ დაცვას, გამჭვირვალობასა და კონტროლს აძლევს;

3) Android-10 აფართოებს მომხმარებელთა ციფრული კეთილდღეობის კონტროლს, რაც იმის წინაპირობაა, რომ ინდივიდებსა და ოჯახებს შეუძლიათ იპოვნონ უკეთესი ბალანსი ტექნოლოგიასთან [11].



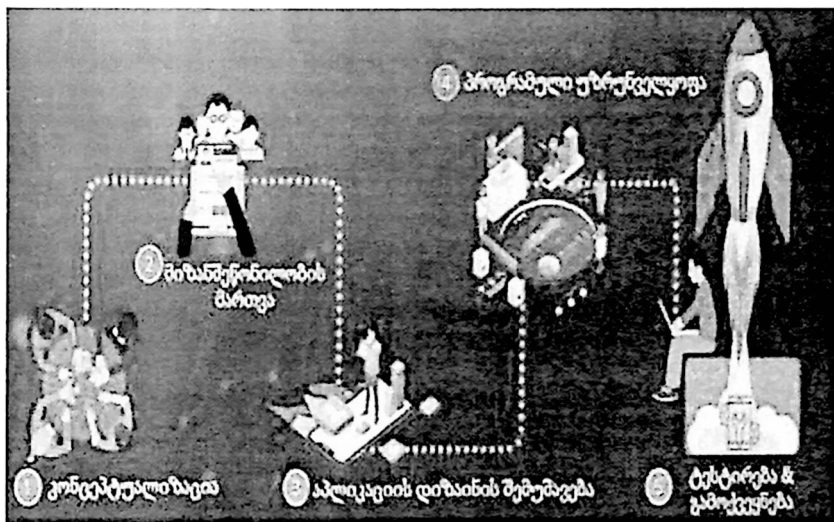
ნახ.1.2. Android-10 -ის საშუალებით ოპტიმიზებული აპლიკაცია დასაკვეთ გერანზე

1.2. Android-ის პირველი პროგრამა

1.2.1. დეველოპმენტის პროცესი

Android აპლიკაციის პროექტი იწყება იდეით და ამ იდეის რეალიზაციისთვის აუცილებელი მოთხოვნების განსაზღვრით [4, 9]. პროექტზე მუშაობს გუნდი და დასაწყისიდან დასასრულამდე გადის 5 ძირითად ეტაპს (ნახ.1.3):

- 1) კონცეპტუალიზაცია (სტრატეგიის შემუშავება);
- 2) მიზანშეწონილობის შეფასება (ანალიზი, დაგეგმვა);
- 3) აპლიკაციის დიზაინის შემუშავება (UI/UX);
- 4) პროგრამული უზრუნველყოფა (App Development);
- 5) ტესტირება და გამოქვეყნება (დანერგვა, თანხლება).



ნახ 1.3. 5-ბოჯი მობილური აპლიკაციის
შესაქმნელად

1.2.2 კონცეპტუალიზაცია – Android პროგრამის განვითარების (დეველოპმენტის) პროცესში პირველი და ყველაზე მნიშვნელოვანია სწორი კონცეფციის და მისი რეალიზაციის მოთხოვნების ჩამოყალიბება. აპლიკაციის საწყისი ანალიზი უნდა შეიცავდეს სამიზნე ჯგუფის განსაზღვრას, ქცევის მოდელებს და მიზნებს. პროგრამის წარმატებით შექმნა დამოკიდებულია აღნიშნულ მახასიათებლებზე. ამ ეტაპზე უნდა ჩამოყალიბდეს ყველა საჭირო საფუძველი პროცესის შემდეგი განვითარებისთვის.

პროექტის წარმატებით განსახორციელებლად აუცილებელია გათვლების გაკეთება და გონებრივი შეტევის (Brainstorming) მეთოდის გამოყენება, რათა მაქსიმალურად იყოს შესაძლებელი ყველა შემდგომი გართულების პრევენცია. კიდევ ერთი მნიშვნელოვანი ნაწილი, რაც უნდა შესრულდეს, არის კონკურენტების ანალიზი, იმის გასარკვევად თუ რა მახასიათებელს შეუძლია ჩვენი აპლიკაცია გახადოს ბაზარზე წარმატებული.

1.2.3. მიზანშეწონილობის შეფასება – აპლიკაციაზე მომუშავე გუნდმა უნდა განსაზღვროს თუ რამდენად მიზანშეწონილია პროექტის განხორციელება. უკვე კონცეპტუალიზებული პროდუქტის კარგასული მოდელის და დეტალური ესკიზების საფუძველზე, შესაძლებელია აპლიკაციის კონცეფციის ტექნიკურად რეალიზებადობის მკაფიოდ დანახვა.

მიზანშეწონილობის ტესტირების დასრულებისას გუნდს შეიძლება ჰქონდეს სრულიად განსხვავებული იდეა თუ ორიგინალი ფუნქციონალურობა შეუძლებელია.

1.2.4. აპლიკაციის დიზაინის შემუშავება – მომხმარებლის ინტერფეისის დიზაინის ფორმირება არის საკმაოდ რთული პროცესი და მოიცავს სხვადასხვა ღონისძიებას. უნდა შეიქმნას აპლიკაციის დიზაინი, რომელიც ინიცირებულია მომხმარებლის გამოცდილებიდან, სადაც გათვალისწინებული იქნება მრავალი კონცეფცია და მოსაზრება (UI/UX) [9,10]. აუცილებელია სამუშაო ჯგუფის მიერ შემუშავდეს ინდივიდუალური ფორმა, მაგრამ მასში საჭიროა დაცულ იქნას მარტივი დიზაინური აღნიშვნები, რათა მომხმარებელში არ გამოიწვიოს გაურკვევლობა რეალურ მნიშვნელობაზე.

1.2.5. პროგრამული უზრუნველყოფა – პროგრამირების ენა Java ერთგვარად შეუცვლელია Android აპლიკაციების შექმნისას, რადგან Android ოპერაციულ სისტემაზე მომუშავე მოწყობილობებისათვის იგი ბუნებრივ ენად ითვლება. აქედან გამომდინარე, Java ენაზე Android აპლიკაციის შექმნისას, შესაძლებელია ყველა იმ შესაძლებლობის სრულფასოვნად გამოყენება, რომელსაც Android ოპერაციული სისტემა გვაძლევს.

პროგრამული უზრუნველყოფა შედგება ერთი ან მეტი ქმედებისგან. სამუშაოს მიმდინარეობისას Android-ისთვის აპლიკაციის შემუშავება კონცეფციურად იგივეა, როგორც სხვა აპლიკაციების პლატფორმებში. ამ ყველაფერთან ერთად კარგად შემუშავებული აპლიკაციის ეფექტურად მოსაწყობად, საჭიროა რამდენიმე სპეციალური საშუალება.

შემდეგ ჩამონათვალში მოცემულია მიმოხილვა Android პროგრამის შექმნის პროცესზე და მოიცავს Android Studio

პლატფორმის გადმოსაწერ ბმულს, რომელიც გამოიყენება დეველოპმენტის ყოველი ეტაპის განმავლობაში [1]:

1) შეიქმნას სამუშაო ადგილი – დაინსტალირდეს Android Studio და შეიქმნას პროექტი;

2) დაიწეროს აპლიკაცია – Android Studio მოიცავს მრავალფეროვან ხელსაწყოებს და ინტელექტუალური კოდების რედაქტორს, რომელიც ეხმარება პროგრამისტს უკეთესი და ხარისხიანი კოდის დაწერაში, იყოს იგი უფრო პროდუქტიული და სწრაფი, უზრუნველყოს კოდის დასრულება Kotlin, Java და C/C ++ ენებისათვის.

1.2.6. ტესტირება და გამოქვეყნება – აპლიკაციის დეველოპმენტის პროცესის ერთ-ერთი მნიშვნელოვანი კომპონენტია ტესტირება ადრეულ ეტაპზე, ინტერფეისის და უსაფრთხოების შემოწმება, თავსებადობა სხვადასხვა მოწყობილობაზე, წარმადობა და ა.შ. შეცდომების დაფიქსირების და აღმოფხვრის შემდეგ პროგრამა გადადის გამოქვეყნების ეტაპზე. სხვადასხვა application stores (მაღაზიებს) აქვს გამოქვეყნების განსხვავებული პოლიტიკა და გამოქვეყნების ფაზის გეგმა უნდა შეესაბამებოდეს app store -ებს.

თუმცა Android პროგრამის დეველოპმენტის სერვისი აპლიკაციის გამოქვეყნებით არ მთავრდება. როგორც თუ არა თქვენი შექმნილი აპლიკაცია მიაღწევს მომხმარებლების ხელს, დაიწყება უკუკავშირი, ხოლო აპლიკაციის შემქმნელებს დასჭირდება ამ უკუკავშირის გაანალიზება რათა განავითარონ პროგრამის მომავალი ვერსიები. როგორც კი აპლიკაციის პირველი ვერსია ამოწურავს თავის რესურსს, მისი განვითარების ციკლი ახალ ეტაპზე გადადის, საჭიროა მახასიათებლების დახვეწა და მისი მორგება თანამედროვე

გამოწვევებზე. აქ საუბარია გრძელვადიანი ვალდებულების აღებაზე, რომელიც მოითხოვს პროდუქტის განვითარებაში მნიშვნელოვანი მატერიალური თუ ადამიანური რესურსის ინვესტირებას. წინააღმდეგ შემთხვევაში აპლიკაცია დაკარგავს სიცოცხლისუნარიანობას [12].

1.3. Android Studio-ს ინსტალაცია

Android Studio არის ოფიციალური ინტეგრირებული დეველოპმენტის გარემო Google-ის Android ოპერაციული სისტემისთვის, რომელიც შექმნილია JetBrains-ის გუნდის მიერ IntelliJ IDEA პროგრამულ უზრუნველყოფაზე დაყრდნობით სპეციალურად Android-ის განვითარებისათვის. მისი ინსტალირება საკმაოდ მარტივია და შესაძლებელია Windows, macOS, Linux ოპერაციულ სისტემებზე [13].

ბიჯი 1: გადადით ამ ბმულზე და გადმოიწერეთ Android Studio ფაილი.

ბიჯი 2: ჩამოტვირთეთ android studio (ნახ.1.4).

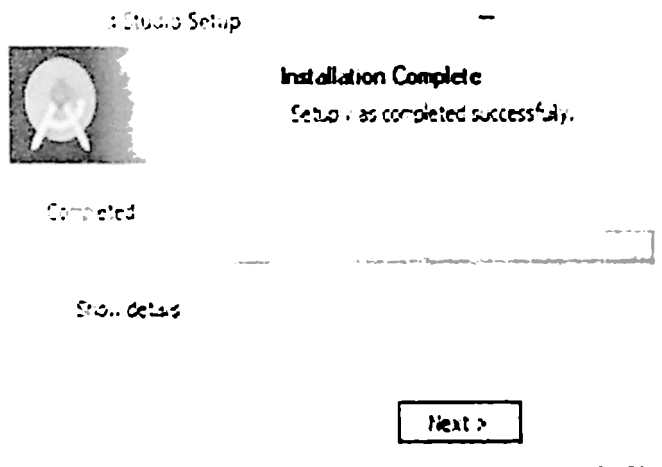
android studio



ნახ.1.4

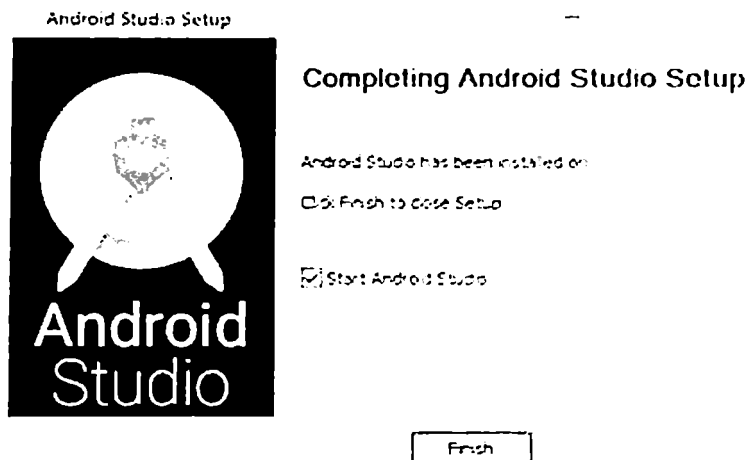
შემდეგ ეტაპზე უნდა ავირჩიოთ ანდროიდის სტუდიოს რომელი კომპონენტის ინსტალაცია გვსურს (Android Studio, Android Virtual Device). გადავდივართ შემდეგ პოზიციაზე სადაც შეგვიძლია განვსაზღვროთ ადგილმდებარეობა თუ სად გვსურს პროგრამის ინსტალაცია.

ბიჯი 4: Android Studio დაიწყებს ინსტალაციას და მისი დასრულების შემდეგ მივიღებთ ფანჯარას (ნახ 1.7).



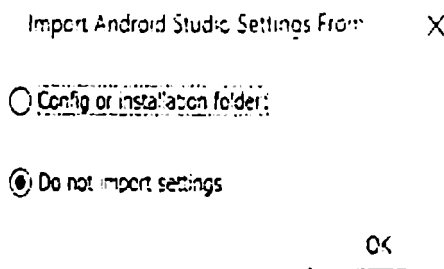
ნახ 1.7

ვირჩევთ „Next“ ღილაკს და გადავდივართ შემდეგ ფანჯარაზე, რომლითაც სრულდება Android Studio-ს ინსტალაცია (ნახ. 1.8).



ნახ.1.8

ბიჯი 5: ინსტალაციის დასრულების შემდეგ, პროგრამა იკითხავს საჭიროა თუ არა წინა პარამეტრების იმპორტი. თუ Android studio ადრე იყო დაყენებული შესაძლოა დაგვჭირდეს, თუ არა, აირჩიოთ "Do not import settings" (ნახ. 1.9).



ნახ.1.9

ბიჯი 6: შემდეგი დაიწყება Android Studio-ს გაშვება.



ნახ.1.10

ამავდროულად, ის პოულობს ხელმისაწვდომ SDK (პროგრამული უზრუნველყოფის დამუშავების ნაკრები) კომპონენტებს და იწყებს გადმოწერას (ნახ.1.11).

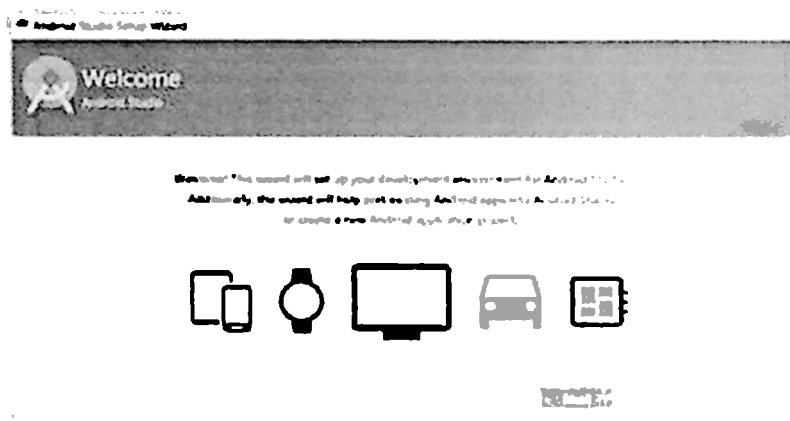
Finding Available SDK Components

Downloading...

<https://dl.google.com/android/repository/repository2-1.xml>

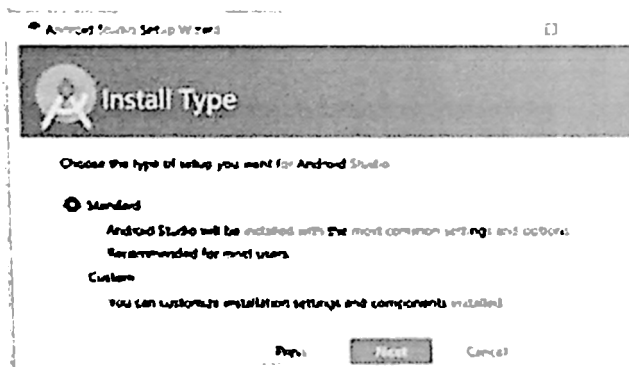
ნახ.1.11

ბიჯი 7: მას შემდეგ, რაც ის მოიპოვებს SDK კომპონენტებს, ის გადამისამართდება Welcome-ის დიალოგურ ფანჯარაზე და ვაკლიკებთ next ღილაკზე (ნახ. 1.12).



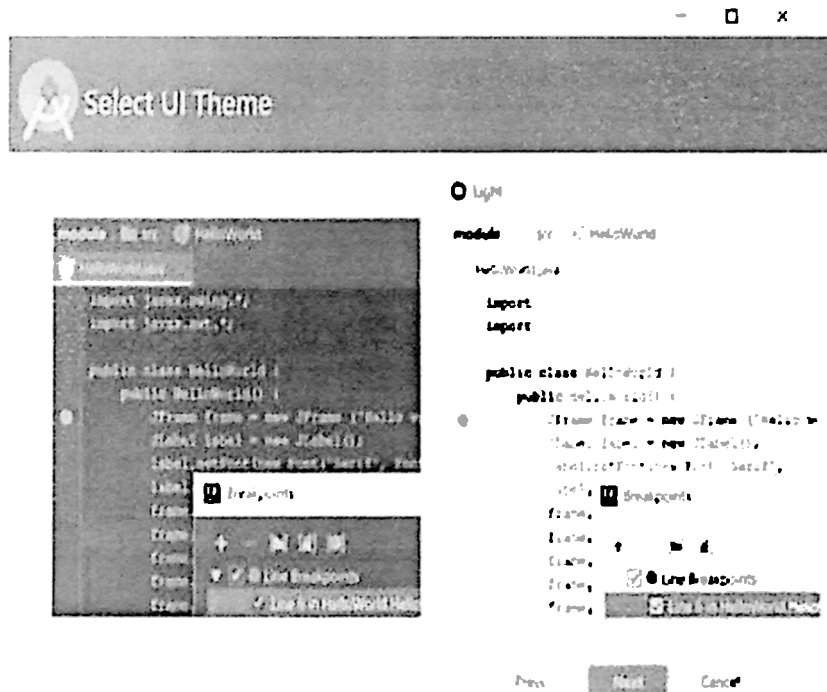
ნახ. 1.12. Welcome დიალოგური ფანჯარა

შემდეგ გამოდის Instal Type-ს დიალოგური ფანჯარა, ვირჩევთ Standard და next ღილაკს (ნახ 1.13).



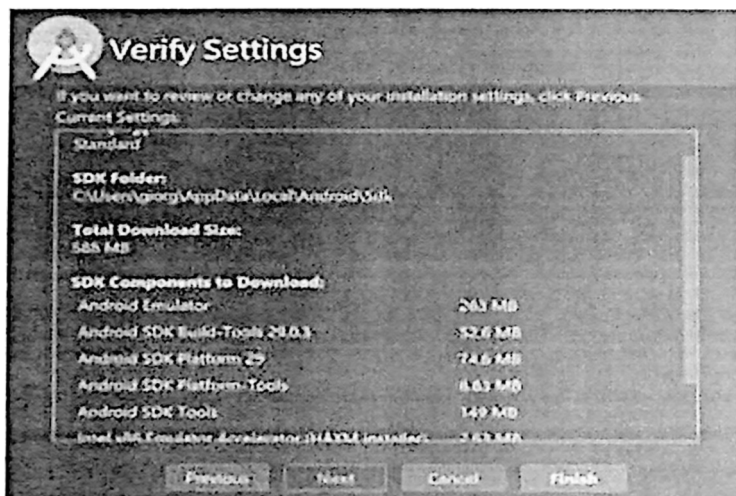
ნახ.1.13

ახლა უნდა ავირჩიოთ Android Studio -ს სამუშაო გარემოს თემა, რომელიც შესაძლოა იყოს ნათელი ან ბნელი და ვაკლიკებთ next ღილაკს (ნახ. 1.14).

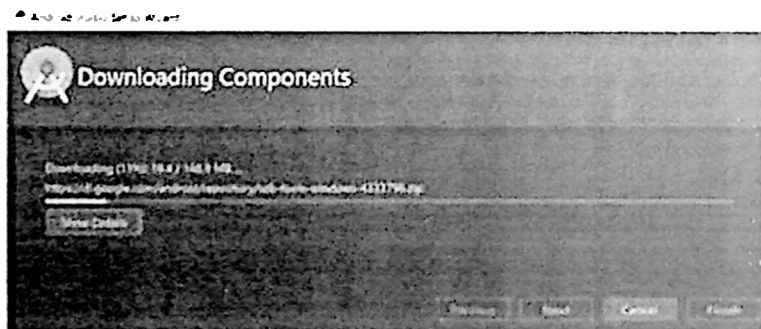


ნახ.1.14

ბიჯი 8: ახლა დროა ჩამოვტვირთოთ SDK კომპონენტები. ვირჩევთ Finish ღილაკს და იწყებს კომპონენტების ჩამოტვირთვას (ნახ. 1.15, 1.16).

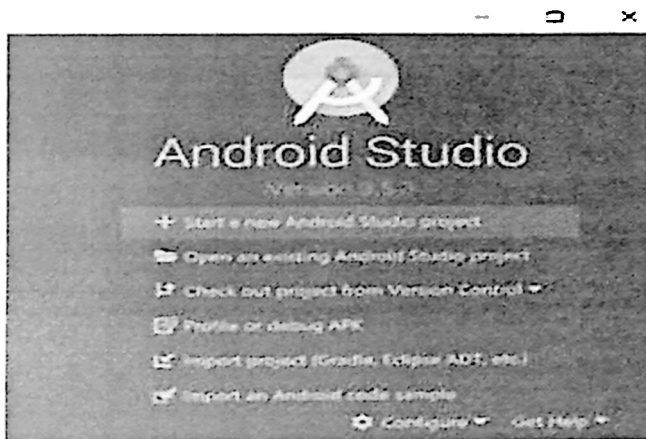


ნახ.1.15



ნახ.1.16

ბიჯი 9: Android Studio წარმატებით დაინსტალირდა, შეიძლება „Start new android project“ არჩევა და პირველი აპლიკაციის შექმნის დაწყება (ნახ. 1.17).



ნახ.1.17

1.4. პირველი Android პროექტის შექმნა „Hello World app“

Android Studios ინსტალაციის შემდეგ, სამუშაო გარემოს შესამოწმებლად და მისი გაცნობის მიზნით, კარგი მეთოდია „Hello Word“-ის მაგალითის განხორციელება. პირველ ეტაპზე ეკრანზე გამოვიტანთ რაიმე მარტივ შეტყობინებას. იგი გვადლევს იმის შესაძლებლობას, რომ დავრწმუნდეთ სწორად მუშაობს თუ არა ჩვენი სამუშაო გარემო და მზად ვართ თუ არა რეალური პროექტების შესაქმნელად.

ჩვენი პირველი აპლიკაციის (App – აპი) შემუშავებამდე განვსაზღვროთ დეველოპმენტის პროცესის ზოგადი ბიჯები:

- 1) Android Studio პროექტის შექმნა;
- 2) აპის მომხმარებლის ინტერფეისის (UI) დაყენება;

3) UI კომპონენტების, როგორცაა ღილაკები, ტექსტური ველები და ა.შ. Java- ს კოდთან დაკავშირება;

4) კოდირება ჯავაში - პროგრამირების ნამდვილი ნაწილი;

5) პროექტის აგება: ეს ნიშნავს შემსრულებლის შექმნას (ფაილი, რომელიც რეალურად მუშაობს მოწყობილობაზე ან ემულატორზე). Android Studio ასრულებს მთელ სამუშაოს ერთი დაწკაპუნებით;

6) აპლიკაციის ემულატორზე მოსინჯვა;

7) აპის გაშვება ნამდვილ Android მოწყობილობაზე (არასავალდებულო);

8) აპის გამოქვეყნება Google Play- ზე (არასავალდებულო).

როდესაც ანდროიდ სტუდიოს პირველად გავუშვებთ, ჩვენ წარმოგვიდგება დიალოგური ფანჯარა (ნახ.1.18), სადაც გვაქვს რამდენიმე ვარიანტის არჩევის შესაძლებლობა.

i) Start a new Android Studio Project;

ii) Open an existing Android Studio project;

iii) Check out a project from a version control website (Git, Google Cloud, Mercurial, Subversion);

iv) Profile or debug APK

v) Import a project (Gradle, Eclipse ADT, etc.)

vi) Import an Android code sample

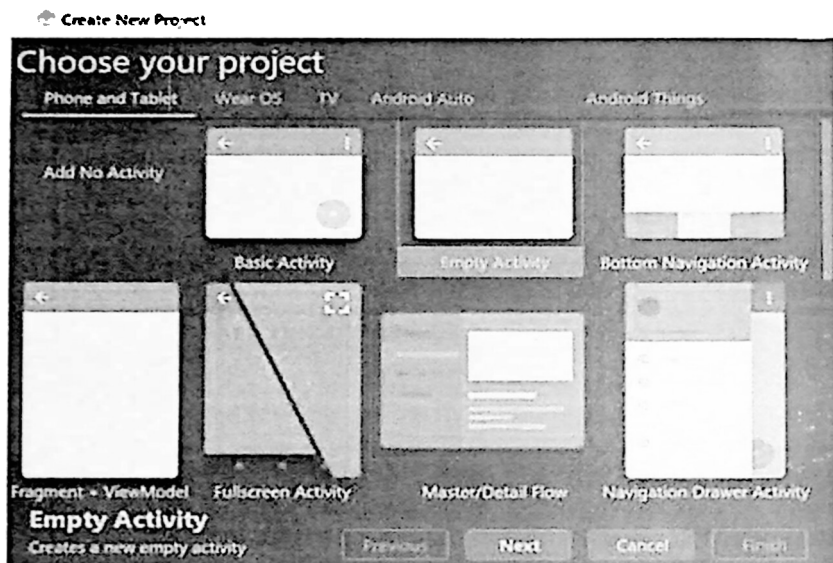
ჩვენ შევქმნათ პირველი Android პროექტი ამიტომაც ავირჩევთ “Start a new Android Studio Project.



ნახ.1.18

იმის მერე რაც ავირჩევთ ახალი პროექტის შექმნას, გამოჩნდება დიალოგური ფანჯარა სადაც უნდა განვსაზღვროთ პროექტის ეკრანის ტიპი, რომლის შექმნაც გსურს მოწყობილობის ფორმების კატალოგებიდან. არსებობს რამდენიმე შაბლონი, მათ შორის, შესვლის აქტივობა, რუკების აქტივობა და ა.შ.

აქტივობები შეიძლება განისაზღვროს, როგორც მომხმარებლის ინტერფეისით ნაჩვენები ეკრანები. ამიტომ, ჩვენ უნდა ჩავრთოთ აქტივობა, რომ გვქონდეს აპლიკაცია, რადგან მოგეხსენებათ, Android პროგრამები არის ვიზუალური პროგრამირების ნაწილი, რომლებსაც აქვთ ერთი ან მეტი მომხმარებლის ინტერფეისი. დიალოგური ფანჯარა (Choose your Project) რამოდენიმე კატეგორიად არის დაყოფილი, ესენია: Phone and Tablet, Wear OS, TV, Android Auto, Android Things. ამ კატეგორიებიდან ჩვენ ავირჩევთ პირველს, რომელსაც აქვს ძირითადი Android აქტივობა არჩეული ტელეფონისა და ტაბლეტისთვის (ნახ.1.19) [.

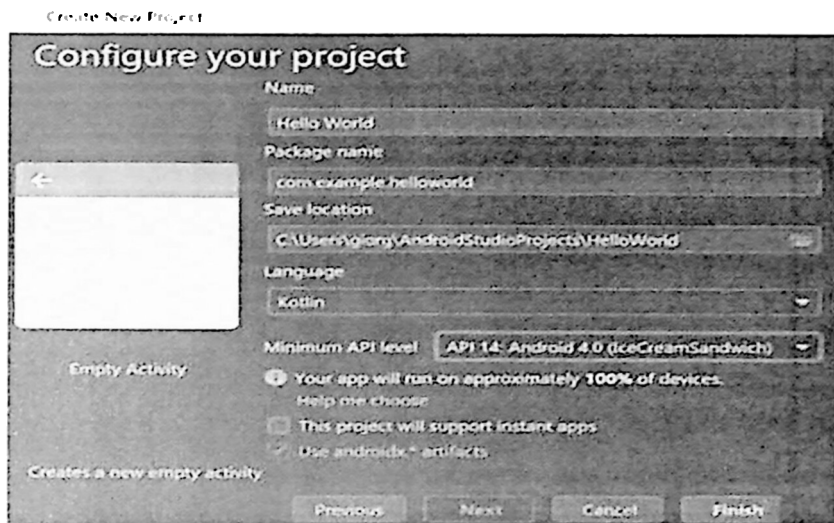


ნახ.1.19

შეარჩიეთ პროექტის ტიპი რომლის შექმნაც გსურთ, Android Studio-ს შეუძლია შეიცავდეს კოდისა და რესურსების ნიმუშს, რაც დაგეხმარებათ სრულყოფილი აპლიკაციის შექმნის დაწყებაში.

შერჩევის შემდეგ, დააჭირეთ Next.

შემდეგი ბიჯი არის ზოგიერთი პარამეტრის კონფიგურაცია და თქვენი ახალი პროექტის შექმნა (ნახ.1.20). პროექტის პარამეტრების გასასწორებლად პირველ ტექსტურ ველში (Name), ჩვენ უნდა ჩავწეროთ პროექტის სახელი, რომელიც ასევე იქნება აპლიკაციის სახელი. ამ აპლიკაციას დავარქვათ "Hello World", მაგრამ შეგიძლიათ შეიტანოთ ასევე სხვა სახელი, რომელიც თქვენ გსურთ (მისი უნიკალურობა არ არის აუცილებელი).



ნახ.1.20

პაკეტის სახელში (Package name) მოცემულია შემდეგ ტექსტის ჩასაწირი სივრცე. ეს არის ვებ – მისამართის მსგავსი სტრიქონი, რომელიც გამოიყენება Google Play ბაზარზე დეველოპერების ერთმანეთისგან გასარჩევად. აქ შეგიძლიათ გამოიყენოთ ნებისმიერი უნიკალური სახელი. თუ არ აპირებთ თქვენი აპლიკაციის ატვირთვას Google Play-ზე (როგორც აქ), შეგიძლიათ გამოიყენოთ თქვენთვის სასურველი ნებისმიერი სახელი. შემდეგ, უნდა ავირჩიოთ მდებარეობა ჩვენ კომპიუტერში (Save location), რომ შევინახოთ პროექტის ფაილები [14].

ამის შემდგომ გვაქვს ასევე ორი პარამეტრი განსასაზღვრი. პირველი პროგრამირების ენა (Language) და მეორე მინიმალური API დონე (Minimum API level).

1.5. Kotlin vs Java

რომელია უკეთესი ვარიანტი Android პროგრამების დეველოპმენტისთვის Kotlin თუ Java ? განვიხილოთ ეს საკითხი უფრო დეტალურად [15].

Kotlin მობილური აპლიკაციების დეველოპმენტის საზოგადოება განაგრძობს ზრდას. ჯერ კიდევ 2017 წელს Google-მა დაადასტურა Kotlin, რითაც იგი Android ენის მეორე ოფიციალური ენა გახდა. მას შემდეგ პროგრამირების ენა მნიშვნელოვნად გაფართოვდა და მასზე მოთხოვნილება განსაკუთრებით გაიზარდა როგორც დეველოპერების, ასევე საწარმოო ორგანიზაციებში. Google-მა გამოაცხადა, რომ Kotlin ახლა მხოლოდ სასურველი ენაა Android აპლიკაციების შექმნისთვის, არამედ ამტკიცებს, რომ ეს არის პრაგმატული, თანამედროვე და ინტუიციური პროგრამირების ენა [15].

Kotlin არის სტატიკურად ტიპიზირებული პროგრამირების ენა Java-ს ვირტუალური მანქანისა (JVM) და JavaScript-ისთვის. Kotlin ზოგადი დანიშნულების ენაა და იძლევა ფუნქციურ შესაძლებლობებს java-სთან ურთიერთქმედების მხარდასაჭერად. Kotlin-ის პროექტის შემუშავების მიზანი იყო მწარმოებლობის ამაღლება, კოდირების გამოცდილების გაუმჯობესება, როგორც პრაქტიკული, ასევე ეფექტიანობის თვალსაზრისით.

Kotlin-ის მთავარი მიზანი შერეული ენებით პროექტების შექმნაა. კოტლინს შემოაქვს გაუმჯობესებული სინტაქსი, ასევე მოკლე გამონათქვამები და აბსტრაქციები.

Kotlin-ის გამოყენება ჯავასთან ერთად ამცირებს ჭარბ კოდს, რაც უდიდესი გამარჯვებაა Android დეველოპერებისთვის.

თანამედროვე ტექნოლოგიურ სამყაროში Kotlin-ს ბოლო პერიოდში აქვს Android დეველოპმენტში მნიშვნელოვანი ზრდა. ყველა პლატფორმაზე ხელმისაწვდომობა ყოველთვის იყო მთავარი მიზანი Kotlin-სთვის და ახლაც ასე რჩება. მრავალპროფილური პროგრამირებისთვის ინოვაციური ხედვის საფუძველია კოდის გაზიარება ყველა პლატფორმას შორის. Kotlin 1.3-ის გამოშვებით, Kotlin/Native-ის გაუმჯობესება ხელს უწყობს მრავალპლატფორმულ კონცეფციას. და ბოლოს, Android-ის დეველოპერებს შეუძლიათ გამოიყენონ ერთი ინტეგრირებული დეველოპმენტის გარემო (IDE), რათა განავითარონ Kotlin ყველა პლატფორმაზე. უახლესი გამოცემა მობილური აპების მასშტაბურობას უფრო მისაღწევს ხდის კოდის განმეორებით სარგებლობის ფასდაუდებელი დახმარებით, რაც მნიშვნელოვნად ზოგავს დროსა და ძალისხმევას უფრო რთული ამოცანებისთვის.

Kotlin-ის გამოჩენა ბაზარზე ხომ არ არის Java-ს დასასრული Android დეველოპმენტში ?

Java არის რეპუტაციული პროგრამირების ენა, ფართოდია წყაროებითა და ბიბლიოთეკებით, დეველოპერების დასახმარებლად. ერთი სიტყვით რომ ვთქვათ არცერთი ენა არ არის უმეცდომო და Java-ც კი ექვემდებარება გართულებებს, რამაც შეიძლება დეველოპერის სამუშაო მოსაწყენი გახადოს. სწორედ ამიტომ Java-ს ეკოსისტემის გასაუმჯობესებლად გამოიყენება Kotlin.

ზოგიერთ დეველოპერს სჯერა, რომ Kotlin უახლოეს წლებში გააუქმებს Java -ს Android- ის დეველოპმენტისთვის. სხვა ექსპერტები კი მიიჩნევენ, რომ kotlin და Java თანაარსებობენ ერთმანეთის გადაწონვის გარეშე.

მოკლედ, რომ ვთქვათ, ბევრი დეველოპერი ადიდებს Kotlin-ს ლაკონურად კოდის ჩაწერის შესაძლებლობისთვის. დიახ მისი ლაკონური ხასიათი უმარტივებს დეველოპერს საქმეს და ამცირებს შეცდომების დაშვების რისკს.

ქვემოთ ვიხილავთ მარტივი კალკულატორის ფუნქციას, რომელიც აგებულია Java-ზე და წარმოვადგენთ მის ალტერნატიულ ვერსიას Kotlin ენაზე.

➤ კალკულატორის ფუნქცია Java ენის გამოყენებით

```
public class ClearBridge {  
    public static double calculate (double a, String op, double b)  
throws Exception {  
    switch (op) {  
        case "add":  
            return a + b;  
        case "subtract":  
            return a - b;  
        case "multiply":  
            return a * b;  
        case "divide":  
            return a / b;  
        default:  
            throw new Exception();  
    }  
}
```

```
}
}
}
```

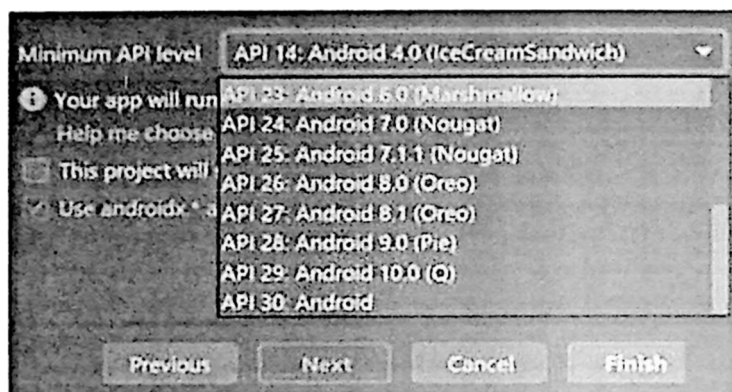
➤ კალკულატორის ფუნქცია Kotlin ენის გამოყენებით

```
fun calculate (a: Double, op: String, b: Double): Double {
    when (op) {
        "add" -> return a + b
        "subtract" -> return a - b
        "multiply" -> return a * b
        "divide" -> return a / b
        else -> throw Exception()
    }
}
```

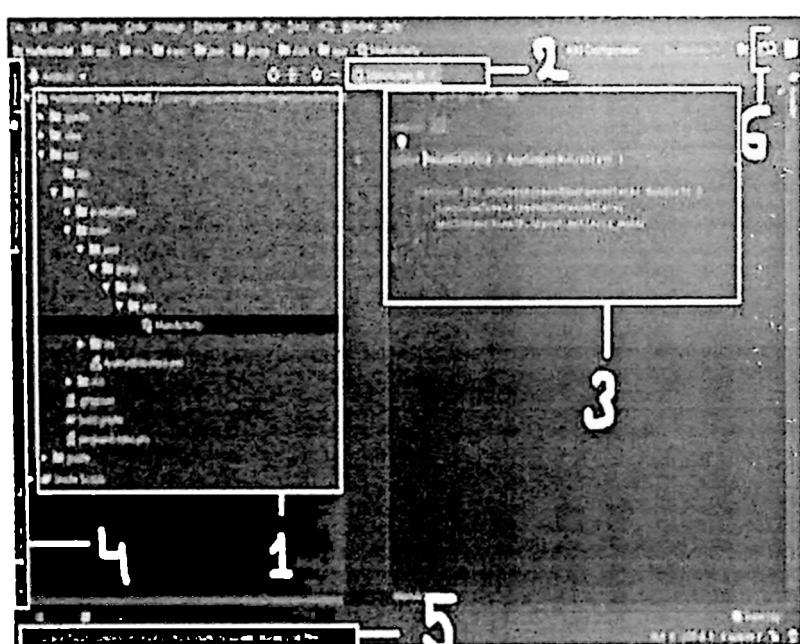
დავუბრუნდეთ ისევ მთავარ საკითხს და განვსაზღვროთ მინიმალური API დონე (Minimum API level). ამ გრაფაში ჩვენ ვირჩევთ იმ მინიმალურ Android ვერსიას, რომლის დაბალ დონეზე მყოფი ვერსია ვეღარ შემძლებს ჩვენი აპლიკაციის გაშვებას (ნახ.1.21).

ყველაფრის განსაზღვრის შემდგომ დააჭირეთ Finish ღილაკს და ამ მოცემულობით მივიღებთ ახალ Android პროექტს და გაიხსნება სამუშაო გარემო (სურ. 1.22).

Android Studio დახვეწილი ინსტრუმენტი, ამიტომ მას აქვს ათეულობით თვისება, რომ აპლიკაციების დეველოპ-მენტი გაამარტივოს. დავიწყოთ Android Studio-ს ძირითადი განყოფილებების ახსნით (ნახ. 1.22).



ნახ.1.21



ნახ.1.22

ნახაზზე მოცემულია Android Studio-ს სამუშაო გარემოს სექციები:

1-სექცია: აქ შესაძლებელია პროექტის ფაილებისა და საქალაქდებების ნახვა. გარდა ამისა, ამ ფანჯრიდან შეიძლება ახალი ფაილების დამატება. პროექტის სტრუქტურა დეტალურად განისაზღვრება შემდეგ სექციაში;

2-სექცია: გახსნილი ფაილების გააქტიურება შესაძლებელია აქ განთავსებული ჩანართებიდან შუა პანელზე;

3-სექცია: ამ შუა პანელზე შესაძლებელია აქტიური ფაილების კონტენტის ნახვა და შეცვლა. ნახაზზე ნაჩვენებია პროექტისთვის ფაილი, სახელწოდებით "MainActivity.kt". იგი არის მე-2 სექციის აქტიური ჩანართი, ამიტომ მე-3-ე სექციის შუა პანელზე მოცემულია მისი შინაარსი;

4-სექცია: ამ განყოფილებას ასევე აკონტროლებენ ჩანართების საშუალებით. დეველოპერს შეუძლია შეცვალოს პროექტის ფაილები, სტრუქტურები, ფავორიტები მარცხენა სარკმელში;

5-სექცია: აქ ნაჩვენებია მიმდინარე ან წინა კომპილაციის, აგების ან გამართვის პროცესები.

6-სექცია: ესაა Android Studio-ს გაშვების ღილაკი. როდესაც ჩვენ დავაყენებთ მომხმარებლის ინტერფეისს და ვწერთ პროექტის კოდს, ამ ღილაკის ამოქმედებით, Android Studio ქმნის მუშა პროექტს (რაც ნიშნავს შემსრულებელი ფაილის შექმნას პროექტის ფაილებიდან).

1.6. ვირტუალური მოწყობილობათა შექმნა და მართვა

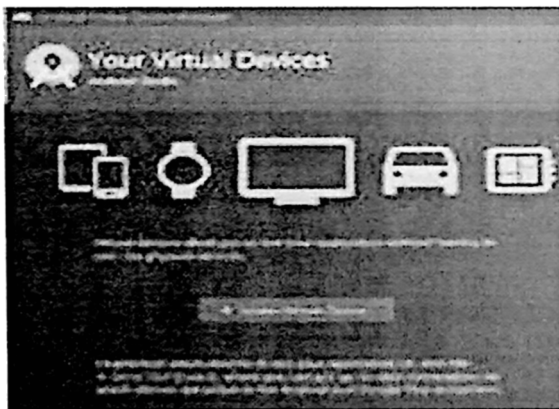
Android ვირტუალური მოწყობილობა (AVD) არის კონფიგურაცია, რომელიც განსაზღვრავს Android ტელეფონის, ტაბლეტის, Wear OS, Android TV ან Automotive OS მოწყობილობის მახასიათებლებს, რომელთა სიმულაციაც საჭიროა Android Emulator-ში. AVD მენეჯერი ინტერფეისია, რომლის გაშვება შესაძლებელია Android Studio-დან და ის გვებმარება AVD-ების შექმნასა და მართვაში [16].

AVD Manager-ის გასახსნელად გვაქვს რამდენიმე გზა:

- ვირჩივთ Tools და შემდეგ AVD Manager;
- ვირჩევთ AVD Manager-ის  ხატულას (Icon)

ინსტრუმენტების ზოლში.

AVD Manager არჩევის შემდეგ გამოდის დიალოგური ფანჯარა (ნახ.1.23) და ვქმნით ვირტუალურ მოწყობილობას.



ნახ.1.23

ვირტუალური მოწყობილობის შექმნა შედგება რამდენიმე ბიჯისაგან:

- 1) ვარჩევთ სასურველ მოწყობილობას;
- 2) ვირჩევთ ანდროიდ სისტემას, (უმჯობესია არჩეული იქნას უახლესი ვერსია);
- 3) ვარქმევთ ჩვენს მოწყობილობას სახელს და ამით სრულდება პროცესი.

1.7. მომხმარებლის ინტერფეისის შექმნა

Android Studio გვთავაზობს მომხმარებლის ინტერფეისების დიზაინის შექმნის მარტივ გზას. ფაილი სახელად "activity_main.xml", რომელიც მდებარეობს "res / layout" საქაღალდის ქვეშ, შეიცავს მიმდინარე საქმიანობის სტრუქტურის ყველა ინფორმაციას.

გავხსნათ activity_main.xml და ჩავწეროთ კოდი:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android
d="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="example.app.javatpoint.helloworld.MainActivity">
```

```
<TextView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello World!"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

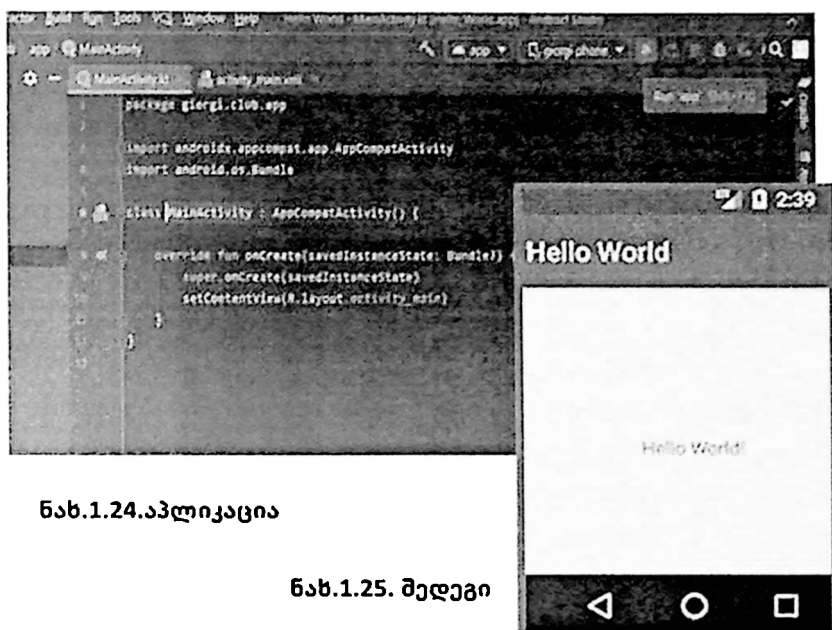
ასევე გავხსნათ **MainActivity.kt** და ჩავწეროთ კოდი:

```
package giorgi.club.app
import android.support.v7.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

შემდეგ გავუშვათ აპლიკაცია (ნახ. 1.24). შედეგი იქნები ასეთი (ნახ.1.25).



ნახ.1.24. აპლიკაცია

ნახ.1.25. შედეგი

1.8. Android ღილაკი და საილუსტრაციო მაგალითი

Android Button არის ღილაკი, რომელითაც შეგვიძლია შევასრულოთ სხვადასხვა ღონისძიებები. იგი არის UI კომპონენტი, რომელიც შედის android.widget.Button კლასში.

კოტლინის გამოყენებით, ჩვენ შეგვიძლია შევასრულოთ ღონისძიებები Android Button-ზე, თუმცა სხვადასხვა გზით:

1. **setOnClickListener**

```

button1.setOnClickListener(){
    Toast.makeText(this,"button 1 clicked", Toast.LENGTH_SHORT).show()
}
    
```

2. View.OnClickListener

```
button2.setOnClickListener(this)
```

```
    override fun onClick(view: View) {  
        // TODO("not implemented") //To change body of  
        created functions use File | Settings | File Templates.  
    }
```

3. განლაგების ფაილში ღილაკის onClick ატრიბუტის დამატება და მისი ფუნქციის განხორციელება.

```
<Button  
    android:onClick="clickButton"/>  
  
fun clickButton(v: View){  
    val mToast = Toast.makeText(applicationContext,"but  
ton 3 clicked", Toast.LENGTH_SHORT)  
    mToast.show()  
}
```

4. პროგრამულად ღილაკის შექმნა და მისი განთავსება სტრუქტურაში.

```
button4.setLayoutParams(ViewGroup.LayoutParams(Vi  
ewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.Layo  
utParams.WRAP_CONTENT))    button4.setId(button4_Id)  
button4.x = 250f  
button4.y = 500f  
button4.setOnClickListener(this)  
    constraintLayout.addView(button4)
```

➤ Android დილაკის მაგალითი

საილუსტრაციო მაგალითისათვის შევექმნით დილაკს და შევასრულებთ ქმედებებს მათი ამოქმედებით. დილაკზე დაჭერით აჩვენებს შეტყობინებას.

activity_main.xml

დამატეთ სამი დილაკი ვიჯეტების (Widget) პალიტრადან activity_main.xml სტრუქტურის ფაილში. ქვემოთ მოცემულია მისი კოდი. Id button3 დამატებულია onClick ატრიბუტი და მისი ფუნქციის სახელი ხორციელდება MainActivity კლასის ფაილში.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
xmlns:tools="http://schemas.android.com/tools"
```

```
android:layout_width="match_parent"
```

```
android:layout_height="match_parent"
```

```
android:id="@+id/constraintLayout"
```

```
tools:context="example.javatpoint.com.kotlinbutton.MainActivity">
```

```
<TextView
```

```
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
```

```
android:text="Button Action Example"
```

```
app:layout_constraintBottom_toBottomOf="parent"
```

```
app:layout_constraintLeft_toLeftOf="parent"
```

```
app:layout_constraintRight_toRightOf="parent"
```

```

        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.073"
        android:textAppearance="@style/Base.TextAppearance.
AppCompat.Medium"/>
<Button
    android:id="@+id/button1"
    android:layout_width="95dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="Button 1"
    app:layout_constraintBottom_toBottomOf="parent"

    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.501"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.498" />

```

<Button

```

    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="80dp"
    android:layout_marginEnd="8dp"

```

```
android:layout_marginStart="8dp"
android:layout_marginTop="8dp"
android:text="Button 2"
app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.762" />
```

<Button

```
android:id="@+id/button3"
android:layout_width="101dp"
android:layout_height="48dp"
android:layout_marginBottom="8dp"
android:layout_marginEnd="8dp"
android:layout_marginStart="8dp"
android:layout_marginTop="8dp"
android:onClick="clickButton"
android:text="Button 3"
app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.502"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.774" />
```

```
</android.support.constraint.ConstraintLayout>
```

შემდეგი კოდი დამატეთ MainActivity.kt კლასში.

```
package giorgi.club.app
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*
class MainActivity : AppCompatActivity() , View.OnClickListener {
    val button4_Id: Int = 1111
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        button1.setOnClickListener(){
            Toast.makeText(this,"button 1 clicked", Toast.LENGTH_SHORT).show()
        }
        button2.setOnClickListener(this)
        // add button dynamically
        val button4 = Button(this)
        button4.setText("Button 4 added dynamically")
        button4.setLayoutParams(ViewGroup.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT))
        button4.setId(button4_Id)
```

```

        button4.x = 250f
        button4.y = 500f
        button4.setOnClickListener(this)
        constraintLayout.addView(button4)
    }
    override fun onClick(view: View) {
        // TODO("not implemented") //To change body of
created functions use File | Settings | File Templates.
        when (view.id) {
            R.id.button2 ->
                Toast.makeText(this,"button 2 clicked", Toast.
LENGTH_SHORT).show()//single line code
            button4_Id->{//multiline code
                val myToast = Toast.makeText(this,"button 4 c
licked", Toast.LENGTH_SHORT)
                myToast.show()
            }
        }
    }
    fun clickButton(v: View){
        val mToast = Toast.makeText(applicationContext,"b
utton 3 clicked", Toast.LENGTH_SHORT)
        mToast.show()
    }
}

```

საბოლოოდ მივიღებთ შედეგს:



ნახ.1.26

1.9. კოტლინის Android სია

Android ListView არის ხედვის კომპონენტი, რომელიც შეიცავს ნივთების ჩამონათვალს და ნაჩვენებია გადასაადგილებელ სიაში. სიის ელემენტები ავტომატურად ემატება სიას, ადაპტერის კლასის გამოყენებით.

➤ Android ListView მაგალითი

შევექმნათ ListView და შევასრულოთ მოქმედებათა სია ერთეულებზე. სიის ერთეულები შეიძლება შეიქმნას კლასის ფაილში ან ცალკეულ ფაილში, როგორიცაა strings.xml.

მაგალითად, სიის ერთეულები კლასის ფაილში დავამატოთ ArrayAdapter კლასში:

```
val language = arrayOf("C","C++","Java",".Net","Kotlin",
    "Ruby","Rails","Python","Java Script","Php","Ajax","Perl","Hadoop")
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val arrayAdapter = ArrayAdapter<String>(this,android.R.layout
    t.simple_list_item_1,language)
    listView.adapter = arrayAdapter
}
```

სიების ერთეულების შექმნა ცალკეულ strings.xml ფაილში და დამატება ArrayAdapter კლასში:

```
<string-array name="technology_list">
```

```
<item>C</item>
<item>C++</item>
<item>Java</item>
<item>.Net</item>
</string-array>
```

```
val language:Array<String> = resources.getStringArray(R.array.technology_list)
val arrayAdapter = ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,language)
listView.adapter = arrayAdapter
```

დაამატეთ ListView კომპონენტი activity_main.xml ფაილში

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="
http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.kotlinlistview.MainActivity">

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
```

```
android:layout_height="fill_parent" />
</android.support.constraint.ConstraintLayout>
```

Strings.xml ფაილში შექმენით სტრიქონების მასივი და დაამატეთ სიის ელემენტები **<item>** ტეგში.

```
<resources>
  <string name="app_name">Kotlin ListView</string>
  <string-array name="technology_list">
    <item>C</item>
    <item>C++</item>
    <item>Java</item>
    <item>.Net</item>
    <item>Kotlin</item>
    <item>Ruby</item>
    <item>Rails</item>
    <item>Python</item>
    <item>Java Script</item>
    <item>Php</item>
    <item>Ajax</item>
    <item>Perl</item>
    <item>Hadoop</item>
  </string-array>
</resources>
```

შემდეგი კოდი დაამატეთ MainActivity.kt კლასის ფაილში.

```
package giorgi.club.app
```

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.widget.*
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    // val language = arrayOf("C","C++" "Java"," Net" "Kotl
in","Ruby","Rails","Python","Java Script","Php","Ajax" "Perl" "H
adoop")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val language:Array<String> = resources.getStringArray(R.arr
ay.technology_list)
        val arrayAdapter = ArrayAdapter<String>(this,android.R.lay
out.simple_list_item_1,language)

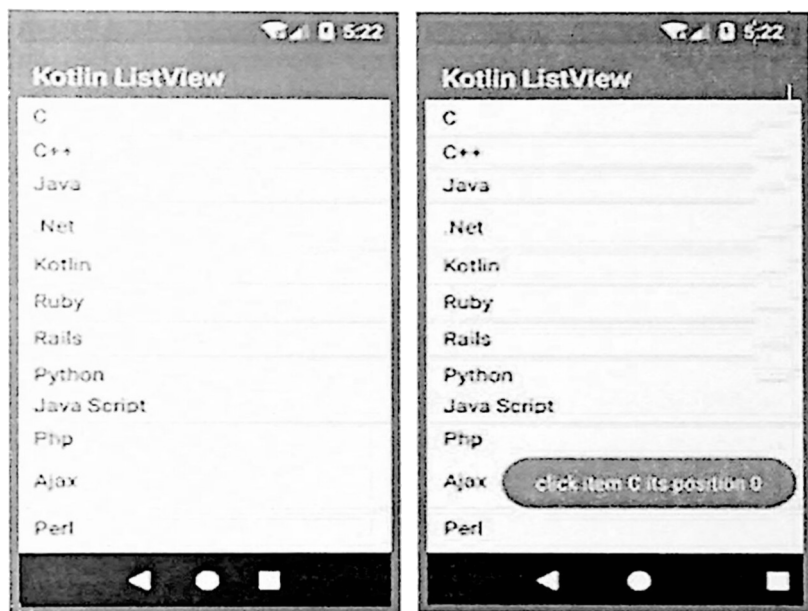
        listView.adapter = arrayAdapter
        listView.setOnItemClickListener = AdapterView.OnItemClickListener { adapterView, view, position, id ->
            val selectedItem = adapterView.getItemAtPosition(position)
as String
            val itemIdAtPos = adapterView.getItemIdAtPosition(position)
```

```

        Toast.makeText(applicationContext,"click item $selectedI
        tem its position $itemIdAtPos",Toast.LENGTH_SHORT).show()
    }
}
}

```

მივიღებთ შედეგს:



სახ.1.27

1.10. Android პარამეტრების მენიუ

Android Options Menu არის მენიუს ერთობლიობა აქტივობისთვის. პარამეტრების მენიუ საშუალებას იძლევა განათავსოთ მოქმედებები, რომლებიც გლობალურ გავლენას ახდენს პროგრამაზე. ამ მაგალითში, ჩვენ დავამატებთ პარამეტრების მენიუს ელემენტებს სამოქმედო ზოლში. მენიუში დაწკაპუნებით ჩანს ოფციის მენიუს ელემენტები, რომლებზეც შეგვიძლია შევასრულოთ შესაბამისი მოქმედება.

შევქმნათ android პროექტი და ავირჩიოთ ძირითადი აქტივობა. ეს აქტივობა ავტომატურად ქმნის კოდებს მენიუს ვარიანტისა და ინსტრუმენტთა პანელისთვის.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout xmlns:andr
oid="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.kotlinoptionsmenu.M
ainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
android:theme="@style/AppTheme.AppBarOverlay">
```

```
<android.support.v7.widget.Toolbar
```

```
    android:id="@+id/toolbar"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="?attr/actionBarSize"
```

```
    android:background="?attr/colorPrimary"
```

```
    app:popupTheme="@style/AppTheme.PopupOverlay" />
```

```
</android.support.design.widget.AppBarLayout>
```

```
    <include layout="@layout/content_main" />
```

```
</android.support.design.widget.CoordinatorLayout>
```

content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<android.support.constraint.ConstraintLayout xmlns:android="
```

```
http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
```

```
        tools:context="example.javatpoint.com.kotlinoptionsmenu.MainActivity"
```

```
        tools:showIn="@layout/activity_main">
```

```
        <TextView
```

```
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
```

```

    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
strings.xml
<resources>
    <string name="app_name">Kotlin OptionsMenu</string>
    <string name="action_settings">Settings</string>
    <string name="action_share">Share</string>
    <string name="action_exit">Exit</string>
</resources>
menu_main.xml
<menu xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="example.javatpoint.com.kotlinoptionsmenu.M
ainActivity">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"/>
    <item
        android:id="@+id/action_share"
        android:title="@string/action_share"
        app:showAsAction="never"/>
    <item

```



```

        android:id="@+id/action_exit"
        android:title="@string/action_exit"
        app:showAsAction="never"/>

```

```

</menu>

```

MainActivity.kt

```

package giorgi.club.app

```

```

import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import android.view.Menu
import android.view.MenuItem
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setSupportActionBar(toolbar)
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {

// Inflate the menu; this adds items to the action bar if it is present.
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean
    {
        return when (item.itemId) {
            R.id.action_settings -> {
                Toast.makeText(applicationContext, "click on setting",
                    Toast.LENGTH_LONG).show()
                true
            }

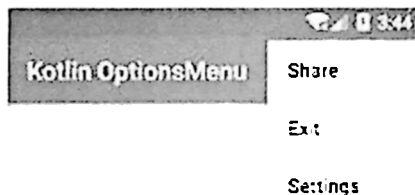
```

```

    }
    R.id.action_share ->{
        Toast.makeText(applicationContext, "click on share", T
oast.LENGTH_LONG).show()
        return true
    }
    R.id.action_exit ->{
        Toast.makeText(applicationContext, "click on exit", Toa
st.LENGTH_LONG).show()
        return true
    }
    else -> super.onOptionsItemSelected(item)
}
}
}

```

შედეგი:



ნახ.1.28

1.11. Android.media.MediaPlayer და მისი მაგალითი

Android.media.MediaPlayer კლასი გამოიყენება აუდიო ან ვიდეო ფაილების გასაკონტროლებლად. მას აქვს წვდომა მედიაპლეიერის ჩამონტაჟებულ სერვისებზე, როგორიცაა აუდიო, ვიდეო და ა.შ. MediaPlayer კლასის გამოსაყენებლად, საჭიროა შეიქმნას მაგალითი create() მეთოდის გამოძახებით. MediaPlayer კლასის რამდენიმე მეთოდი არსებობს. ზოგიერთი მათგანი მოცემულია 1.2 ცხრილში:

MediaPlayer კლასის მეთოდები

ცხრ.1.2

მეთოდი	განმარტება
public void setDataSource(String path)	იგი ადგენს მონაცემთა წყაროს (ფაილის ბილიკს ან http url) გამოსაყენებლად.
public void prepare()	იგი ამზადებს ფლეერს სინქრონულად დაკვრისთვის.
public void start()	იგი იწყებს ან განაგრძობს დაკვრას.
public void stop()	იგი აჩერებს დაკვრას.
public void pause()	იგი აჩერებს დაკვრას.
public boolean isPlaying()	იგი ამოწმებს თუ უკრავს მედია ფლეერი.
public void seekTo(int millis)	იგი ცდილობს განსაზღვროს დრო მილიწამებში.
public void setLooping(boolean looping)	ის ადგენს ფლეიერი ციკლშია თუ არ არის ციკლში.
public boolean isLooping()	
public void selectTrack(int index)	იგი ირჩევს ტრეკს მითითებულ ინდექსისთვის.
public int getCurrentPosition()	ის განსაზღვრავს დაკვრის მიმდინარე პოზიციას.
public int getDuration()	ის უბრუნებს ფაილის ხანგრძლივობას.
public void setVolume(float leftVolume, float rightVolume)	ის ადგენს ამ ფლეიერის ხმას.

➤ Android MediaPlayer-ის მაგალითი SeekBar-ით

ამ მაგალითში ჩვენ შევექმნით Media Player-ს დაკვრის კონტროლის ფუნქციონირებით, როგორცაა ჩართვა, პაუზა და შეჩერება. ჩვენ ვაერთიანებთ SeekBar-ს, მედიაპლეიერის პროგრესის დონის სანახავად.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="
http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.kotlinmediaplayer.M
ainActivity">
```

<Button

```
    android:id="@+id/pauseBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:enabled="false"
    android:text="Pause"
```

```
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toStartOf="@+id/playBtn"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

<Button

```
android:id="@+id/playBtn"
android:layout_width="88dp"
android:layout_height="wrap_content"
android:layout_marginBottom="8dp"
android:layout_marginEnd="8dp"
android:layout_marginStart="8dp"
android:layout_marginTop="8dp"
android:text="Play"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toStartOf="@+id/stopBtn"
app:layout_constraintStart_toEndOf="@+id/pauseBtn"
app:layout_constraintTop_toTopOf="parent" />
```

<Button

```
android:id="@+id/stopBtn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginBottom="8dp"
android:layout_marginEnd="24dp"
android:layout_marginRight="24dp"
android:layout_marginTop="8dp"
android:enabled="false"
android:text="Stop"
```

```
app:layout_constraintBottom_toBottomOf="parent"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintTop_toTopOf="parent" />
```

<RelativeLayout

```
android:layout_width="368dp"  
android:layout_height="wrap_content"  
android:layout_marginEnd="8dp"  
android:layout_marginStart="8dp"  
android:layout_marginTop="76dp"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintHorizontal_bias="1.0"  
app:layout_constraintStart_toStartOf="parent"  
app:layout_constraintTop_toTopOf="parent">
```

<TextView

```
android:id="@+id/tv_pass"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content" />
```

<TextView

```
android:id="@+id/tv_due"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_alignParentEnd="true"  
android:layout_alignParentRight="true" />
```

<SeekBar

```
android:id="@+id/seek_bar"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/tv_pass"
        android:saveEnabled="false" />
    </RelativeLayout>
</android.support.constraint.ConstraintLayout>
```

MainActivity.kt

```
package Giorgi.club.mediaplayer

import android.media.MediaPlayer
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*
import android.os.Handler
import android.widget.SeekBar

class MainActivity : AppCompatActivity() {

    private lateinit var mediaPlayer: MediaPlayer
    private lateinit var runnable: Runnable
    private var handler: Handler = Handler()
    private var pause: Boolean = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Start the media player
```

```

playBtn.setOnClickListener{
    if(pause){
        mediaPlayer.seekTo(mediaPlayer.currentPosition)
        mediaPlayer.start()
        pause = false
        Toast.makeText(this,"media playing",Toast.LENGTH_S
HORT).show()
    }else{

        mediaPlayer = MediaPlayer.create(applicationContext,
R.raw.school_bell)
        mediaPlayer.start()
        Toast.makeText(this,"media playing",Toast.LENGTH_S
HORT).show()

    }
    initializeSeekBar()
    playBtn.isEnabled = false
    pauseBtn.isEnabled = true
    stopBtn.isEnabled = true

    mediaPlayer.setOnCompletionListener {
        playBtn.isEnabled = true
        pauseBtn.isEnabled = false
        stopBtn.isEnabled = false
        Toast.makeText(this,"end",Toast.LENGTH_SHORT).show()

    }
}

```



```
// Pause the media player
pauseBtn.setOnClickListener {
    if(mediaPlayer.isPlaying){
        mediaPlayer.pause()
        pause = true
        playBtn.isEnabled = true
        pauseBtn.isEnabled = false
        stopBtn.isEnabled = true
        Toast.makeText(this,"media pause",Toast.LENGTH_SH
ORT).show()
    }
}

// Stop the media player
stopBtn.setOnClickListener{
    if(mediaPlayer.isPlaying || pause.equals(true)){
        pause = false
        seek_bar.setProgress(0)
        mediaPlayer.stop()
        mediaPlayer.reset()
        mediaPlayer.release()
        handler.removeCallbacks(runnable)

        playBtn.isEnabled = true
        pauseBtn.isEnabled = false
        stopBtn.isEnabled = false
        tv_pass.text = ""
        tv_due.text = ""
        Toast.makeText(this,"media stop",Toast.LENGTH_SHORT).show()
    }
}
```

```

    }
    // Seek bar change listener
    seek_bar.setOnSeekBarChangeListener(object : SeekBar.On
SeekBarChangeListener {
        override fun onProgressChanged(seekBar: SeekBar, i: Int,
b: Boolean) {
            if (b) {
                mediaPlayer.seekTo(i * 1000)
            }
        }

        override fun onStartTrackingTouch(seekBar: SeekBar) {
        }

        override fun onStopTrackingTouch(seekBar: SeekBar) {
        }
    })
}
// Method to initialize seek bar and audio stats
private fun initializeSeekBar() {
    seek_bar.max = mediaPlayer.seconds

    runnable = Runnable {
        seek_bar.progress = mediaPlayer.currentSeconds

        tv_pass.text = "${mediaPlayer.currentSeconds} sec"
        val diff = mediaPlayer.seconds - mediaPlayer.currentSeconds
    }
}

```

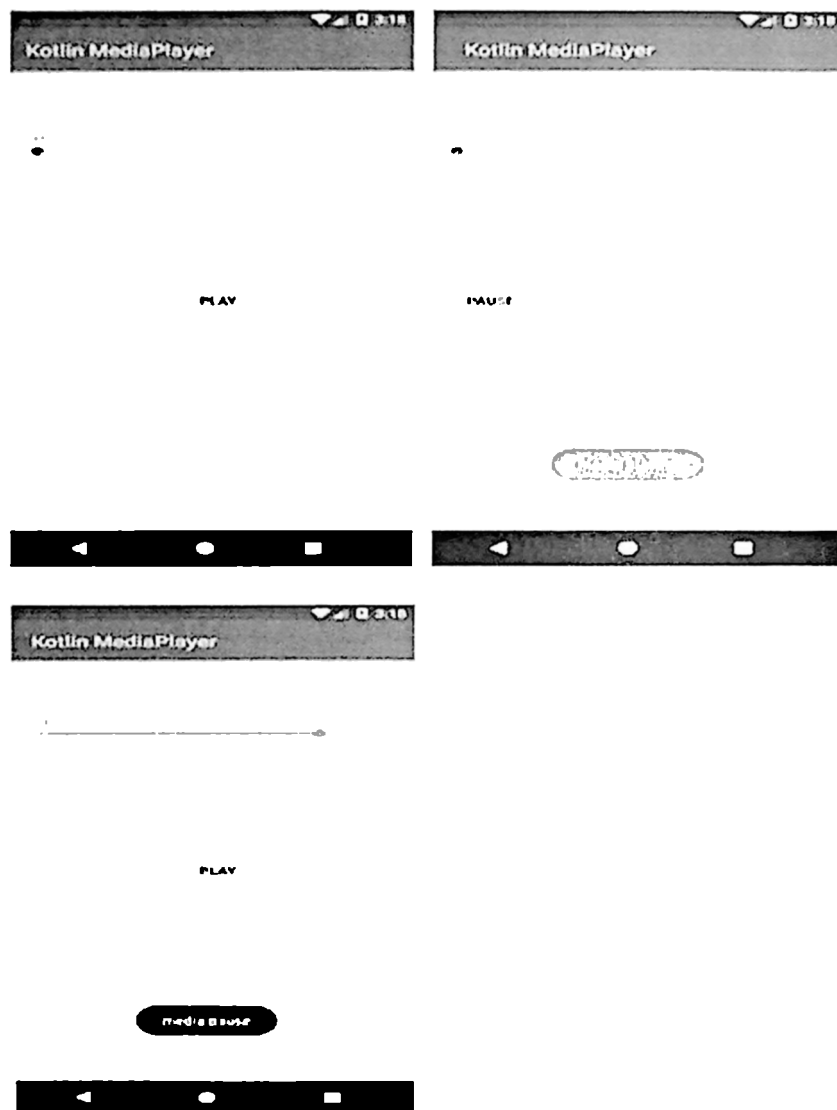
```
tv_due.text = "$diff sec"

    handler.postDelayed(runnable, 1000)
}
handler.postDelayed(runnable, 1000)
}
}

// Creating an extension property to get the media player time d
uration in seconds
val MediaPlayer.seconds: Int
    get() {
        return this.duration / 1000
    }

// Creating an extension property to get media player current p
osition in seconds
val MediaPlayer.currentSeconds: Int
    get() {
        return this.currentPosition / 1000
    }
```

შედეგი მოცემულია 1.29 ნახაზზე.



ნახ.1.29

II თავი. ტელეფონია და SMS

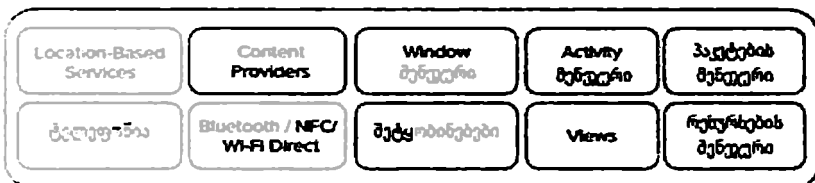
2.1. ანდროიდის პროგრამული უზრუნველყოფა

ანდროიდის პროგრამული უზრუნველყოფა დაფუძნებულია Linux kernel-ზე და C/C++ ბიბლიოთეკების კოლექციებზე, რომლებიც ვლინდება აპლიკაციის framework-ში. იგი უზრუნველფს სერვისებს, მენეჯმენტს, run time-ს და სხვადასხვა აპლიკაციებს (ნახ.2.1) [16].

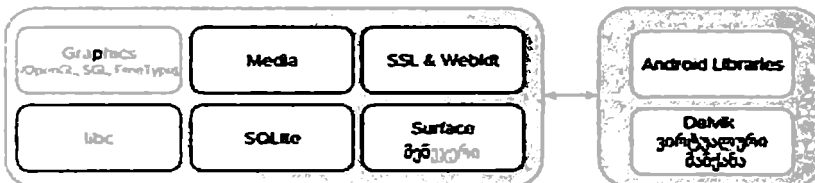
აპლიკაციის შრე



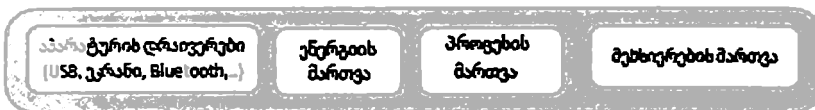
აპლიკაციის ფრეიმვორკი



ბიბლიოთეკები



Linux-ის ბირთვი



ნახ.2.1

➤ Linux kernel – ძირითადი სერვისებია, მათ შორის მოწყობილობის ასამუშავებლად, უზრუნველყოფს დაბალი დონის დრაივერებს, პროცესებს, მეხსიერების მენეჯმენტს, დაცვას, ინტერნეტ კავშირს და სხვ. იმართება Linux 2.6 kernel-ის მეშვეობით. Kernel ასევე წარმოადგენს აბსტრაქტულ დონეს ფიზიკურ მოწყობილობასა და პროგრამულ უზრუნველყოფას შორის.

➤ ბიბლიოთეკები – ანდროიდი შეიცავს C/C++ სხვადასხვა ბიბლიოთეკას, როგორებიცაა:

- Media ბიბლიოთეკა, რომლის საშუალებითაც შესაძლებელია ვიდეო და აუდიო ფაილების გახსნა;
- Surface მენეჯერი გამოსახულების მენეჯმენტისთვის;
- გრაფიკული ბიბლიოთეკა, რომელიც შეიცავს SGL და OpenGL-ს 2D და 3D გამოსახულებისთვის;
- SQLite ლოკალური მონაცემთა ბაზა;
- SSL და WebKit web browser-ის ინტეგრაციისა და ინტერნეტ დაცვისთვის.

➤ Android run time – run time არის ის, რაც ანდროიდის ტელეფონს განასხვავებს უბრალოდ ლინუქს მობილური იმპლემენტაციისგან. ეს არის პრაქტიკა, რომელიც შეიცავს ძირითად ბიბლიოთეკებს და Dalvik VM-ს;

- Core libraries - ანდროიდ აპლიკაციების უმეტესობა დაწერილია java პროგრამირების ენაზე, Dalvik არ არის Java VM. ანდროიდის ძირითადი ბიბლიოთეკები გვამღებს მრავალ ფუნქციონალს, რომელიც ხელმისაწვდომია Java ბიბლიოთეკებში;

Dalvik VM – Dalvik არის ვირტუალური მანქანა, რომელიც ოპტიმიზებულია იმისთვის, რომ მოწყობილობამ იმუშაოს ეფექტურად მრავალ კლიენტზე. იგი დაფუძნებულია Linux kernel-ზე მულტი პროცესინგისთვის და მეხსიერების დაბალი დონით მოხმარებისთვის.

➤ Application framework – აპლიკაციის framework შეიცავს ისეთ კლასებს, რომლებიც საჭიროა აპლიკაციების შესაქმნელად. იგი ასევე აკეთებს აბსტრაქციას ფიზიკური მოწყობილობის გამოსაყენებლად;

➤ Application layer ყველა აპლიკაცია, მესამე დონის და მშობლიური, შექმნილია Application layer-ში. Application layer გაშვებულია და მუშაობს ანდროიდის run time-ში, Application framework-ის კლასების და სერვისების გამოყენებით.

2.2. განსახილველი თემები

განვიხილავთ იმ ბიბლიოთეკებს, რომელთა საშუალებითაც შევძლებთ მონიტორინგი გავუწიოთ მობილურში ხმას, მონაცემებთან წვდომას, შემომავალ და გამავალ ზარებს, ასევე მივიღოთ და გავგზავნოთ მოკლე ტექსტური შეტყობინებები. ამგვარად, შევჩვენებთ შემდეგ საკითხებს:

- შემომავალი სატელეფონო ზარები;
- ტელეფონის, ქსელის, მონაცემებზე წვდომის და სიმ-ბარათის შესახებ მიმდინარე სტატუსის გაგება;
- Intent-ების გამოყენება SMS და MMS გასაგზავნად;
- SMS Manager-ის გამოყენება SMS-ის გასაგზავნად;
- შემომავალი SMS-ების დაჭერა.

ანდროიდ პლატფორმა გვამლევს სრულ წდომას SMS ფუნქციონალთან, გვამლევს საშუალებას გავგზავნოთ და მივიღოთ შეტყობინებები აპლიკაციაში. შესაძლებლობა გვაქვს შევქმნათ ჩვენი საკუთარი SMS კლიენტი აპლიკაცია და ჩავანაცვლოთ მობილურ ტელეფონში შიგნით ჩაშენებული აპლიკაცია. ალტერნატიულად შესაძლებელია ვამუშაოთ ჩვენი აპლიკაცია მთავარ ტელეფონში ჩაშენებულ Native აპლიკაციასთან ერთად [16,17].

2.3. ფიზიკური მოწყობილობის მხარდაჭერა ტელეფონისთვის

მას შემდეგ რაც ბაზარზე გამოჩნდა Wi-Fi-only მოწყობილობები, ჩვენ აღარ უნდა გვქონდეს იმის იმედი რომ ჩვენი აპლიკაცია, რომელიც ტელეფონიას იყენებს, იმუშავებს მათზე.

2.4. ტელეფონის მონიშვნა, როგორც აუცილებელი ფიზიკური თვისება

ზოგიერთი აპლიკაციისათვის არ აქვს მნიშვნელობა მოწყობილობას აქვს თუ არა ტელეფონის მხარდაჭერა. აპლიკაცია, რომელსაც აქვს საშუალება შემომავალი ზარების მონიტორინგის ან თუნდაც SMS კლიენტის ჩანაცვლება, შეიძლება უბრალოდ ვერ იმუშაოს Wi-Fi-only მოწყობილობებზე.

იმისათვის, რომ განვსაზღვროთ სჭირდება თუ არა ჩვენს აპლიკაციას ტელეფონის მხარდაჭერა, საჭიროა აპლიკაციის Manifest ფაილში ჩავწეროთ შემდეგი ტექსტი:


```
<uses-feature android:name="android.hardware.telephony"
            android:required="true"/>
```

2.5. ტელეფონიის ფიზიკური მოწყობილობის შემოწმება

თუ ვიყენებთ ტელეფონიის ბიბლიოთეკებს, მაგრამ ისინი მკაცრად აუცილებელი არაა აპლიკაციის გამოსაყენებლად, მაშინ უბრალოდ შესაძლებელია შევამოწმოთ, გააჩნია თუ არა მოწყობილობას ტელეფონიის ფიზიკური მხარდაჭერა. შესამოწმებლად გამოგვადგება Package Manager კლასის hasSystemService მეთოდი. ამ კლასის მეშვეობით ასევე შესაძლებელია შემოწმდეს CDMA ან GSM არსებობა.

```
PackageManager pm = getPackageManager();
boolean telephonySupported =
    pm.hasSystemFeature(PackageManager.FEATURE_TELEPHONY);
boolean gsmSupported =
    pm.hasSystemFeature(PackageManager.FEATURE_TELEPHONY_
        CDMA);
boolean cdmaSupported =
    pm.hasSystemFeature(PackageManager.FEATURE_TELEPHONY_
        GSM);
```

კარგი პრაქტიკაა თუ ჩვენ წინასწარ შევამოწმებთ ტელეფონიის მხარდაჭერას და ამის მიხედვით იმოქმედებს შემდგომში აპლიკაცია.

2.6. ტელეფონის გამოყენება

ანდროიდის ტელეფონის ბიბლიოთეკა გვამლევს საშუალებას დავწეროთ ჩვენი საკუთარი აპლიკაცია, რომელიც ჩაანაცვლებს დარეკვის Native ფუნქციონალს, ანდა დავიჭერთ და მონიტორინგს გავუწევთ ზარებს ან სტატუსს.

მომდევნო პარაგრაფებში განვიხილავთ, როგორ განვახორციელოთ მონიტორინგი და როგორ გავაკონტროლოთ ტელეფონი, სერვისი და ა.შ.

2.7. ზარის წამოწყება

საუკეთესო პრაქტიკა ზარის წამოსაწყებად არის

`Intent.ACTION_DIAL`

გამოყენება.

მაგალითი 3

```
Intent whoyougonnacall = new Intent(Intent.ACTION_DIAL,  
                                     Uri.parse("tel:555-2368"));  
startActivity(whoyougonnacall);
```

ამ კოდის საშუალებით ტელეფონი გადავა ზარის წამოწყების ნომერზე, რომელიც უკვე გავუწერეთ. დამრეკი ფუნქციონალის მოგვემს საშუალებას შევცვალოთ მითითებული ნომერი. შედეგად მივიღეთ ის, რომ `ACTION_DIAL` `Intent`-ს არ სჭირდება რაიმე განსაკუთრებული ნებართვა.

2.8. ტელეფონის Native Dialer-ის ჩანაცვლება

ტელეფონის Native Dialer-ის ჩანაცვლება ორ ბიჯად იყოფა:

- გადავკვეთა ის Intent-ები რომლებსაც იყენებს Native Dialer;
- გამავალი ზარების ინიციალიზება და შემდგომი მენეჯმენტი.

Native Dialer აპლიკაცია რეაგირებს მაშინ, როდესაც მომხმარებელი დააჭერს „დარეკვა“ ღილაკს, რომელიც გადასცემს ბრძანებას შემდეგნაირად tel: schema, ან მაშინ, როდესაც მოხდება ბრძანება ACTION_DEAL tel: schema-ს გამოყენებით, როგორც წინა მაგალითში არის ნაჩვენები.

იმისათვის რომ გადავკვეთოთ ეს ბრძანებები საჭიროა დავამატოთ intent-filter ტაგები manifest ფაილში რათა მოვუსმინოთ შემდგომში ქმედებებს:

➤ Intent.ACTION_CALL_BUTTON – მოხდება მაშინ როდესაც მომხმარებელი დააჭერს ეკრანზე არსებულ „დარეკვა“ ღილაკს. უნდა გავაკეთოთ intent-filter, რომელიც მოუსმენს ამ ბრძანებას როგორც საკუთარს;

➤ Intent.ACTION_DIAL – როგორც უკვე ვახსენეთ, ეს გამოიყენება აპლიკაციების საშუალებით, რომელთაც სურს ზარის წამოწყება. საჭიროა დავამატოთ intent-filter, რომელიც ასევე მოუსმენს ამ ბრძანებას;

➤ Intent.ACTION_VIEW – გამოიყენება აპლიკაციების მიერ, რომელთაც სურს ინფორმაციის ნახვა. საჭიროა intent-

filter გნსაზღვრავდეს tel: schema-ს იმისათვის, რომ შევძლოთ ტელეფონის ნომრების ნახვა.

ქვემოთ ნაჩვენებია manifest ფაილი, სადაც intent-filter-ები არის შესაბამისად გაწერილი:

მაგალითი 4

```
<activity
    android:name=".MyDialerActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action
            android:name="android.intent.action.CALL_BUTTON" />
        <category
            android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <action android:name="android.intent.action.DIAL" />
        <category
            android:name="android.intent.category.DEFAULT" />
        <category
            android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="tel" />
    </intent-filter>
</activity>
```

მას შემდეგ რაც აპლაცია ჩაირთვება, მომხმარებლის ინტერფეისი მზად უნდა იყოს ნომრების შესაყვანად, დასარეკად და ა.შ. ზარის განსახორციელებლად საჭიროა შემდეგი ბრძანება:

მაგალითი 5

```
Intent whoyougonnacall = new Intent(Intent
    .ACTION_CALL, Uri.parse("tel:555-2368"));
startActivity(whoyougonnacall);
```

იმისათვის, რომ განვახორციელოთ დარეკვა, საჭიროა manifest ფაილში შემდეგი ნებართვის გაწერა:

```
<uses-permission android:name =
    "android.permission.CALL_PHONE"/>
```

ბრძანების შესრულების ალტერნატიული გზა არის საკუთარი დარეკვისა და ხმის დამუშავების ბიბლიოთეკის დამუშავება. საუკეთესო ვარიანტია VOIP აპლიკაციის იმპლემენტაცია.

2.9. ტელეფონიის პარამეტრებზე და მდგომარეობზე წვდომა

ტელეფონიის პარამეტრებზე წვდომა შესაძლებელია Telephony Manager კლასის getSystemService მეთოდის საშუალებით:

```
String srcvName = Context.TELEPHONY_SERVICE;
TelephonyManager telephonyManager =
    (TelephonyManager) getSystemService(srcvName);
```

Telephony Manager კლასს აქვს პირდაპირი წვდომა ტელეფონის ბევრ პარამეტრზე, მათ შორის მოწყობილობაზე, ქსელზე, სიმზარათზე და სხვ. აგრეთვე შესაძლებელია ქსელთან კავშირის შესახებ ინფორმაციაზე წვდომაც.

2.10. მოწყობილობის პარამეტრებზე წვდომა

Telephony Manager-ის გამოყენებით შეგვიძლია გავიგოთ ტელეფონის ტიპი (GSM, CDMA ან SIP), უნიკალური ID (IMEI ან MAID), პროგრამული უზრუნველყოფის ვერსია და ტელეფონში ჩადებული ნომერი (მაგალითი_7).

პარამეტრების წასაკითხად საჭიროა სპეციალური ნებართვა manifest ფაილში:

```
<uses-permission android:name =
    "android.permission.READ_PHONE_STATE"/>
```

```
String phoneTypeStr = "unknown";
int phoneType = telephonyManager.getPhoneType();
switch (phoneType) {
    case (TelephonyManager.PHONE_TYPE_CDMA):
        phoneTypeStr = "CDMA";
        break;
    case (TelephonyManager.PHONE_TYPE_GSM) :
        phoneTypeStr = "GSM";
        break;
    case (TelephonyManager.PHONE_TYPE_SIP):
        phoneTypeStr = "SIP";
        break;
    case (TelephonyManager.PHONE_TYPE_NONE):
        phoneTypeStr = "None";
        break;
    default: break;
}
String deviceId = telephonyManager.getDeviceId();
String softwareVersion =
telephonyManager.getDeviceSoftwareVersion();
String phoneNumber = telephonyManager.getLine1Number();
```

2.11. სიმ-ზარათის პარამეტრების წაკითხვა

თუ აპლიკაცია გაშვებულია GSM მოწყობილობაზე, მაშინ მასში ჩადებული იქნება სიმ-ზარათი, რომლის პარამეტრებზე წვდომა შეგვიძლია Telephony Manager კლასის საშუალებით. ინფორმაციის ცოდნა სიმის შესახებ კარგია

მაშინ, როდესაც გვჭირდება განსაკუთრებული ფუნქციონალი რომელიმე ქვეყანაში.

თუ manifest ფაილში გავწერთ READ_PHONE_STATE ნებართვას, მაშინ შევძლებთ გავიგოთ სიმ სერიული ნომერი შემდეგნაირად:

მაგალითი 8

```
int simState = telephonyManager.getSimState();
switch (simState)
{
    case (TelephonyManager.SIM_STATE_ABSENT):
        break;
    case (TelephonyManager.SIM_STATE_NETWORK_LOCKED):
        break;
    case (TelephonyManager.SIM_STATE_PIN_REQUIRED):
        break;
    case (TelephonyManager.SIM_STATE_PUK_REQUIRED):
        break;
    case (TelephonyManager.SIM_STATE_UNKNOWN):
        break;
    case (TelephonyManager.SIM_STATE_READY): {
        String simCountry = telephonyManager.getSimCountryIso();
        String simOperatorCode = telephonyManager.getSimOperator();
        String simOperatorName =
            telephonyManager.getSimOperatorName();
        String simSerial = telephonyManager.getSimSerialNumber();
        break;
    }
    default: break;
```


2.12. მონაცემთა კავშირისა და მიმოცვლის სტატუსის დეტალების წაკითხვა

getDataState და getDataActivity მეთოდების გამოყენებით შესაძლებელია გავიგოთ მიმდინარე მონაცემთა კავშირის და მონაცემთა მიმოცვლის სტატუსი.

მაგალითი 9

```
int dataActivity = telephonyManager.getDataActivity();
int dataState = telephonyManager.getDataState();
switch (dataActivity) {
    case TelephonyManager.DATA_ACTIVITY_IN : break;
    case TelephonyManager.DATA_ACTIVITY_OUT : break;
    case TelephonyManager.DATA_ACTIVITY_INOUT : break;
    case TelephonyManager.DATA_ACTIVITY_NONE : break;
}
switch (dataState) {
    case TelephonyManager.DATA_CONNECTED : break;
    case TelephonyManager.DATA_CONNECTING : break;
    case TelephonyManager.DATA_DISCONNECTED : break;
    case TelephonyManager.DATA_SUSPENDED : break;}
```

2.13. ტელეფონის სტატუსის ცვლილების მონიტორინგი

ანდროიდის ტელეფონის ბიბლიოთეკა საშუალებას გვაძლევს მონიტორინგი გავუწიოთ ტელეფონის მდგომარეობას და მასთან დაკავშირებულ დეტალებს, როგორიცაა შემომავალი ზარის ნომერი.

ტელეფონის სტატუსის ცვლილების მონიტორინგი PhoneStateListener კლასის გამოყენებითაა შესაძლებელი.

აღნიშნული ფუნქციონალის გამოსაყენებლად საჭიროა ნებართვა manifest ფაილში:

```
<uses-permission android:name =  
    "android.permission.READ_PHONE_STATE"/>
```

საჭიროა შევქმნათ კლასი, რომელიც იმპლემენტაციას გაუკეთებს PhoneStateListener კლასს და რეაგირებას მოახდენს ტელეფონის სტატუსის, ზარის სტატუსის, მობილური სიგნალის სიძლიერის ცვლილებაზე და ა.შ.

იმპლემენტაცია უნდა გავუკეთოთ ყველა საჭირო event-ს, რომელსაც უნდა მოვუსმინოთ. თითოეული event handler მიიღებს შესაბამის პარამეტრებს, რომელიც მიუთითებს ტელეფონის მდგომარეობაზე.

მაგალითი 10

```
telephonyManager.listen(phoneStateListener,  
    PhoneStateListener.LISTEN_CALL_FORWARDING_INDICATOR  
    |  
    PhoneStateListener.LISTEN_CALL_STATE |  
    PhoneStateListener.LISTEN_CELL_LOCATION |  
    PhoneStateListener.LISTEN_DATA_ACTIVITY |  
    PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |  
    PhoneStateListener.LISTEN_MESSAGE_WAITING_INDICATOR |  
    PhoneStateListener.LISTEN_SERVICE_STATE |  
    PhoneStateListener.LISTEN_SIGNAL_STRENGTH);
```

2.14. შემომავალი ზარების მონიტორინგი

თუ აპლიკაციამ უნდა მოახდინოს რეაგირება შემომავალ ზარებზე მხოლოდ მაშინ, როდესაც ჩართულია, მაშინ შეიძლება აღიწეროს `onCallStateChanged` მეთოდი `Phone State Listener` კლასის იმპლემენტაციაში და დარეგისტრირდეს იმისათვის, რომ მიღებულ იქნას შეტყობინება, როდესაც ზარის სტატუსი შეიცვლება.

მაგალითი 11

```
PhoneStateListener callStateListener = new PhoneStateListener()
{
    public void onCallStateChanged(int state, String incomingNumber)
    {
        String callStateStr = "Unknown";
        switch (state) {
            case TelephonyManager.CALL_STATE_IDLE :
                callStateStr = "idle"; break;
            case TelephonyManager.CALL_STATE_OFFHOOK :
                callStateStr = "offhook"; break;
            case TelephonyManager.CALL_STATE_RINGING :
                callStateStr = "ringing. Incoming number is: "
                    + incomingNumber;
                break;
            default : break;
        }
        Toast.makeText(MyActivity.this, callStateStr,
            Toast.LENGTH_LONG).show();
    }
    telephonyManager.listen(callStateListener,
        PhoneStateListener.LISTEN_CALL_STATE);
}
```

onCallStateChanged მეთოდი პარამეტრად მიიღებს შემომავალი ზარის შესახებ ტელეფონის ნომერს და მდგომარეობას, რომელიც მიმდინარე ზარის ერთ-ერთი მდგომარეობაა შემდეგი სამი პარამეტრიდან:

➤ TelephonyManager.CALL_STATE_IDLE – თუ ტელეფონი არც რეკავს და არც შემომავალი ზარი აქვს;

➤ TelephonyManager.CALL_STATE_RINGING – როდესაც ტელეფონი რეკავს;

➤ TelephonyManager.OFFHOOK – როდესაც ტელეფონი არის უკვე საუბრის რეჟიმში.

როგორც კი ტელეფონი გადადის CALL_STATE_RINGING სტატუსში, სისტემა აჩვენებს მომხმარებელს შესაბამისი ზარის ეკრანს და ეკითხება მას - თუ სურს პასუხი.

აპლიკაცია უნდა იყოს ჩართული თუ გვსურს, რომ ზარები მივიღოთ. თუ აპლიკაცია უნდა იყოს ჩართული მაშინ, როდესაც ტელეფონის სტატუსი იცვლება, მაშინ უნდა დავარეგისტრიროთ Intent Receiver, რომელიც მოუსმენს ტელეფონის სტატუსის ცვლილებებს.

2.15. მობილურის ადგილმდებარეობის

ცვლილების ტრეკინგი

შესაძლებელია შეტყობინების მიღება მაშინ, როდესაც ტელეფონი შეიცვლის ადგილმდებარეობას, ამის მისაღებად საჭიროა აღწეროთ onCellLocationChanged მეთოდი Phone State Listener კლასში, მაგრამ ამ ინფორმაციის მისაღებად საჭიროა სპეციალური ნებართვა:

```
<uses-permission android:name =  
    "android.permission.ACCESS_COARSE_LOCATION"/>
```

onCellLocationChanged მეთოდი პარამეტრად იღებს CellLocation ობიექტს, რომელსაც აქვს მეთოდები, სხვადასხვა ადგილმდებარეობის შესახებ ინფორმაციის მისაღებად, რომლებიც დამოკიდებულია ტელეფონის ქსელის ტიპზე.

მაგალითად GSM ქსელისთვის Cell Id მისაღებად getCid და მიმდინარე ადგილისთვის getLac მეთოდი. CDMA ქსელისთვის getBaseStationId, getBaseStationLatitude და getBaseStationLongitude.

მე-12 მაგალითში ნაჩვენებია კოდის ფრაგმენტი, რომელიც აღწერს Phone State Listener კლასს მობილურის ადგილმდებარეობის ცვლილებას.

2.16. სერვისის ცვლილებების ტრეკინგი

onServiceStateChanged მოწყობილობის სერვისის დეტალების ტრეკინგს აკეთებს. სერვისის მდგომარეობის მისაღებად უნდა გამოვიყენოთ ServiceState პარამეტრი.

ServiceState ობიექტის getState მეთოდი აბრუნებს სერვისის მიმდინარე მდგომარეობას შემდეგი ჩამონათვალიდან რომელიმეს:

➤ STATE_IN_SERVICE – ტელეფონის ნორმალური მდგომარეობა;

➤ STATE_EMERGENCY_ONLY – ტელეფონის სერვისი ნორმალურია, მაგრამ შესაძლებელია მხოლოდ გადაუდებელი დახმარების ნომერზე დარეკვა;

```
PhoneStateListener cellLocationListener = new PhoneStateListener()
{
    public void onCellLocationChanged(CellLocation location)
    {
        if (location instanceof GsmCellLocation)
        {
            GsmCellLocation gsmLocation = (GsmCellLocation)location;
            Toast.makeText(getApplicationContext(),
                String.valueOf(gsmLocation.getCid()),
                Toast.LENGTH_LONG).show();
        }
        else if (location instanceof CdmaCellLocation)
        {
            CdmaCellLocation cdmaLocation = (CdmaCellLocation)location;
            StringBuilder sb = new StringBuilder();
            sb.append(cdmaLocation.getBaseStationId());
            sb.append("\n@");
            sb.append(cdmaLocation.getBaseStationLatitude());
            sb.append(cdmaLocation.getBaseStationLongitude());
            Toast.makeText(getApplicationContext(),
                sb.toString(),
                Toast.LENGTH_LONG).show();
        }
    }
};
telephonyManager.listen(cellLocationListener,
    PhoneStateListener.LISTEN_CELL_LOCATION);
```

➤ STATE_OUT_OF_SERVICE – სატელეფონო სერვისები მიუწვდომელია;

➤ STATE_POWER_OFF – სატელეფონო სერვისი გამორთულია (ჩვეულებრივ მაშინ როდესაც Airplane Mode ჩართულია)

მაგალითი 13

```
PhoneStateListener serviceStateListener = new
PhoneStateListener()
{
    public void onServiceStateChanged(ServiceState
serviceState)
    {
        if (serviceState.getState() ==
ServiceState.STATE_IN_SERVICE)
        {
            String toastText = "Operator: " +
serviceState.getOperatorAlphaLong();
            Toast.makeText(MyActivity.this, toastText,
Toast.LENGTH_SHORT);
        }
    }
};
telephonyManager.listen(serviceStateListener,
PhoneStateListener.LISTEN_SERVICE_STATE);
```

2.17. მონაცემთა კავშირის და მონაცემთა გაცვლის სტატუსის ცვლილების მონიტორინგი

მობილური ინტერნეტის კავშირის და მონაცემთა გაცვლის მონიტორინგისათვის შეგვიძლია გამოვიყენოთ Phone State Listener კლასი. ეს კლასი არ შეიცავს კონტროლის მექანიზმებს Wi-Fi შემთხვევაში, ამისათვის უმჯობესია გამოვიყენოთ Connectivity Manager კლასი.

Phone State Listener კლასი შეიცავს ორ კალსს მონიტორინგისთვის. საჭიროა აღვწეროთ `onDataActivity` მონაცემთა გაცვლის საკონტროლებლად, ხოლო მობილური ინტერნეტის სტატუსის ცვლილების შესახებ ინფორმაციის მისაღებად უნდა აღვწეროთ `onDataConnectionStateChanged` მეთოდი (მაგალითი_14).

2.18. Intent Receiver-ების გამოყენება შემომავალი ზარების საკონტროლებლად

როდესაც იცვლება ტელეფონის მდგომარეობა ზარის შემოსვლისას, ნაპასუხები ან გაუქმებული სატელეფონო ზარისას, Telephony Manager კლასი გააკეთებს ACTION_PHONE_STATE_CHANGED Intent-ის Broadcast-ს.

იმ შემთხვევაში თუ manifest ფაილში დავარეგისტრირებთ Intent Receiver-ს, რომელიც მიიღებს:

ACTION_PHONE_STATE_CHANGED

Intent-ს, ნებისმიერ დროს შესაძლებელია შემომავალი ზარის მონიტორინგი.


```
PhoneStateListener dataStateListener = new PhoneStateListener() {
    public void onDataActivity(int direction) {
        String dataActivityStr = "None";
        switch (direction)
        { case TelephonyManager.DATA_ACTIVITY_IN
          dataActivityStr = "Downloading"; break;
          case TelephonyManager.DATA_ACTIVITY_OUT :
          dataActivityStr = "Uploading"; break;
          case TelephonyManager.DATA_ACTIVITY_INOUT :
          dataActivityStr = "Uploading/Downloading"; break;
          case TelephonyManager.DATA_ACTIVITY_NONE :
          dataActivityStr = "No Activity"; break; }
        Toast.makeText(MyActivity.this,
            "Data Activity is " + dataActivityStr,
            Toast.LENGTH_LONG).show();    }
    public void onDataConnectionStateChange(int state) {
        String dataStateStr = "Unknown";
        switch (state)
        { case TelephonyManager.DATA_CONNECTED :
          dataStateStr = "Connected"; break;
          case TelephonyManager.DATA_CONNECTING :
          dataStateStr = "Connecting"; break;
          case TelephonyManager.DATA_DISCONNECTED :
          dataStateStr = "Disconnected"; break;
          case TelephonyManager.DATA_SUSPENDED :
          dataStateStr = "Suspended"; break; }
        Toast.makeText(MyActivity.this,
            "Data Connectivity is " + dataStateStr,
            Toast.LENGTH_LONG).show();
    }
};
```

იმ შემთხვევაშიც კი, თუ თქვენი აპლიკაცია გათიშულია. ასევე საჭიროა READ_PHONE_STATE ნებართვის მიღება, საჭიროა იგი გავწეროთ manifest ფაილში:

მაგალითი 15

```
<receiver android:name="PhoneStateChangedReceiver">
    <intent-filter>
        <action
            android:name="android.intent.action.PHONE_STATE"></a
            ction>
        </intent-filter>
    </receiver>
```

2.19. SMS და MMS

თუ ტელეფონი არაა 2 ათწლეულზე უფრო ძველი, დიდი ალბათობაა, რომ თქვენთვის ნაცნობი იყოს SMS მიმოწერა. დღესდღეობით SMS არის ერთ-ერთი ყველაზე მეტად გამოყენებადი მექანიზმი ყველა მობილურ ტელეფონში.

SMS ტექნოლოგია არის შექმნილი იმისთვის, რომ გავგზავნოთ მოკლე ტექსტური შეტყობინება მობილურ ტელეფონებს შორის. მისი საშუალებით შეგვიძლია გავგზავნოთ როგორც ტექსტური მესიჯები, ასევე ინფორმაციული მესიჯები სხვა აპლიკაციებისთვის. მულტიმედია მესიჯ სერვისები (MMS) მესიჯები მომხმარებელს აძლევს საშუალებას, რომ გაგზავნონ და მიიღონ სურათები, ვიდეოები და აუდიო ფაილები.

2.20. SMS და MMS გამოყენება თქვენს აპლიკაციაში

ანდროიდი მხარს უჭერს აპლიკაციებს, რომლებიც გზავნი SMS და MMS შეტყობინებებს SEND და SEND_TO Broadcast Intent-ის საშუალებით.

ანდროიდი ასევე გვადლევს საშუალებას გამოვიყენოთ სრული ფუნქციონალური SmsManager კლასის საშუალებით. შესაძლებელია ჩავანაცვლოთ Native SMS აპლიკაცია ჩვენი აპლიკაციით და მივიღოთ, გავგზავნოთ SMS ჩვენთვის.

ჯერჯერობით ანდროიდი არ გვადლევს საშუალებას შევქმნათ MMS აპლიკაციის საშუალებით. SMS შეტყობინება არის ნელი, შესაძლოა ფასიანიც კი. ამიტომ SMS არაა შესაფერისი ინფორმაციის სწრაფი მიმოცვლასათვის.

2.21. SMS და MMS გაგზავნა აპლიკაციიდან Intent-ის საშუალებით

უმეტეს შემთხვევებში უმჯობესია SMS და MMS შეტყობინებები გავგზავნოთ Intent-ის საშუალებით ვიდრე გამოვიყენოთ სრული SMS კლიენტი.

ქვემოთ ნაჩვენებია მაგალითი თუ როგორ უნდა გავაგზავნოთ SMS შეტყობინება Intent-ის საშუალებით:

მაგალითი 16

```
Intent smsIntent = new Intent(Intent.ACTION_SENDTO,  
                             Uri.parse("sms:55512345"));  
smsIntent.putExtra("sms_body", "Press send to send me");  
startActivity(smsIntent);
```

SMS-ზე ფაილის მისამაგრებლად კოდს უნდა დაეუმატოთ Intent.EXTRA_STREAM იმ ფაილის მისამართით რომლის მიმაგრებაც გვსურს.

მაგალითი 17

```
// Get the URI of a piece of media to attach.
Uri attached Uri
= Uri.parse("content://media/external/images/media/1");
// Create a new MMS intent
Intent mmsIntent = new Intent(Intent.ACTION_SEND, attached Uri);
mmsIntent.putExtra("sms_body", "Please see the attached image");
mmsIntent.putExtra("address", "07912355432");
mmsIntent.putExtra(Intent.EXTRA_STREAM, attached Uri);
mmsIntent.setType("image/jpeg");
startActivity(mmsIntent);
```

2.22. SMS შეტყობინების გაგზავნა SMS Manager-ის საშუალებით

SMS შეტყობინებების მიმოცვლას უზრუნველყოფს SmsManager კლასი, რომლის ობიექტის მიღება ასეა შესაძლებელი:

```
SmsManager smsManager = SmsManager.getDefault();
```

ამ შემთხვევაში დაგვჭირდება სპეციალური ნებართვა:

```
<uses-permission android:name=
    "android.permission.SEND_SMS"/>
```

2.22.1. ტექსტური შეტყობინების გაგზავნა

შეტყობინების გასაგზავნად საჭიროა გამოვიყენოთ `sendTextMessage` მეთოდი `SmsManager` კლასიდან, რომელსაც პარამეტრად უნდა გადავცეთ ტელეფონის ნომერი, სადაც ვგზავნით, და ტექსტი:

მაგალითი 18

```
SmsManager smsManager = SmsManager.getDefault();
String sendTo = "5551234";
String myMessage = "Android supports programmatic SMS messaging!";
smsManager.sendTextMessage(sendTo, null, myMessage, null, null);
```

მეორე პარამეტრი შეიძლება იყოს სერვის ცენტრის ნომერი, თუ გადავცემთ `null`-ს, მაშინ იგი ავტომატურად აიღებს მას ქსელის მიხედვით, რომელზეც არის დაკავშირებული.

2.22.2. SMS შეტყობინების ტრეკინგი

SMS tracker პროგრამაა, რომელიც ხსნის დეტალურ ინფორმაციას ტექსტური და სურათის შეტყობინების მიწოდებისა და შინაარსის შესახებ. იგი იძლევა შეტყობინების ანალიზის საშუალებას, რომ აისახოს მიწოდების ნიმუშები, კოდირების დეტალები და შეცდომის პირობები.

იმისათვის, რომ განხორციელდეს ტრეკინგი, გაგზავნილი შეტყობინება მივიდა თუ არა დანიშნულ ადგილზე, საჭიროა აღიწეროს და დარეგისტრირდეს `Broadcast Receiver` (სამაუწყებლო მიმღები), რომელიც მოუსმენს გაგზავნილი შეტყობინების `Intent`-ებს.

პირველი პარამეტრი გვაცნობებს, გაიგზავნა თუ არა ტექსტური შეტყობინება. მისი შედეგი იქნება შემდეგი სტატუსებიდან ერთ-ერთი:

- Activity.RESULT_OK – გაიგზავნა წარმატებით;
- SmsManager.RESULT_ERROR_GENERIC_FAILURE – შეტყობინება არ გაგზავნილა უცნობი მიზეზის გამო;
- SmsManager.RESULT_ERROR_RADIO_OFF – შეტყობინება არ გაგზავნილა სიხშირის გათიშვის გამო (Airplane mode);
- SmsManager.RESULT_ERROR_NULL_PDU – შეტყობინება არ გაგზავნილა PDU-ს გამო;
- SmsManager.RESULT_ERROR_NO_SERVICE – ფიჭური კავშირი მიუწვდომელია.

მეორე პარამეტრი პასუხისმგებელია იმ ინფორმაციაზე, მივიდა თუ არა შეტყობინება დანიშნულების ადგილზე.

ქვემოთ მოყვანილი კოდის ლისტინგი არის ტიპური პატერნი შეტყობინების სტატუსების კონტროლისათვის:

//--ლისტინგი -----

```
String SENT_SMS_ACTION =
    "com.paad.smssnippets.SENT_SMS_ACTION";
String DELIVERED_SMS_ACTION =
    "com.paad.smssnippets.DELIVERED_SMS_ACTION";
// -- sentIntent SentIntent პარამეტრის შექმნა ---
Intent sentIntent = new Intent(SENT_SMS_ACTION);
PendingIntent sentPI =
    PendingIntent.getBroadcast(getApplicationContext(), 0,
        sentIntent, PendingIntent.FLAG_UPDATE_CURRENT);
// -- deliveryIntent პარამეტრის შექმნა ---
Intent deliveryIntent = new Intent(DELIVERED_SMS_ACTION);
PendingIntent deliverPI =
```

```

PendingIntent.getBroadcast(getApplicationContext(), 0,
    deliveryIntent, PendingIntent.FLAG_UPDATE_CURRENT);
//-- Broadcast Receivers (მაუწყებლის მიმღებების) რეგისტრირება--
registerReceiver(new BroadcastReceiver()
{ @Override
    public void onReceive(Context _context, Intent _intent)
    { String resultText = "UNKNOWN";
      switch (getResultCode())
      { case Activity.RESULT_OK:
          resultText = "Transmission successful"; break;
        case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
          resultText = "Transmission failed"; break;
        case SmsManager.RESULT_ERROR_RADIO_OFF:
          resultText = "Transmission failed: Radio is off";
          break;
        case SmsManager.RESULT_ERROR_NULL_PDU:
          resultText = "Transmission Failed: No PDU specified";
          break;
        case SmsManager.RESULT_ERROR_NO_SERVICE:
          resultText = "Transmission Failed: No service";
          break;
      }
      Toast.makeText(_context, resultText,
        Toast.LENGTH_LONG).show();
    }
}
new IntentFilter(SENT_SMS_ACTION));
registerReceiver(new BroadcastReceiver()
{ @Override
    public void onReceive(Context _context, Intent _intent)
    {
      Toast.makeText(_context, "SMS Delivered",
        Toast.LENGTH_LONG).show();
    }
}

```

```

},
new IntentFilter(DELIVERED_SMS_ACTION));
// -- შეტყობინების გაგზავნა ---
SmsManager smsManager = SmsManager.getDefault();
String sendTo = "5551234";
String myMessage = "Android supports programmatic SMS messaging!";
smsManager.sendTextMessage(sendTo, null, myMessage, sentPI,
deliverPI);

```

2.22.3. SMS შეტყობინების მაქსიმალური ზომა

ტექსტური შეტყობინების მაქსიმალური სიგრძე შეიძლება განსხვავდებოდეს ქსელის ოპერატორის მიხედვით, მაგრამ, როგორც წესი, შეზღუდვა არის 160 სიმბოლოზე. შედეგად უფრო გრძელი შეტყობინებები უნდა დაიხლიჩოს პატარა შეტყობინებებად. SmsManager კლასს გააჩნია მეთოდი `divideMessage`, რომელიც პარამეტრად იღებს ტექსტს და შედეგად მას დაყოფს პატარა ტექსტების მასივად, რომელთაგან თითოეული არ აღემატება მაქსიმალურ ზომას.

ამის შემდეგ შეგვიძლია გამოვიძახოთ:

`sendMultipartTextMessage`

მეთოდი, რათა დაყოფილი ტექსტი გავგზავნოთ:

მაგალითი 19

```

ArrayList<String> messageArray = smsManager.divideMessage(myMessage);
ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>();
for (int i = 0; i < messageArray.size(); i++)
    sentIntents.add(sentPI);
smsManager.sendMultipartTextMessage(sendTo, null, messageArray,
sentIntents, null);

```


2.22.4. ინფორმაციული შეტყობინებების გაგზავნა

შესაძლებელია ბინარული ინფორმაციის SMS-ით გაგზავნა `sendDataMessage` მეთოდით `SmsManager` კლასიდან. `sendDataMessage` მუშაობს დაახლებით ისევე, როგორც `sendTextMessage`, ერთი განსხვავებით, რომ უნდა მივუთითოთ მიმართულების პორტი სადაც ვაპირებთ გაგზავნას და გასაგზავნი ინფორმაციის შესაბამისი ბინარული ინფორმაცია.

მაგალითი 20

```
String sendTo = "5551234";  
short destinationPort = 80;  
byte[] data = [ ... your data ... ];  
smsManager.sendDataMessage(sendTo, null, destinationPort,  
data, null, null);
```

2.25. შემომავალი SMS შეტყობინებების მოსმენა

როდესაც მოწყობილობა მიიღებს ახალ SMS შეტყობინებას, ახალი `Broadcast Intent` აღიბერება

`android.provider.Telephony.SMS_RECEIVED` action-თან ერთად.

აღსანიშნავია ის, რომ შემომავალი პარამეტრი არის პირდაპირ ტექსტური და იგი შეგვიძლია უშუალოდ გამოვიყენოთ.

იმისათვის, რომ გამოვიყენოთ აპლიკაციაში ზემოაღნიშნული ფუნქციონალი, საჭიროა გვქონდეს სპეციალური ნებართვა RECEIVE_SMS გაწერილი manifest ფაილში:

```
<uses-permission android:name =  
    "android.permission.RECEIVE_SMS"/>
```

SMS Broadcast Intent შეიცავს შემომავალი sms-ის პარამეტრებს. ინფორმაციის ამოსაღებად მოსული შეტყობინებიდან უნდა გამოვიყენოთ pdu გასაღები.

მაგალითი 21

```
Bundle bundle = intent.getExtras();  
if (bundle != null)  
{  
    Object[] pdus = (Object[]) bundle.get("pdus");  
    SmsMessage[] messages = new SmsMessage[pdus.length];  
    for (int i = 0; i < pdus.length; i++)  
        messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]); }
```

2.26. მეორე თავის დასკვნა

ანდროიდის SDK იძლევა მრავალ საჭირო საშუალებას იმისათვის, რომ დამუშავდეს სატელეფონო ზარები, SMS და MMS (მივიღოთ და შევქმნათ) შესაბამის უფლებებთან ერთად. აგრეთვე იძლევა საშუალებას რათა გაუკეთდეს ფორმატირება ტელეფონის ნომრებს, რომლებიც შეჰყავს მომხმარებელს ან მოიძოვება სხვა გზებით.

დეველოპერებს შეუძლიათ ჩააშენონ ხმის დამუშავება, SMS და MMS საკუთარ აპლიკაციებში, ასევე დაამატონ საკუთარი ახალი ფუნქციონალი. SMS ჩატი არის ერთ-ერთი ყველაზე პოპულარული, ასე რომ მისი ინტეგრაცია აპლიკაციაში შექმნის კარგ ფუნქციონალს, რომელიც მომხმარებელს აუცილებლად მოეწონება.

III თავი. ლოკაცია და რუქა

3.1. Android Location API

Android Location API პირდაპირ არის ჩაშენებული ანდროიდის SDK-ში. იქამდე სანამ Google Play Services წარადგენდა Google Location Services API-ს android.location პაკეტი წარმოადგენდა ერთადერთ შესაძლო საშუალებას ლოკაციის სერვისებთან სამუშაოდ მოწყობილობაში.

მიუხედავად იმისა რომ Google Location Services API უფრო მდგრადი საშუალებაა, ქვემოთ მოყვანილია რამდენიმე მიზეზი, თუ რატომ უნდა გამოვიყენოთ Android Location API:

- ყველა მოწყობილობას არ აქვს დაყენებული Google Play Service-ები;
- Android Location API მოიცავს ისეთ კლასებს, რომლებიც არ აქვს Google Location API-ს. აქედან გამომდინარე საუკეთესო ფუნქციონალის მისაღებად უკეთესი იქნება ორივე საშუალების გამოყენება, რა თქმა უნდა, თუ მოწყობილობა იძლევა ამის საშუალებას.

როგორც დასახელებიდან ჩანს, Google Play Service-ების მწარმოებელი არის Google კომპანია. მისი უმთავრესი მოვალეობაა ჩაერთოს და აკონტროლოს ანდროიდ ოპერაციულ სისტემაში მომხმარებლის აუთენტიფიკაცია, აპლიკაციების განახლება, privacy პარამეტრები და ზოგიერთი ლოკაციის სერვისები.

იგი ასევე მუშაობს მალულად სისტემის ასაჩქარებლად, რუქების გასაუმჯობესებლად და სხვ.

3.2. GPS (Global Positioning System)

გამოყენება

Android Location API გვთავაზობს რიგ საშუალებებს, რათა გამოვიყენოთ ლოკაციის სერვისები ჩაშენებული ფიზიკური GPS დეტალის გამოყენებით.

დღეისათვის თითქმის ყველა მობილურ ტელეფონს აქვს ლოკაციის დასადგენი საშუალებები. ამერიკის შეერთებული შტატების გადაუდებელი დახმარების სერვისები, ისევე როგორც საქართველოს „112“, იყენებს ლოკაციის მიერ მოცემულ ინფორმაციას სერვისის გასაუმჯობესებლად.

თუმცა ყველა მოწყობილობა, რომელიც ანდროიდ ოპერაციულ სისტემაზე მუშაობს, არ არის ტელეფონი. თუ GPS ფუნქია არაა აქტიური, ან ანდროიდ მოწყობილობას არ აქვს GPS ფიზიკური დეტალი, ანდროიდის SDK მაინც გვთავაზობს ლოკაციის დადგენისთვის საჭირო ფუნქციონალს სხვა ალტერნატიული პროვაიდერების საშუალებით.

ალტერნატიულ პროვაიდერებს აქვს თავისი უპირატესობები და ნაკლევანობები ენერგიის მოხმარების, სისწრაფის და სიზუსტის თვალსაზრისით.

3.3. GPS-ის გამოყენება აპლიკაციაში

ისევე როგორც ფიზიკური GPS დეტალი ლოკაციის სერვისებიც არის დამატებითი ფუნქციონალი მოწყობილობისთვის. აპლიკაციის შესაბამისი სათანადო მოთხოვნებისა და უფლებების გაწერა ჩვენ შეგვიძლია ანდროიდ მანიფესტ ფაილში. იმის სათქმელად რომ აპლიკაცია იყენებს ან საჭიროებს ლოკაციის სერვისებს დაგვჭირდება ტეგი <uses-

feature> manifest ფაილიდან. ნახსენები ტეგის საშუალებით Google ჩვენს აპლიკაციას შესთავაზებს მხოლოდ ისეთ მოწყობილობებს რომლებიც აკმაყოფილებენ მითითებულ ტეგში გაწერილ ფიზიკურ თუ ლოგიკურ მოთხოვნებს.

ფრაგმენტი 3.0

```
<uses-feature  
    android:name="android.hardware.location" />
```

თუ აპლიკაცია მოიხმარს მხოლოდ რაიმე მკონკრეტულ ლოკაციის სერვისს, მაგალითად, GPS-ს, ამ შემთხვევაში შესაძლებელია გაიწეროს ტეგი შემდეგნაირად:

ფრაგმენტი 3.1

```
<uses-feature  
    android:name="android.hardware.location.gps"/>
```

3.4. მოწყობილობის ლოკაციის დადგენა

Android Location API საშუალებით ლოკაციის დასადგენად საჭიროა რამდენიმე ეტაპის გავლა და რამდენიმე არჩევანის გაკეთება. ქვემოთ მოყვანილია ეს პროცესი:

- LocationManager კლასის ობიექტის მიღება და getSystemService() მეთოდის გამოძახება LOCATION_SERVICE გამოყენებით;

- შესაბამისი უფლების გაწერა `AndroidManifest.xml` ფაილში, რაც დამოკიდებულია იმაზე, თუ რა სახის ინფორმაციის დადგენას ვაპირებთ;

- პროვაიდერის არჩევა `getAllProviders()` ან `getBestProvider()` მეთოდის გამოყენებით;

- `LocationListener` კლასის იმპლემენტაცია;

- `requestLocationUpdates()` მეთოდის გამოძახება უკვე არჩეული პროვაიდერისა და `LocationListener` ობიექტის გამოყენებით, ადგილმდებარეობის ინფორმაციის მისაღებად.

`LocationManager` კლასის ობიექტის მისაღებად რაიმე სპეციალური ნებართვის გაწერა არაა საჭირო. ნებართვა საჭიროა მხოლოდ რომელიმე შესაძლო პროვაიდერის არჩევისათვის. ქვემოთ მოყვანილი კოდის საშუალებით შეგვიძლია მივიღოთ `LocationManager` კლასის ობიექტი:

ფრაგმენტი 3.2

```
LocationManager locationManager =
    locationManager.getSystemService(MainActivity.LOCATION_SERVICE);
```

შემდეგი ეტაპი არის უფლებების გაწერა აპლიკაციის მანიფესტ ფაილში:

ფრაგმენტი 3.3

```
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

ახლა, როდესაც აპლიკაციას აქვს უფლებები ლოკაციის ინფორმაციის გამოსაყენებლად და LocationManager ობიექტი, შესაძლებელია განისაზღვროს სასურველი პროვაიდერი ლოკაციის მისაღებად.

ფრაგმენტი 3.4

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.NO_REQUIREMENT);
criteria.setPowerRequirement(Criteria.NO_REQUIREMENT);
String bestProvider = locationManager.getBestProvider(criteria, true);
```

setAccuracy() მეთოდს შეუძლია მიიღოს პარამეტრები ACCURACY_COARSE და ACCURACY_FINE, რაც იმას ნიშნავს, რომ აპლიკაციაში წინასწარ უნდა გვექონდეს გაწერილი შესაბამისი უფლება. იმისათვის, რომ განვსაზღვროთ პროვაიდერი, ენერგიის მოხმარების მიხედვით, ჩვენ უნდა გამოვიყენოთ მეთოდი setPowerRequirement() პარამეტრებით POWER_LOW ან POWER_HIGH.

Criteria ობიექტის საშუალებით შესაძლებელია განისაზღვროს პროვაიდერი თუ აწესებს ფასს ლოკაციის დასადგენად და ზოგიერთი სხვა დეტალი. თუ აპლიკაციას აქვს რაიმე განსაკუთრებული მოთხოვნები ზუსტად ესენია რაც უნდა გავწეროთ წინასწარ.

მას შემდეგ რაც getBestProvider() განსაზღვრავს პროვაიდერს, მისი გამოყენებით შესაძლებელი იქნება ლოკაციის დასადგენი ბრძანება. სანამ ეს გაკეთდება, მანამდე აპლიკაციაში აუცილებლად უნდა აღიწეროს LocationListener კლასის შვილობილი კლასი, რომელიც შედგება რამდენიმე მეთოდისაგან: აპლიკაციას შეატყობინოს გაითიშა თუ ჩაირთო რომელიმე პროვაიდერი, ასევე აცნობოს ინფორმაცია

ლოკაციის დადგენისას. ქვემოთ მოყვანილია ბოლოს ნახსენები `onLocationChanged()` მეთოდის ნიმუში:

ფრაგმენტი 3.5

```
@Override
public void onLocationChanged(Location location)
{
    StringBuilder locInfo = new StringBuilder("Current loc = ")
        .append(location.getLatitude()).append(", ")
        .append(location.getLongitude()).append("@ ")
        .append(location.getAltitude()).append(" meters up)\n");
    if (lastLocation != null) {
        float distance = location.distanceTo(lastLocation);
        locInfo.append("Distance from last = ").append(distance)
            .append(" meters\n");
    }
}
```

`onLocationChanged()` მეთოდი პარამეტრად იღებს `Location` ტიპის ობიექტს, რომელსაც აქვს უახლესი ინფორმაცია ლოკაციის შესახებ არჩეული პროვაიდერის მიხედვით. მოცემული ნიმუშით აპლიკაცია ლოკაციას ბეჭდავს ეკრანზე, მათ შორის ზღვის დონიდან სიმაღლეს, ეს ინფორმაცია გვექნება თუ არა, არჩეულ პროვაიდერზეა დამოკიდებული. შემდეგ `distanceTo()` მეთოდის საშუალებით აპლიკაცია ადგენს თუ რა მანძილი გაგვივლია მას შემდეგ, რაც წინა ჯერზე დავადგინეთ ლოკაცია.

ლოკაციის ინფორმაციის დამუშავება პირდაპირ მომხმარებლის მოთხოვნებზეა დამოკიდებული. მოთხოვნა შეიძლება იყოს მრავალფეროვანი: ლოკაციის მისამართად გადაქცევა, რუკაზე ჩვენება, ან Google-ის ჩაშენებული რუკის აპლიკაციის გახსნა, სადაც პინი იქნება დასმული ჩვენს ლოკაციაზე და სხვ.

3.5. ლოკაციის სერვისების გამოყენება Emulator-ში

ანდროიდის ემულატორს აქვს საშუალება გააკეთოს ლოკაციის სერვისების სიმულაცია [19, 22]. მაგრამ, მას არ აქვს ფიზიკური დეტალი რეალური მონაცემის მისაღებად სატელიტიდან. ანდროიდის SDK გვთავაზობს რიგ საშუალებებს, რომ გავაკეთოთ ლოკაციის ინფორმაციის სიმულაცია single location point-ის, GFX ან KML ფაილის გამოყენებით [23,24].

ეს მხოლოდ ემულატორისთვის მუშაობს და არა ფიზიკური მოწყობილობისთვის. ის მოგონილია მხოლოდ ლოკაციაზე დაფუძნებული აპლიკაციების ტესტირებისათვის.

3.6. GeoCoding

გრძელისა და განედის მიღება გამოსადეგია ზუსტი ლოკაციის დადგენისას, ტრეკინგისას და გაზომვებისას, თუმცა, ჩვეულებრივ, ისინი არაა აღწერითი მომხმარებლისათვის.

ანდროიდის SDK გვთავაზობს რამდენიმე დამხმარე მეთოდს იმისათვის, რომ გრძელი და განედი გარდაქმნათ მისამართად და აღვწეროთ ადგილმდებარეობის დასახელებად [19]. ამ მეთოდებს ასევე შეუძლია იმუშაოს პირიქითაც, გარდაქმნას ადგილმდებარეობის დასახელება კოორდინატებად.

Geocoder ტიპის ობიექტი არ საჭიროებს რაიმე სპეციალურ ნებართვას. ქვემოთ მოყვანილი კოდის ბლოკი ასახავს

Geocoder ობიექტის საშუალებით კოორდინატებისგან მივიღოთ მისამართის დასახელება:

ფრაგმენტი 3.6

```
if (Geocoder.isPresent()) {
    Geocoder coder = new Geocoder(this);
    try {
        List<Address> addresses = coder.getFromLocation(
            location.getLatitude(), location.getLongitude(), 3);
        if (addresses != null) {
            for (Address namedLoc addresses) {
                String placeName = namedLoc.getLocality();
                String featureName = namedLoc.getFeatureName();
                String country = namedLoc.getCountryName();
                String road = namedLoc.getThoroughfare();
                locInfo.append(String.format("[%s][%s][%s]
                [%s]\n", placeName, featureName, road, country));
                int addIdx = namedLoc.getMaxAddressLineIndex();
                for (int idx = 0; idx <= addIdx; idx++) {
                    String addLine = namedLoc.getAddressLine(idx);
                    locInfo.append(String.format("Line %d: %s\n", idx,
                        addLine));
                }
            }
        }
    } catch (IOException e) {
        Log.e("GPS", "Failed to get address", e);
    }
    } else {
        Toast.makeText(GPSActivity.this, "No geocoding
        available",
            Toast.LENGTH_LONG).show();
    }
}
```

`getFromLocation()` მეთოდით შეიძლება მივიღოთ ჩვენთვის სასურველი ინფორმაცია, როგორ ნაჩვენებია კოდის ფრაგმენტზე. შესაძენეია ისიც, რომ რაიმე კონკრეტულ ლოკაციას შესაძლოა ჰქონდეს ბევრი მისამართი, რომლებიც გვიბრუნდება `List<Address>` სახით. როგორც წესი, მისამართი, რომელიც პირველი მოხვდება სიაში არის უფრო დეტალური, ვიდრე ყოველი შემდეგი.

პირველი მეთოდები გვაწვდის სპეციფიურ ინფორმაციას, მაგალითად `getFeature()` და `getLocality()`. ისინი გარანტიას ვერ იძლევა, რომ ყოველთვის დაგვიბრუნებს სასარგებლო ინფორმაციას ყველა ლოკაციისთვის. მათი გამოყენება მიზანშეწონილია მხოლოდ მაშინ, როდესაც გვინდა ისეთი ინფორმაციის გაგება, როგორიცაა ქვეყანა.

`getAddressLine()` მეთოდი გამოიყენება მაშინ, როდესაც გვინტერესებს ქუჩების დასახელებები. შესაძლოა დაბრუნებული მისამართები იყოს არასრული. მისამართებში იტერაციების შესასრულებლად შეიძლება გამოყენებულ იქნას `getMaxAddressLineIndex()` და `getMaxAddressLine()` მეთოდები. 3.1 ნახაზზე ნაჩვენებია ლოკაცია მისამართის სამი შემდეგით.

Geocoder ობიექტს შეუძლია მისამართის მიხედვით დაადგინოს გრძედი და განედი. მოცემულ მისამართებს შეუძლია დააბრუნონ ზუსტი კოორდინატები: “Eiffel Tower”, “London, UK”, “IceLand”, “BOS”, “Yellowstone” და სხვ.

ყოველთვის არსებობს ალბათობა, რომ მივიღებთ ერთზე მეტ შედეგს. მისაღები ფორმაა აპლიკაციაში გვექონდეს შედეგიდან მისამართის ასარჩევი საშუალება.



ნახ.3.1

სხვა კარგი საშუალება იმისა, რომ ვაჩვენოთ მომხმარებლის მიერ შეყვანილი მისამართი არის რუკა.

Geocoding ოპერაციას როგორც წესი სჭირდება ინტერნეტთან კავშირი და ეს პროცესი არ უნდა ხდებოდეს აპლიკაციის მთავარ არხზე. საჭიროა მისთვის დამოუკიდებელი ახალი არხის შექმნა რათა არ გამოვიწვიოთ აპლიკაციის გაშეშება.

3.7. მეტის გაკეთება Location-Based სერვისებისგან

ჩვენ უკვე გავეცანით ბევრ სხვადასხვა ლოკაციის ხელსაწყოებს Android Location API-დან. მაგრამ არსებობს კიდევ დამატებით სასარგებლო სხვა ფუნქციონალიც [19,103].

LocationManager კლასი გვთავაზობს შეტყობინებების სისტემას, რომელიც აქტიურდება რაიმე ლოკაციასთან მიახლოებისას, რაც შეგვიძლია გამოვიყენოთ მაშინ, როდესაც მომხმარებელი უახლოვდება რომელიმე მიმართულებას, სანადირო ადგილებს ან სხვა, მომხმარებლისთვის რაიმე საინტერესო ლოკაციას.

GpsStatus, GpsStatus.Listener და GpsSatellite კლასები გვაწვდის მრავალ სასარგებლო ინფორმაციას GPS სატელიტების შესახებ, რომელსაც GPS ძრავა იყენებს. GpsStatus და Listener ქვეკლასი ახდენს GPS ძრავის მონიტორინგს და გვაძლევს გამოყენებული gps სატელიტების ჩამონათვალს.

GpsSatellite კლასი წარმოადგენს თითოეული სატელიტის შესახებ მიმდინარე მდგომარეობას და გვაძლევს ინფორმაციას მისი სიმაღლის შესახებ.

3.8. Google Location Services API

Google Location Services დამატებულია Google Play Services SDK-ში და თავსებადია იმ მოწყობილობებთან რომლებსაც დაინსტალებული აქვს Google Play და ანდროიდის ვერსია არის 2.2 ან მეტი [27,28]. თუ ემბედ მარტივ გზას ლოკაციის ფუნქციონალის დასამატებლად აპლიკაციაში, უკეთესი სიზუსტითა და დაბალი ენერგიის მოხმარებისთვის შეიძლება Google Location Services გამოყენება.

3.9. Fused Location Provider

ლოკაციის ფუნქციონალი გახდა ინტეგრირებული Google Play Services-ებში [25]. დეველოპერი აღარ წერს ლოკაციის ფუნქციონალს თვითონ. Google Play Services თვითონ აკეთებს ამას fused location provider-ის დახმარებით [26]. დეველოპერს აღარ სჭირდება ყურადღება მიაქციოს ენერგიის მოხმარებას და ასევე მოახდინოს კარგი სიზუსტის მქონე ლოკაციის დადგენა პატარა კოდის დახმარებით.

fused location provider-ის გამოსაყენებლად საჭიროა რამდენიმე ეტაპი:

- Android manifest ფაილში საჭირო ნებართვის დამატება, სიზუსტის წინასწარ განსაზღვრა, რომელსაც აპლიკაცია მოითხოვს, როგორებიცაა ACCESS_FINE_LOCATION და ACCESS_COARSE_LOCATION;

- აუცილებელია წინასწარ შემოწმდეს არის თუ არა დაინსტალირებული Google Play Services მოწყობილობაში. წინააღმდეგ შემთხვევაში აპლიკაცია შეწყვეტს მუშაობას. ასეთი შემთხვევა კი მომხმარებელში გამოიწვევს უკმაყოფილებას, რასაც მოყვება ცუდი შეფასებები მათი მხრიდან;

- საჭიროა გავაკეთოთ შესაბამისი მეთოდების იმპლემენტაცია onConnectionFailedListener და ConnectionCallbacks შემთხვევებისთვის. onConnectionFailedListener() შემთხვევისას უნდა აღვწეროთ მეთოდი onConnectionFailed(), ხოლო ConnectionCallbacks კლასისთვის დაგვჭირდება onConnected() და onDisconnected() მეთოდები;

- ლოკაციის კლიენტის შექმნა LocationClient კლასის გამოყენებით აპლიკაციის onCreate() მეთოდში;

- გამოვიყენოთ `getLastLocation()` მეთოდი ლოკაციის კლიენტიდან, რათა მივიღოთ მოწყობილობის იმჟამინდელი ლოკაციის ინფორმაცია. აპლიკაციის ნებართვისა და მომხმარებლის მიერ მოწყობილობაში გააქტიურებული ლოკაციის ფუნქციების შესაბამისად, მოწყობილობა გადაწყვეტს - რა ტიპის ლოკაციის ინფორმაცია მოგვაწოდოს.

3.10. მეტის გაკეთება Google Location Services-ით

Fused location provider არის პირველი განვითარება Google Location Services-დან. გარდა ამისა არსებობს კიდევ 2 სხვა API, რომლის ცოდნა სასურველია, ესენია მოქმედების ამოცნობის API და Geofencing API.

3.10.1. მოქმედების ამოცნობა

მოქმედების ამოცნობის API გამოიყენება მომხმარებლის მიმდინარე მოქმედების დასადგენად. ქვემოთ მოყვანილია აქტივობების სია, კონსტანტები თავისი აღწერით, რომელთა დადგენაც აპლიკაციას შეეძლება:

- `IN_VEHICLE` – ავტომობილით მოძრაობა;
- `ON_BICYCLE` – ველოსიპედით მოძრაობა;
- `ON_FOOT` – სეირნობა ფეხით ან სირბილი;
- `STILL` – მომხმარებელი გაჩერებულია;
- `TILTING` – მომხმარებელმა ტელეფონი გადახარა;
- `UNKNOWN` – იმ შემთხვევაში, თუ მომხმარებლის მიმდინარე აქტივობა არ ემთხვევა არცერთ ზემოთ ჩამოთვლილს, მაშინ მისი აქტივობა არის `UNKNOWN`.

3.10.2. Geofencing API

Geofence არის მოქმედების არეალი რომელსაც აპლიკაცია განსაზღვრავს იმის დასადგენად, მომხმარებელმა გადაკვეთა თუ არა არეალის საზღვრები. Geofence-ის ასაწყობად აპლიკაციაში დაგვჭირდება რამდენიმე ეტაპი:

- Android manifest ფაილში საჭირო ნებართვის დამატება;
- აუცილებელია წინასწარ შემოწმდეს, არის თუ არა დაინსტალირებული Google Play Services, რადგან Geofencing API ხელმისაწვდომია მხოლოდ მაშინ, როდესაც ტელეფონს იგი აქვს;
- Geofencing სთორიჯი მექანიზმის განსაზღვრა Shared-Preferences საშუალებით;
- Geofence შექმნა გრძელის, განედის და რადიუსის დახმარებით;
- კონტროლის დაწყება, როდესაც მომხმარებელი შევა თუ გამოვა Geofence არეალიდან;
- შევწყვიტოთ დაკვირვება და წავშალოთ Geofence.

3.11. Google Maps Android API v.2

მოწყობილობის ლოკაციის დადგენა ან რაიმე გეოგრაფიული კოორდინატები მომხმარებლისთვის არ წარმოადგენს აღწერილობით ინფორმაციას.

ამიტომ უმჯობესია მოცემული ინფორმაცია ვაჩვენოთ რუკაზე, რომელიც მომხმარებლისათვის იქნება მისაღები.

საბედნიეროდ Google გვთავაზობს Map API-ის ანდროიდისათვის [29-33].

3.11.1. ლოკაციის მაპინგი

Android SDK გვთავაზობს 2 მეთოდს, რათა ვაჩვენოთ ლოკაცია Google-ის რუკაზე.

პირველი მეთოდია გამოვიყენოთ ლოკაციის URI მოწყობილობაში ინტეგრირებული რუკის აპლიკაციის გასახსნელად, რომელზეც ვაჩვენებთ ლოკაციას.

მეორე მეთოდია გამოვიყენოთ MapFragment ჩვენს აპლიკაციაში და ჩვენი კოორდინატები ვაჩვენოთ ჩვენსავე აპლიკაციაში.

3.11.2. Mapping Intents

წინა პარაგრაფებში ჩვენ განვიხილეთ თუ როგორ უნდა დავადგინოთ მისამართის მიხედვით კოორდინატები. ქვემოთ მოყვანილია კოდის ნიმუში რომლითაც გაიხსნება მოწყობილობაში ინტეგრირებული რუკის აპლიკაცია:

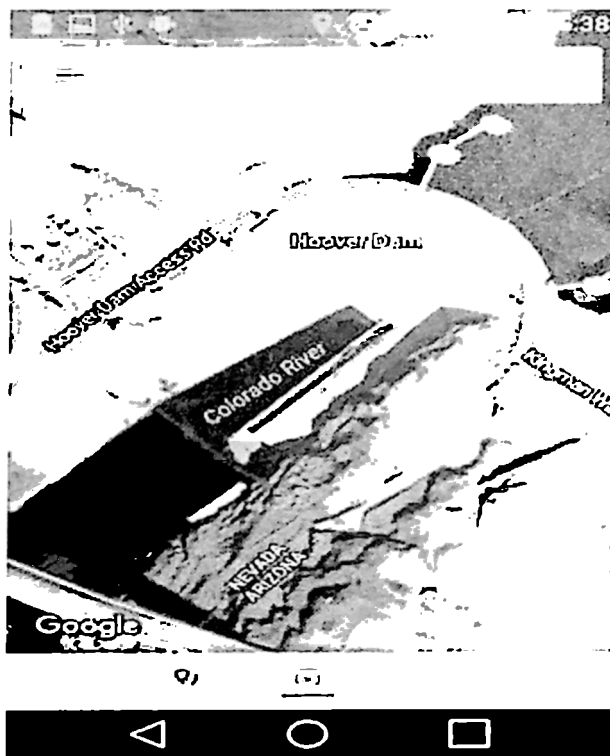
ფრაგმენტი 3.7

```
String geoURI = String.format("geo:%f,%f",  
    lat, lon);  
Uri geo = Uri.parse(geoURI);  
Intent geoMap = new  
Intent(Intent.ACTION_VIEW, geo);  
startActivity(geoMap);
```

პირველი ეტაპი არის შეიქმნას ტექსტური ბრძანება URI სახით, რომელსაც მიიღებს რუკის აპლიკაცია. ამ შემთხვევაში არის "geo: ", რომელსაც შემდეგ მოყვება გრძელი და განედი. შემდეგ URI გამოიყენება ახალი Uri ობიექტის შესაქმნელად ACTION_VIEW ინტენთისთვის. საბოლოოდ ვიძახებთ startActivity() მეთოდს. იმ შემთხვევაში თუ გრძელი და განედი დასაშვებია. თუ მოვებზნით, „Hoover Dam“ 3.2 ნახაზზე ნაჩვენებია შედეგი.

ამ მეთოდის გამოყენებით მაპინგი გაუშვებს მოწყობილობაში რუკის აპლიკაციას - ამ შემთხვევაში Google Map-ს.

დრო რომ არ დავხარჯოთ რუკის სრული ფუნქციონალის ახლიდან დასაწერად ან თუ არ გვჭირდება იმაზე მეტი ფუნქციონალი ვიდრე გვთავაზობს Google Map, ამიტომ სწრაფი და მარტივი გზა იქნება გამოვიყენოთ Google Map.



ნახ 3.2

3.11.3. რუკის API KEY-ს მოპოვება

აპლიკაციაში რუკაზე მუშაობის დაწყებამდე, პირველი რაც გვჭირდება, ეს არის Google-სგან მივიღოთ Maps API Key. გასაღები გენერირებულია SHA-1 სერთიფიკატის ანაბეჭდით, რომელსაც ვიყენებთ აპლიკაციაში [33-35].

ტესტირებისთვის ჩვენ შეგვიძლია გამოვიყენოთ debug სერთიფიკატი, რომლის შექმნა შეუძლია Android SDK-ს.

საწარმოო ვერსიისთვის კი საჭიროა გამოვიყენოთ ზემოთ ნახსენები Google-ის მიერ მოცემული სერთიფიკატი.

აპლიკაციისათვის შესაბამისი API Key რომ დავაგენერიროთ, საჭიროა შემდეგი ეტაპები:

- დავამზადოთ debug სერთიფიკატისთვის SHA-1 სერთიფიკატი;
- გავიაროთ ავტორიზაცია Google Developer Console და ავირჩიოთ ის პროექტი, რომლისთვისაც გვჭირდება API Key;
- დავრწმუნდეთ, რომ სერვისების ბმულის ქვეშ ჩართულია Google Maps API v.2;
- გადავიდეთ API Access ბმულზე და Simple API Access-ქვეშ დავაჭიროთ ღილაკს Create new Android key;
- პირველ ეტაპზე დაგენერირებული API Key დავაკოპიროთ და ჩავსვათ მიმდინარე გვერდზე. გავყვეთ გვერდის ინსტრუქციებს აპლიკაციის Android key-ს კონფიგურაციისთვის და ბოლოს დავაჭიროთ ღილაკს Create.

პირველი ნაბიჯი უკვე გადადგმულია აპლიკაციის დეველოპმენტში. ყველა პლატფორმაზე სერთიფიკატის დასახელება იქნება debug.keystore თუ ჩვენ მას სახელს არ შევუცვლით. თუ თქვენ იყენებთ Android IDE-ს, მაშინ ფაილის შექმნისას მისი მისამართი იქნება Android Build Preference-

ების ქვეშ. ამ ფაილის გამოყენებით თქვენ უნდა გაუშვათ შემდეგ ბრძანება:

ფრაგმენტი 3.8

```
keytool -list -keystore /path/to/debug.keystore
-storepass android
```

შედეგად მივიღებთ SHA-1 სერთიფიკატს, რომელიც დაგვჭირდება მეხუთე ეტაპზე ჩასასმელად. Android SDK-ის მიერ დაგენერირებული Debug keystore ფაილის ვადა არის 1 წელი და არის უნიკალური მხოლოდ მისი კომპიუტერისათვის. რეკომენდაციაა, რომ გავაკეთოთ 1 წელზე მეტვადიანი სერთიფიკატი რომელსაც გამოიყენებს ჯგუფის თითოეული წევრი ფგუფური მუშაობისას. გუნდური მუშაობისთვის შექმნილი Google Maps Android API key ძლებს გაცილებით მეტ ხანს. საბედნიეროდ ამის გაკება ძალიან ადვილია keytool command-line-ის საშუალებით. ქვემოთ მოცემული ბრძანება დაგვხმარება ამის გაკეთებაში:

ფრაგმენტი 3.9

```
keytool -genkey -keypass android -keystore
debug.keystore
alias androiddebugkey -storepass android
-validity 10000
-dname "CN=Android Debug,O=Android,C=US"
```

ამ ფრაგმენტში მოყვანილი ბრძანების საშუალებით ჩვენ დავაგენერირებთ debug.keystore ფაილს, რომლის გამოყენებასაც შეძლებს ჯგუფის თითოეული წევრი და მისი ვადა იქნება 10 000 დღე. შექმნის შემდეგ საჭიროა მისი პროექტზე

მიმაგრება, თუ იგი განთვსებულია ჩვენ მიერ მითითებულ მისამართზე.

როდესაც წარმატებით დავაგენერირებთ API Key-ს ჩვენ უნდა გავწეროთ იგი აპლიკაციის მანიფესტ ფაილში. ამისათვის საჭიროა <meta-data> ტეგში გავწეროთ შემდეგნაირი კოდი რომელიც ფრაგმენტ 1.10-ზეა ნაჩვენები.

ფრაგმენტი 3.10

```
<meta-data
    android:value="API_KEY"
    android:name="com.google.android.maps
        .v2.API_KEY" />
```

თუ თქვენ მუშობთ დეველოპმენტის ბევრ მანქანაზე ან ხართ გუნდის წევრი, მაშინ აუცილებლად დაგჭირდებათ ისეთი debug თითოეული მათგანი. ალტერნატიულად თქვენ შეგიძლიათ გადაიტანოთ სერთიფიკატი ერთი კომპიუტერიდან დანარჩენ ყველა კომპიუტერზე რის შემდეგაც Android Maps API Key იქნება ყველგან დასაშვები. ამ გზით თქვენ დაზოგავთ დროს რადგან ასე არ დაგჭირდებათ კოდში ჩარევა.

3.11.4. Map Fragments

MapFragment არის მომხმარებლის ინტერფეისი კომპონენტი, რაც გვამლევს საშუალებას დავამატოთ რუკა აპლიკაციაში fragment-ის სახით. საჭიროა უბრალოდ შევქმნათ layout, რომელიც შეიცავს MapFragment (ან SupportMapFragment თუ ჩვენ ვიყენებთ Support Library-ს ბიბლიოთეკას). კოდის ფრაგმენტი 3.11 წარმოგივიდგენს რუკის დამატებას layout-ში.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/
s/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <fragment
        android:id="@+id/map"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"

class="com.google.android.gms.maps.SupportMapFra
gment" />

</LinearLayout>
```

ამის შემდეგ საჭიროა გავაკეთოთ კლასი რომლიც მემკვიდრე იქნება FragmentActivity კლასის და ამ კლასში უნდა მოვახდინოთ წვდომა ჩვენ მიერ მომხმარებლის გვერდზე ჩასმულ რუკაზე getFragmentManager() ან getSupportFragmentManager() და getMap() მეთოდებით Fragment კლასიდან. კოდის ფრაგმენტი 3.12 წარმოგივლიდგენს რუკის მიმართებით მოპოვებას პროგრამული კოდიდან:

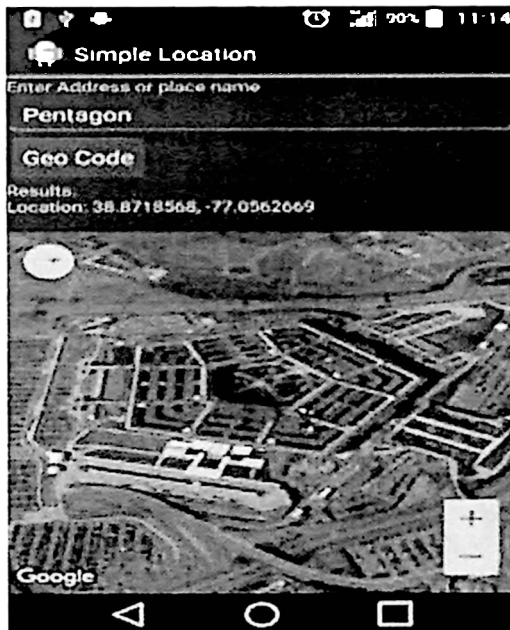
```
public class MappingActivity extends
                                FragmentActivity {
    private GoogleMap map;

    protected void onCreate(Bundle
                            savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mapping);

        map = ((SupportMapFragment)
getSupportFragmentManager()
        .findFragmentById(R.id.map)).getMap();
    }
}
```

ახლა უკვე, როდესაც გვაქვს რუკა ჩვენს აპლიკაციაში, ჩვენ მზად ვართ ვაჩვენოთ მნიშვნელოვანი ინფორმაცია მომხმარებელს. რუკა ნაჩვენებია 3.3 ნახაზზე.

ნახ.3.3



3.12. ლოკაციის მარკირება

დგება დრო, როდესაც გვჭირდება რაიმე გეოგრაფიული ლოკაციის მარკირება ანუ მონიშვნა ეკრანზე მისამართის უკეთესად გარჩევადობის მიზნით. Google Maps გვთავაზობს მოხერხებულ მეთოდებს მარკერების დასამატებლად რუკაზე რაიმე კონკრეტული ლოკაციისთვის,

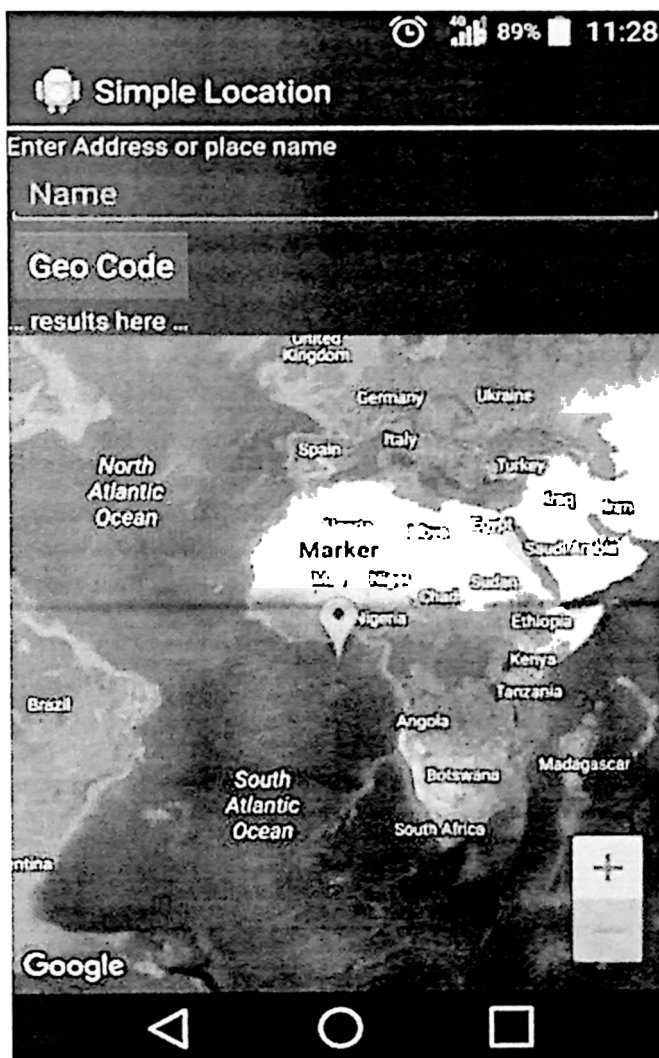
ფრაგმენტი 3.13

```
map.addMarker(new MarkerOptions()  
    .position(new LatLng(0, 0))  
    .title("Marker"));
```

პირველ რიგში რუკის ობიექტისგან უნდა გამოვიძახოთ addMarker() მეთოდი და მას გადავცეთ MarketOptions ტიპის ობიექტი, ჩვენთვის სასურველ გრძედთან და განედთან ერთად, ასევე მარკერის სახელი. როდესაც მომხმარებელი დააჭერს მარკერს, მაშინვე გამოჩნდება მისი დასახელება რუკაზე, რაც ჩვენ გადავცით title() მეთოდში (ნახ.3.4).

3.13. პოზიცია და რუკის ანიმაცია

ზოგჯერ დგება დრო როდესაც მომხმარებელს უნდა რუკაზე ერთი ადგილიდან მეორე ადგილზე გადაადგილება. რუკა არის viewable ტიპის ობიექტი რაც ცნობილია როგორც კამერა, რომლის პოზიციის მასშტაბი შეიძლება შეიცვალოს, გადატრიალდეს, დაიხაროს ან ამოძრავდეს. ფრაგმენტი 3.14 წარმოგვიდგენს კოდს რომლის საშუალებითაც რუკის კამერა გადაადგილდება ერთი ადგილიდან მეორეზე, შეიცვლება მასშტაბი, გადატრიალდება, და დაიხრება:



ნაბ.3.4

```

CameraPosition.Builder destBuilder = new
CameraPosition.Builder();
CameraPosition dest = destBuilder.target(new
    LatLng(lat, lon))
        .zoom(15.5f)
        .bearing(300)
        .tilt(50)
        .build();

map.animateCamera(CameraUpdateFactory.newCameraP
osition(dest));
    
```

პირველად საჭიროა `CameraPosition.Builder()` ობიექტის შექმნა. შემდეგ უნდა შეიქმნას მისამართი, საით გვინდა, რომ კამერა წავიდეს, შემდეგ მასშტაბის ცვლილება, ორიენტაცია და დახრა. შემდეგ ვიძახებთ მეთოდს `Build()` `CameraPosition` ტიპის ობიექტის შესაქმნელად. საბოლოოდ რუკის ტიპის ობიექტისგან ვიძახებთ `animateCamera()` მეთოდს რომელსაც გადაეცემა `CameraUpdateFactory` ობიექტი ახალი მისამართით.

3.14. მესამე თავის დასკვნა

Google Maps for Android v2 ხელმისაწვდომია დეველოპერებისათვის, რომლებიც არეგისტრირებენ API Key-ს. რუკების გამოყენება ანდროიდ აპლიკაციებში მას ხდის უფრო ინფორმაციულად მდიდარს და საინტერესოს. დეველოპერები, აპლიკაციის მოთხოვნებიდან გამომდინარე, რუკას ან ჩასვამენ საკუთარ აპლიკაციაში ან გახსნიან მოწყობილობაში ჩაშენებულ რუკის აპლიკაციას. Android Location Services API, Google Location Services API და Google Maps Android v2 services API გვთავაზობს მრავალ შესაძლებლობას ჩვენი ანდროიდ აპლიკაციის გასავითარებლად.

IV თავი

მობილური აპლიკაციის აგება ექსტრემალური სიტუაციებისათვის 112-ის მაგალითზე

4.1. გადაუდებელი საჭიროების (დახმარების) სიტუაციებში 112-თან დაკავშირების არხები

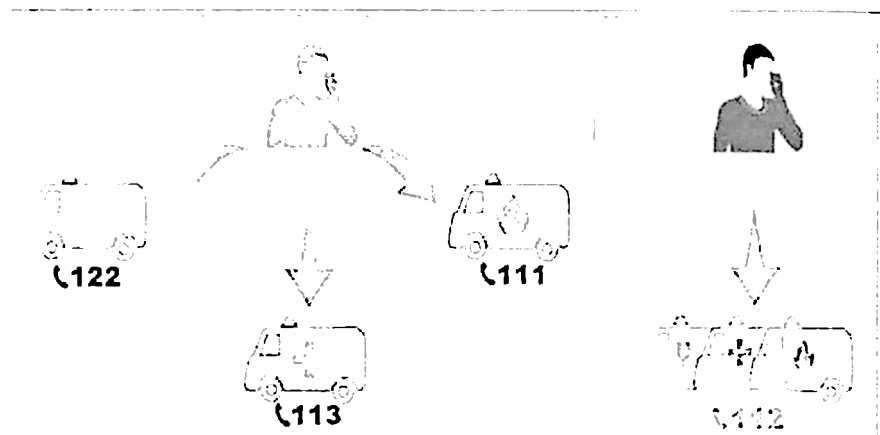
„112“ არის ევროპის ქვეყნებში მოქმედი სატელეფონო ნომერი, რომელიც 2012 წელს გადაუდებელი დახმარებისა და საგანგებო სიტუაციების ეფექტიანი სისტემის ჩამოყალიბების მიზნით შეიქმნა [36,37]. 112 საქართველოს ნებისმიერი წერტილიდან, 24/7 საათის განმავლობაში, იღებს გადაუდებელი დახმარების საჭიროების შესახებ სატელეფონო შეტყობინებებს ერთიანი სატელეფონო ნომრის „1-1-2“ საშუალებით. დამუშავებული ინფორმაცია ოპერატიულად გადაეცემა:

პოლიციას;

სახანძრო / სამაშველო სამსახურს;

- სასწრაფო დახმარების ცენტრს.

112-ის ამოქმედებამდე, საქართველოში არსებობდა გადაუდებელი დახმარების სამი ნომერი, რომლებიც კოორდინირებული სისტემის გარეშე, მოძველებული ტექნოლოგიებით მუშაობდა (ნახ.4.1). ახალმა სისტემამ საგრძნობლად შეამცირა გადაუდებელი დახმარების აღმოჩენის დრო და გაზარდა მომსახურების ხარისხი [38].



ნახ.4.1. „112“ - ევროპის ქვეყნებში მოქმედი გადაუდებელი დახმარების უნიკალური ნომერი

მომდევნო პარაგრაფებში განვიხილავთ იმ ალტერნატიულ არხებს, რომლებიც უკვე არსებობს ან შესაძლოა გაკეთდეს, რათა 112-თან წვდომა იყოს უფრო მარტივი სწრაფი და ხელმისაწვდომი.

4.1.1. „112“-თან დაკავშირება ხმოვანი ზარის საშუალებით

112-თან დაკავშირების უპირველესი საშუალება იყო ხმოვანი ზარი, რომელიც არის სრულიად უფასო და შესაძლებელია განხორციელდეს ყველა მოწყობილობიდან რომელსაც შეუძლია წვდომა ქსელში [39-41]. ასეთი მოწყობილობები შეიძლება იყოს ფიქსირებული და

მობილური ტელეფონები. ზარის განხორციელება შესაძლებელია:

- *ფიქსირებული ტელეფონიდან, მაშინაც კი თუ ტელეფონი ორმხრივად არის გათიშული (!);*
- *მობილური ტელეფონიდან, მაშინაც კი თუ ტელეფონი ორმხრივად არის გათიშული ან არ დევს სიმ-ბარათი (!).*

ტელეფონი აუცილებლად უნდა იყოს მობილური ოპერატორების დაფარვის ზონაში ანუ შეძლოს GSM სერვისებზე წვდომა. სიმ-ბარათი არის ქსელის ფილტრი, რომელსაც გააჩნია ინფორმაცია თუ რომელ ქსელს ეკუთვნის და იმ შემთხვევაში თუ მოვხდება შესაბამისი ოპერატორის დაფარვის ზონაში ტელეფონი, რომელშიც დევს ავტომატურად დარეგისტრირდება მოცემულ ქსელში.

სიმ-ბარათის საშუალებით ხდება აუთენტიფიკაცია ქსელში და ასევე შეიცავს სერვისებით სარგებლობაზე გადასახადების შესახებ ინფორმაციას, იმისათვის რომ მობილურმა ტელეფონმა შეძლოს ქსელში დარეგისტრირება, სიმ-ბარათი საჭირო არაა. ორივე შემთხვევაში, ტელეფონს სიმ ბარათის გარეშე ან სიმ-ბარათით, ჩაპროგრამებული აქვს გადაუდებელი დახმარების ნომრები. როდესაც მომხმარებელები ახორციელებს ზარს 112-ის მიმართულებით, მობილური ოპერატორი, რომელშიც არის დარეგისტრირებული იმ მომენტში ტელეფონი ავტომატურად გადაამისამართებს შესაბამის ორგანიზაციაში, ამ შემთხვევაში 112-ის ქოლ-ცენტრში.



112

112-ის სერვისებით სარგებლობისთვის, უბრალოდ უნდა დავრეკოთ ნომერზე „112“ (ნახ.4.2).

1	2	3
4	5	6
7	8	9
*	0	#

ნახ.4.2



ხმოვანი ზარის უპირატესობანი:

- შესაძლებელია ზარის განხორციელება ნებისმიერი ტელეფონიდან;
- მარტივია მოსახმარებლად;
- სჭირდება ცოტა დრო პრობლემის იდენტიფიკაციისთვის;
- უფასოა.

ხმოვანი ზარის ნაკლოვანებები:

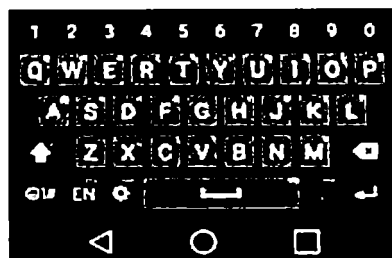
- შეუძლებელია ზარის განხორციელება, სადაც მობილური ოპერატორებიდან არცერთის დაფარვის ზონა არ არის;
- რთულია ზუსტი ადგილმდებარეობის დადგენა;
- სერვისებით სარგებლობას ვერ შეძლებენ ყრუ და სმენადაქვეითებული ადამიანები.

4.1.2. SMS და ვიდეო ზარი

112-ის მთავარი მიზანი არის ის, რომ რაც შეიძლება მეტმა ადამიანმა შეძლოს დახმარების მიღება. იმისათვის, რომ უნარ-შეზღუდულ ადამიანებს შეეძლოთ სერვისის გამოყენება, ევროპის მრავალმა ქვეყანამ, მათ შორის საქართველომ, შეძლო დაენერგა SMS და ვიდეო ზარი, რომლის საშუალებითაც ყრუ და სმენადაქვეითებულ ადამიანებმა შეძლონ სერვისების მიღება.

იმისათვის, რომ ადამიანმა შეძლოს SMS სერვისით სარგებლობა, საჭიროა დარეგისტრირდეს 112-ის ვებ გვერდზე და ამის შემდეგ გამოგზავნოს შეტყობინება ნომერზე „112“

ნახ.4.3.



- SMS და ვიდეო ზარის უპირატესობანი:

- შესაძლებლობას ქმნის რომ ყრუ და სმენადაქვეითებულმა ადამიანებმა შეძლონ 112-ის სერვისების გამოყენება;
 - შესაძლოა განახორციელო ჩუმი შეტყობინება.
- SMS და ვიდეო ზარის ნაკლოვანებები:
- შეტყობინების მიღება და დამუშავებისთვის საჭირო დრო დიდია.

4.1.3. მობილური აპლიკაცია

4.1.3.1. პრობლემის აღწერა

„112“-ის მისიაა შევამციროთ გადაუდებელი დახმარების შეტყობინების მიღების დრო. გადაუდებელი დახმარების შესახებ მიღებული შეტყობინების უმოკლეს დროში დამუშავების თვალსაზრისით, უდიდესი მნიშვნელობა ენიჭება ინიციატორის ზუსტი ადგილმდებარეობის განსაზღვრას.

არის შემთხვევები, როდესაც რთულია შეტყობინების ინიციატორის ლოკაციის დადგენა იმ შემთხვევაში, თუ იგი იმყოფება დაუმისამართებელ ადგილას ან ვერ ახერხებს ადგილმდებარეობის ზუსტ იდენტიფიცირებას.

გარდა აღნიშნულისა, არის შემთხვევები, როდესაც არსებული სიტუაციის სპეციფიკიდან გამომდინარე, ინიციატორი ვერ ახერხებს ოპერატორთან სატელეფონო საუბარს.

4.1.3.2. მთავარი მიზანი

პროექტის მთავარი მიზანია, თანამედროვე ტექნოლოგიების და ინოვაციების გათვალისწინებით, დაინერგოს „112“-თან დაკავშირების ალტერნატიული არხი, რომელიც

ხელს შეუწყობს მოქალაქეთა ხელმისაწვდომობის გაზრდას და გადაუდებელი დახმარების უმოკლეს დროში მიღებას.

4.1.3.3. სასურველი შედეგები

მოქალაქეთა ჩართულობით, „112“-თან დაკავშირების ალტერნატიული არხის (მობილური აპლიკაციის) დანერგვა, რომელიც ხელს შეუწყობს გადაუდებელი დახმარების შესახებ შეტყობინებების უფრო სწრაფად დამუშავებას.

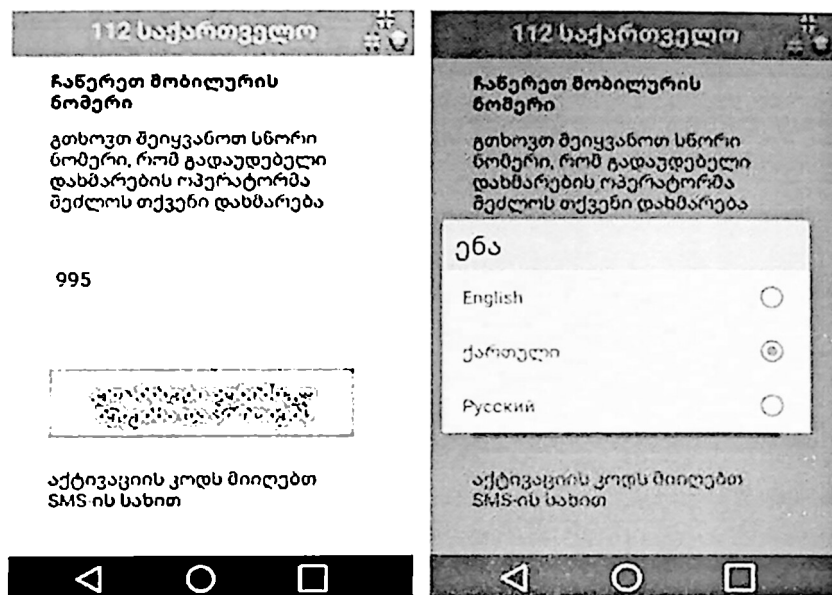
4.1.3.4. აპლიკაციის ფუნქციები

- მრავალენოვანი მხარდაჭერა;
- რეგისტრაცია;
- განგაშის ღილაკი;
- ოპერატორთან ჩატი;
- ზარის განხორციელება;
- კოორდინატების დადგენა;

112-თან სხვადასხვა ტიპის არხებით დაკავშირების საშუალება.

აპლიკაციის თავდაპირველი ჩართვისას მომხმარებელს გამოუჩნდება ნომრის სარეგისტრაციო გვერდი.

იმ შემთხვევაში თუ საჭირო იქნება ენის შეცვლა, ამავე გვერდზე იქნება ღილაკი საიდანაც საშუალება იქნება აირჩიოს მისთვის სასურველი ენა, რათა შემდგომში შეძლოს აპლიკაციის მოხმარება (ნახ.4.4).



ნახ.4.4

სარეგისტრაციო გვერდზე იქნება ღილაკი წესებისა და პირობების სანახავად. ასევე იქნება ღილაკი რომლითაც დარეგისტრირდება მითითებული ნომრით „112“-ის მონაცემთა ბაზაში.

ამავე ნომრით იხელმძღვანელებს აპლიკაცია შემდგომი გამოყენებისას. როდესაც მოხდება დარეგისტრირების ღილაკზე დაჭერა ნომრის შემოწმების მიზნით „112“ გამოგზავნის ერთჯერად SMS კოდს, რის შემდეგაც დასრულდება რეგისტრაცია.

აპლიკაციის გამოსაყენებლად აუცილებლად საჭიროა ინტერნეტ-კავშირი, წინააღმდეგ შემთხვევაში აპლიკაცია

ამოაგდებს შეტყობინებას ინტერნეტის არ არსებობის შესახებ. ასევე აუცილებელია მომხმარებელმა სრულად ჩაწეროს პირადი მობილური ნომერი. ნომრის არასრულად ჩაწერის შემთხვევაში აპლიკაცია შესაბამის შეტყობინებას აჩვენებს მომხმარებელს.

რეგისტრაციის წარმატებით დასრულების შემდეგ აპლიკაცია გადავა მთავარ გვერდზე, სადაც მომხმარებელი შეძლებს სრული ფუნქციონალით სარგებლობას.

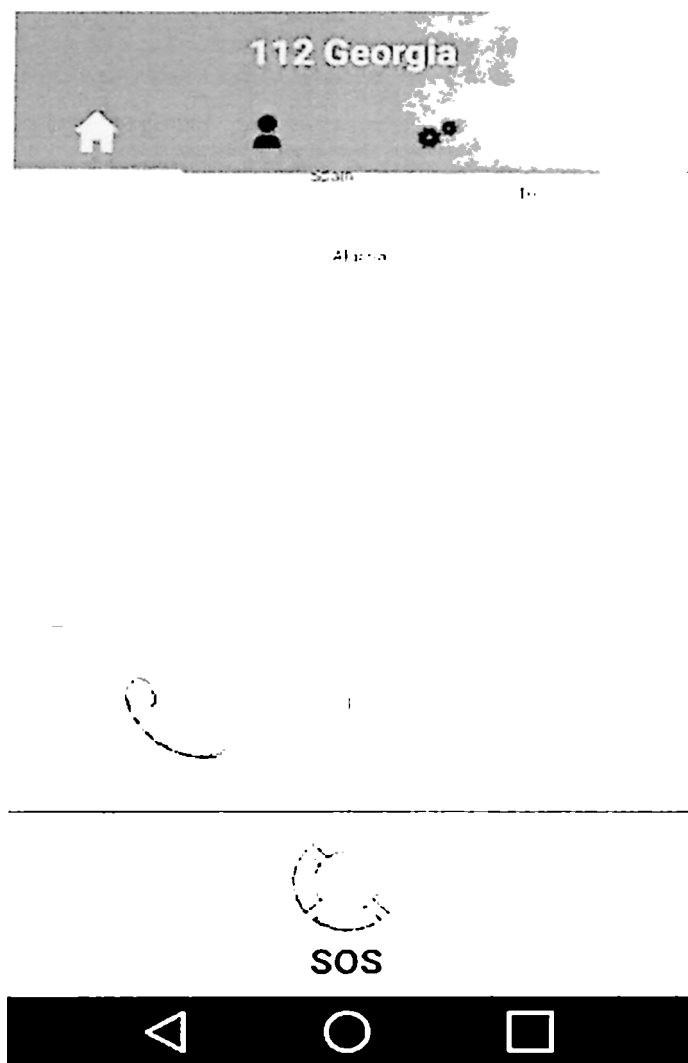
მთავარი გვერდის ზედა ნაწილში იქნება ოთხი TAB-ით გამოყოფილი ფანჯარა, სადაც თითოეულ მათგანში თემატურად იქნება დალაგებული ფუნქციონალი.

პირველი გვერდი ჩანართი იქნება „მთავარი“, მეორე - „მომხმარებლის პროფილი“, მესამე - პარამეტრები, მეოთხე - „სხვა“

მთავარ ჩანართში იქნება რუკა, და ქვედა ნაწილში განლაგებული სამი ღილაკი: „ზარის განხორციელება“, „ჩატი 112-ის ოპერატორთან“ და „ჩუმი განგაში“ (ნახ.4.5).

აპლიკაცია ავტომატურ რეჟიმში დაიწყებს ადგილმდებარეობის დადგენას.

იმ შემთხვევაში თუ მობილურში ჩართული არ არის ლოკაციის სერვისები, მაშინ ამოდის შესაბამისი ფანჯარა, სადაც მომხმარებელს სთხოვს ჩართოს ზემოთ ნახსენები სერვისები და გადაჰყავს შესაბამის განყოფილებაში მობილურის სისტემურ პარამეტრებში.



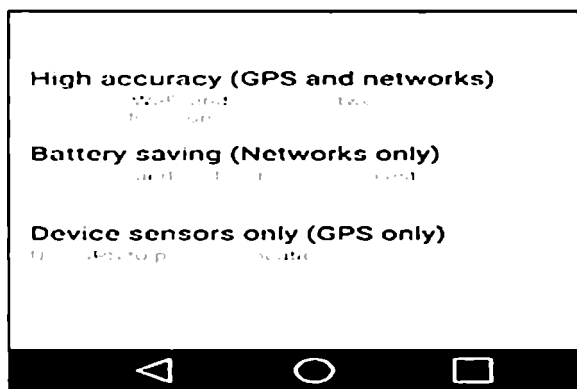
ნახ.4.5

დღეისათვის მობილურ მოწყობილობათა უმეტესობას აქვს GPS (Global Positioning System) მხარდაჭერა.

გლობალური ადგილმდებარეობის განმსაზღვრელი სისტემა გვაწვდის საიმედო ადგილმდებარეობის განსაზღვრას, ნავიგაციასა და განრიგის სერვისებს მსოფლიოს მასშტაბით. იგი მუშაობს ნებისმიერ ამინდში, დღესა თუ ღამეში, ყველგან დედამიწაზე ან დედამიწასთან ახლოს.

GPS დამზადებულია სამი ნაწილისგან: 24-დან 32-მდე დედამიწის ორბიტის გარშემო მოძრავი თანამგზავრი, ოთხი საკონტროლო სადგური დედამიწაზე და GPS მიმღები, რომელიც მომხმარებელს უნდა გააჩნდეს.

GPS სატელიტები კავშირის სიგნალს აძლევს სივრციდან, სადაც გამოყენებულია მომხმარებლის მიერ GPS მიმღები იმისთვის, რომ მიიღოს სამგანზომილებიანი მდებარეობა (განედი, გმედი, სიმაღლე) და დრო (ნახ.4.6).



ნახ.4.6

ადგილმდებარეობის დასადგენად არსებობს სხვადასხვა საშუალება, რომლითაც ხელმძღვანელობს აპლიკაცია. ასეთი საშუალებები შეიძლება იყოს: “GPS” და “NETWORK” ტელეფონის მომხმარებელს აქვს საშუალება თვითონ არეგულიროს, რა საშუალებით სურს კოორდინატების მიღება.

იმ შემთხვევაში თუ ლოკაციის სერვისები ჩართულია, აპლიკაცია ავტომატურად ეცდება ადგილმდებარეობის განსაზღვრას შემდეგი ალგორითმით:

რადგან GPS საშუალებით მიღებული კოორდინატები უფრო ზუსტია ვიდრე სხვა დანარჩენი საშუალებებით, ამის გამო პრიორიტეტი მინიჭებული ექნება gps-ს და პირველი ბრძანება იქნება მისი საშუალებით კოორდინატების მოთხოვნა;

იმ შემთხვევაში, თუ ვერ მოხერხდა ადგილმდებარეობის განსაზღვრა, რაღაც დროის შემდეგ აპლიკაცია ეცდება მოიხმაროს სხვა საშუალებები. მაგალითად „NETWORK“, რომელიც დააბრუნებს იმ ანძის ადგილმდებარეობას რომელზეც არის იმ მომენტისთვის მობილური მოწყობილობა მიერთებული;

კოორდინატების მოპოვების შემდეგ რუკაზე მოხდება ავტომატური მიახლოება ნაპოვნ ადგილმდებარეობასთან.

ზარის განხორციელება ღილაკზე დაჭერით აპლიკაცია გადავა ახალ გვერდზე სადაც იქნება სამი ეტაპი ინფორმაციის დამუშავებისთვის:

- ლოკაციის დადგენა;

ლოკაციის გადაგზავნა 112-ში;

ზარის განხორციელება 112-ში.

თითოეულ ეტაპს წინასწარ ექნება განსაზღვრული ის მაქსიმალური დრო, რაც შეიძლება დასჭირდეს ოპერაციის დასრულებისთვის. მაგალითად, პირველ ეტაპზე, როდესაც ხდება კოორდინატების აღება, გამოყოფილი ექნება 15 წამი, რომლის გასვლის შემდეგ გადავა მომდევნო ეტაპზე.

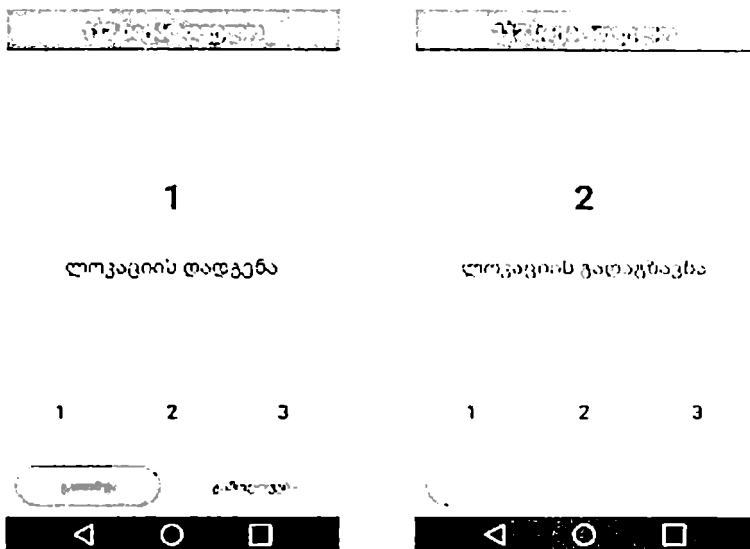
იმ შემთხვევაში თუ აპლიკაციას არ დასჭირდება ზემოთ ხსენებული დრო და წინასწარ მოახერხებს ადგილმდებარეობის განსაზღვრას, აპლიკაცია აღარ დაელოდება 15 წამის გასვლას და ავტომატურად გადავა მომდევნო ეტაპზე.

როგორც წინა პარაგრაფში აღვნიშნეთ აპლიკაცია მუდმივ რეჟიმში ცდილობს პერიოდულობით ახალი კოორდინატების მიღებას, მაგრამ ზარის განხორციელების ეტაპზე საჭიროა მაინც ახალი კოორდინატების განსაზღვრა მონაცემთა უტყუარობის მიზნით (ნახ.4.7).

მოცემულ სურათზე ჩანს, რომ აპლიკაცია პირველ ეტაპზე ცდილობს ადგილმდებარეობის დადგენას, ხოლო წარმატების შემდეგ გადადის ავტომატურად მეორე ეტაპზე.

მეორე ეტაპს, ისევე როგორც პირველს, ექნება წინასწარ განსაზღვრული მაქსიმალური დრო, როდესაც ეცდება თავისი ფუნქციის შესრულებისათვის.

იმ შემთხვევაში, თუ დასრულდება შესასრულებელი მოქმედება იმაზე ადრე, ვიდრე მაქსიმალური დრო იყო გამოყოფილი, მაშინ აპლიკაცია ავტომატურად გადავა მესამე ეტაპზე.



ნახ.4.7

მეორე ეტაპის ფუნქციაა ინფორმაციის გადაგზავნა 112-ის ქოლ-ცენტრში. ინფორმაციის მიმოცვლა მობილურ აპლიკაციასა და 112-ს შორის მოხდება http პროტოკოლით, WEB API ტექნოლოგიით დამზადებული ვებ-სერვისის საშუალებით. თუმცა, იმ შემთხვევაში, თუ აპლიკაციის მომხმარებელს არ აქვს წვდომა ინტერნეტთან, სხვადასხვა მიზეზების გამო, არ აქვს ჩართული Wi-Fi, მობილური ინტერნეტი, არ არის დაფარვის ზონაში ან თუნდაც ანგარიშზე არ აქვს თანხა, რაც საჭიროა ინტერნეტთან კავშირისთვის, მაინც შესაძლებელია 112-მდე შეტყობინების მიღწევა GSM SMS მოკლე ტექსტური შეტყობინების გაგზავნით, რომელიც, როგორც ზემოთ ვახსენეთ, უფასოა.

მას შემდეგ, რაც წარმატებით დასრულდება მეორე ეტაპი, აპლიკაცია გადავა მესამე ეტაპზე - „112-ში დარეკვა“ გადაგზავნილი ინფორმაცია 112-ის ოპერატორს გამოუჩნდება და აღმოუჩენს დახმარებას ინიციატორს. ადგილდებარეობის ცოდნა მომსახურების დააჩქარებას შეუწყობს ხელს.

ზარის განხორციელების რეჟიმში იქნება კიდევ ორი დილაკი როგორც 4.8 ნახაზზეა ნაჩვენები. „გათიშვა“ და „გამოტოვება“ გათიშვა-დილაკით მომხმარებელი აუქმებს ზარის განხორციელებას, ხოლო გამოტოვება-დილაკით კი შეძლებს კოორდინატების მოპოვება ან მათი გადაგზავნა გამოტოვოს და პირდაპირ დარეკვაზე გადავიდეს.



ნახ.4.8

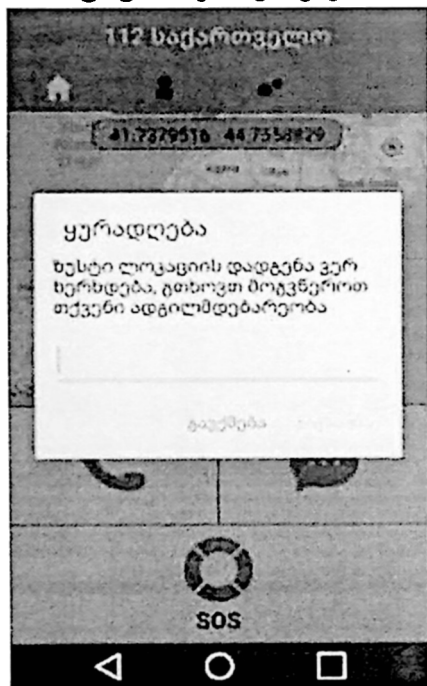
ჩუმი განგაშის დროს გვექნება მსგავსი გვერდი, როგორიც გვაქვს ზარის განხორციელებისას, სადაც ხდება მხოლოდ ორი ეტაპი:

- ლოკაციის დადგენა;
- ლოკაციის გადაგზავნა „112“-ში

მესამე „დარეკვა 112-ში“ ეტაპი არ გვაქვს და სწორედ ამიტომაც ჰქვია ამას „ჩუმი განგაში“, ამ დროს 112-დან არ მოხდება უკუკავშირი და დასახმარებლად გამოიგზავნება ბრიგადა გაგზავნილი კოორდინატების მიხედვით.

ეს ის შემთხვევაა, როდესაც ინიციატორს არ შეუძლია ლაპარაკი, მაგალითად მარცვისას ან სხვა ნებისმიერ შემთხვევაში (მაგალითად, ტერორიზმი). ადგილმდებარეობის დადგენასთან ერთად ავტომატურად გვაქვს ინფორმაცია მისი სიზუსტის შესახებაც გაზომილი მეტრებში. იმ შემთხვევაში,

თუ სიზუსტე წინასწარ განსაზღვრულზე მეტია, მაშინ საჭირო ხდება დამატებითი ინფორმაციის მიღება მომხმარებლისგან ადგილმდებარეობის შესახებ, მაგრამ სავალდებულო არ იქნება. ჩანართს ექნება შემდეგი სახე (ნახ.4.9).



ნახ.9

იმ შემთხვევაში თუ მომხმარებელი შეიყვანს დამატებით ინფორმაციას, რა თქმა უნდა, კარგი იქნება, რაც გააადვილებს მასთან მისვლას. ინფორმაციის გაგზავნისას აპლიკაცია შეამოწმებს ინტერნეტთან კავშირს, თუ კავშირი არსებობს, მაშინ პრიორიტეტი მიენიჭება მას და გაიგზავნება, წინააღმდეგ შემთხვევაში შეტყობინება გაიგზავნება SMS-ის სახით.

მთავარ გვერდზე არსებული ფუნქციონალიდან დარჩა ჩატი ანუ ოპერატორთან ონლაინ მიმოწერა.

ჩატის ღილაკზე დაჭერით აპლიკაცია ეცდება შეამოწმოს ინტერნეტთან კავშირი და ასევე ლოკაციის სერვისები, არის თუ არა გააქტიურებული.

იმ შემთხვევაში თუ ინტერნეტთან კავშირი არ არსებობს, აპლიკაცია მომხმარებელს შეატყობინებს ამის შესახებ.

თუ ლოკაციის სერვისები ჩართული არ არის, ამ დროს აპლიკაცია შეახსენებს და საჭიროების შემთხვევაში მოხდება მათ ჩასართავ პანელში გადამისამართება.

კოორდინატების ჩართვა აუცილებელი არ არის ჩატის ფანჯარაში გადასასვლელად.

ინტერნეტის არსებობის შემთხვევაში ავტომატურად მოხდება 112-ის სერვერებიდან მანამდე არსებული მიმოწერის ჩამოტვირთვა და გამოუჩნდება მომხმარებელს (ნახ.4.10).

მოცემულ გვერდზე იქნება შეტყობინების ჩასაწერი ველი და გაგზავნის ღილაკი. იმ შემთხვევაში თუ მომხმარებელს ჩართული ექნება ლოკაციის სერვისები, აპლიკაცია პირველ შეტყობინებასთან ერთად ავტომატურ რეჟიმში გაგზავნის მათ „112“-ში.



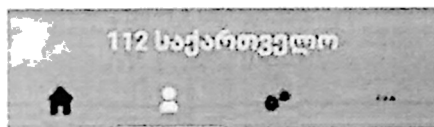
ნახ.4.10

ყველა შეტყობინება, რომელიც განხორციელდება „112“-ის მიმართულებით, ექნება სტატუსი იმის შესახებ მივიდა თუ არა დანიშნულების ადგილამდე. ეს სტატუსები აისახება

თითოეული შეტყობინების ქვემოთ. იმ შემთხვევაში, თუ ვერ მიაღწია შეტყობინებამ „112“-მდე, მაშინ მომხმარებელს ექნება საშუალება ხელმეორედ გაგზავნოს, დააკოპიროს ან სულაც GSM SMS-ის სახით გაგზავნოს.

განვიხილოთ შემდეგი ჩანართი – „მომხმარებლის პროფილი“ (ნახ.4.11). ეს ჩანართი გამიზნულია იმისათვის, რომ უზრუნველყოფილ იყოს მომხმარებლის შესახებ მინიმალური ინფორმაცია, რაც შეიძლება გამოდგეს შემდგომში მის დასახმარებლად.

ფორმის შევსება არაა აუცილებელი, ის სასურველია ოპერატიულობის მიზნით, მომხმარებლის კეთილდღეობისთვის.



ბაზრობითი

ბავშვობითი

ნახ.4.11



როგორც ნახაზზეა ნაჩვენები, შესავსები იქნება მხოლოდ ის საჭირო ინფორმაცია, რომელიც შეიძლება გამოადგეს სასწრაფოს, პატრულს ან სხვა დამხმარე ბრიგადას. შესავსები ინფორმაცია იქნება შემდეგნაირი:

- სახელი;
- გვარი;
- დაბადების თარიღი;
- სქესი;
- სხვა საკონტაქტო პირის ნომერი;
- სამედიცინო ინფორმაცია.

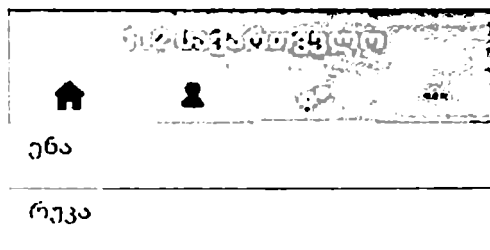
სხვა საკონტაქტო პირის ნომერი ძალზე საჭირო ინფორმაციაა. ასევე სამედიცინო ინფორმაციის ველში სასურველია ჩაიწეროს ის ქრონიკული დაავადებები, რომელიც შეიძლება სჭირდეს პაციენტს. ამ შემთხვევაში სასწრაფო დახმარების ბრიგადას გაუადვილდება და წინასწარ მომზადებული იქნება პაციენტთან შესახვედრად.

მას შემდეგ, რაც მომხმარებელი შეავსებს ყველა საჭირო ინფორმაციას, გვერდის ქვედა პანელზე არსებული „შენახვა“ ღილაკის საშუალებით ინფორმაცია შეინახება როგორც ტელეფონის შიგა მეხსიერებაში, ასევე „112“-ის სერვერზე.

შემდეგი ჩანართი არის „პარამეტრები“ საიდანაც შესაძლებელი იქნება აპლიკაციის მუშაობისთვის განსაზღვრული პარამეტრების ცვლილება (ნახ.4.12).

ასეთები შეიძლება იყოს:

- ენა;
- რუქა.



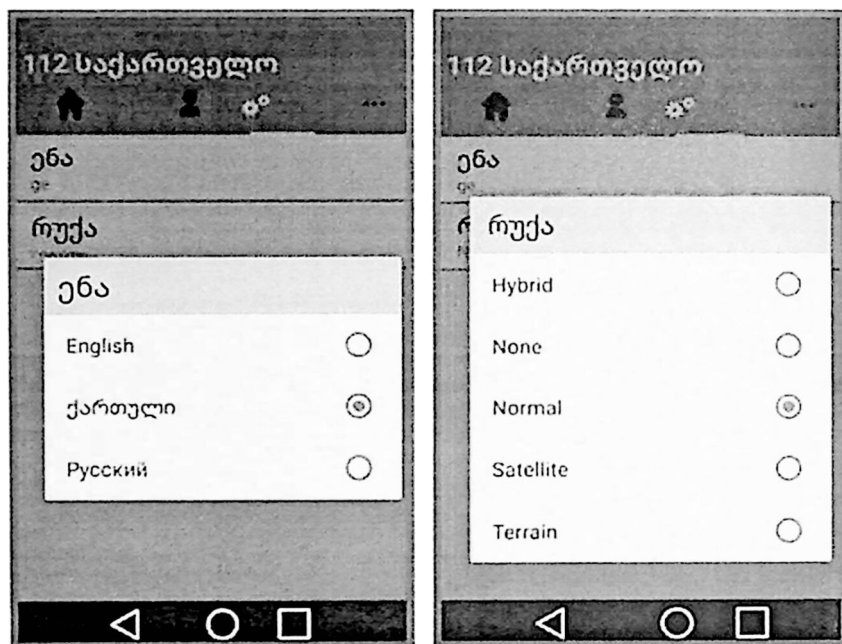
ნახ.4.12

თითოეულ ლილაკზე დაჭერისას გამოჩდება ფანჯარა, სადაც მენიუს სახით ასარჩევი იქნება შესაბამისი პარამეტრები. ლილაკზე „ენა“ დაჭერით გამოჩნდება შემდეგი ველები:

- English;
- ქართული;
- Русский.

დილაკზე „რუქა“ დაჭერით (ნახ.4.13):

- Hybrid
- None
- Normal
- Satellite
- Terrain



ნახ.4.13

ბოლო ჩანართში „სხვა“ იქნება ინფორმაცია აპლიკაციის შესახებ, ასევე ინსტრუქციები, თუ როგორ მუშაობს და გამოვიყენოთ აპლიკაცია.

4.1.3.5. აპლიკაციის უპირატესობანი და ნაკლოვანობები

აპლიკაციის უპირატესობები:

- მრავალფუნქციური
ადვილად ასათვისებელი და მოსახმარებელი
სწრაფი მიზნის მისაღწევად
შეიცავს მოქალაქის შესახებ ინფორმაციას,
რომელიც გამოადგება დამხმარე ბრიგადას
მრავალენოვანი
ადგილმდებარეობის ზუსტი განსაზღვრის
საშუალება

აპლიკაციის ნაკლოვანობები:

- მუშაობს მხოლოდ სმარტფონებზე (ანდროიდ
პლატფორმაზე)

4.2. მუშაობა background არხებში

ანდროიდ ოპერაციულ სისტემას აქვს Service კლასი პროგრამული აპლიკაციის კომპონენტების შესაქმნელად. ასეთი კომპონენტებით იწარმოება ისეთი ოპერაციები, რომლებიც დიდხანს ცოცხლობს და ასრულებს მომხმარებლის ინტერფეისისგან დამალულ დავალებებს [16,43].

ანდროიდი Service-ს უფრო მაღალ პრიორიტეტს ანიჭებს, ვიდრე არააქტიურ Activity-ს. შედეგად სერვისი იქნება განადგურებული ყველაზე ბოლოს, თუ სისტემას რესურსი შემოაკლდება [44,45]. შესაძლებელია ისეთი კონფიგურაცია, რომ თუ სერვისი გაითიშა რესურსის

ნაკლებობის გამო, იგი ისევ ჩაირთოს, როდესაც რესურსი კვლავ გახდება ხელმისაწვდომი. თუ საჭიროა, შეიძლება სერვისის პრიორიტეტი გავუტოლოთ აქტიურ Activity-საც კი. ასეთი შემთხვევა არის მხოლოდ ექსტრემალური სიტუაცია, როდესაც სერვისის გათიშვა აპლიკაციას გამოუსადეგარს ხდის.

სერვისის საშუალებით აპლიკაცია შესაძლოა მაშინაც მუშაობდეს, როდესაც მომხმარებლის ინტერფეისი არაა ხილვადი.

მიუხედავად იმისა, რომ სერვისი მუშაობს მალულად, ისინი მაინც იკავებს აპლიკაციის მთავარ არხს, ისევე როგორც Activity და BroadCast Receivers. იმისათვის, რომ აპლიკაცია არ დასრულდეს ავარიულად, მომდევნო პარაგრაფებში განვიხილავთ, თუ როგორ უნდა ვამუშავოთ დიდი დავალებები განცალკევებულ background არხებში.

4.2.1. სერვისები

Activity-სგან განსხვავებით, რომლებსაც აქვს ვიზუალური მხარდაჭერა, სერვისები მუშაობს მალულად აკეთებს ინტერნეტ-დავალებებს, ავსებს Content Provider-ებს, ამუშავებს დიდ მონაცემებს და ა.შ. Activity-ებისგან განსხვავებით სერვისებს არ სჭირდება მრავალჯერ შექმნა და განადგურება, ისინი შექმნილია იმისათვის, რომ იმუშაოს მუდმივად ისეთი ოპერაციებისთვის, რომელთაც სჭირდებათ დიდი დრო [46].

სერვისის ჩართვა, გათიშვა და მათი კონტროლი შესაძლებელია აპლიკაციის სხვა კომპონენტების მიერ, როგორებიცაა Activity, Broadcast Receivers ან თუნდაც სხვა

სერვისები. თუ აპლიკაცია მოიცავს ისეთ ფუნქციონალს, რომელიც დამოკიდებული არაა მომხმარებლის შესატან მონაცემებზე ან შეიცავს მძიმე ოპერაციებს, მაშინ სერვისების გამოყენება აუცილებლად იქნება საჭირო.

როგორც ზემოთ აღვნიშნეთ, ანდროიდი უფრო მაღალ პრიორიტეტს ანიჭებს Service-ს, ვიდრე არააქტიურ Activity-ს, შედეგად სერვისი იქნება გათიშული ყველაზე ბოლოს, თუ სისტემას რესურსი შემოაკლდება. სერვისის გათიშვის ერთადერთი მიზეზი შეიძლება გახდეს მხოლოდ ის შემთხვევა, როდესაც აქტიურ Activity-ის სჭირდება და სისტემას აღარ აქვს თავისუფალი რესურსი. როდესაც ასეთი სიტუაცია დგება, შეგვიძლია სერვისი ავტომატურ რეჟიმში გავუშვათ რესურსების გათავისუფლებისას.

თუ სერვისი დამოკიდებულია პირდაპირ მომხმარებელზე, მაგალითად, მუსიკის დაკვრა, მაშინ შეგვიძლია გავზარდოთ სერვისის პრიორიტეტი და გავუტოლოთ აქტიურ Activity-ს [47].

4.2.2. სერვისის შექმნა და მათი კონტროლი

მომდევნო სექციებში განვიხილავთ სერვისის შექმნას და მათ გათიშვას `startService()` და `stopService()` მეთოდების გამოყენებით.

სერვისის შესაქმნელად საჭიროა შეიქმნას კლასი, რომელიც იქნება `Service` კლასის შვილი. დაგვჭირდება მშობელი კლასიდან `onCreate()` და `onBind()` მეთოდების აღწერა.

ფრაგმენტი 4.0

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {
    @Override
    public void onCreate() {
        super.onCreate();
        // TODO: Actions to perform when
        service is created.
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO: Replace with service
        binding implementation.
        return null;
    }
}
```

ახლა უკვე როდესაც გვაქვს ახალი სერვისი საჭიროა მისი აპლიკაციის მანიფესტ ფაილში გაწერა. 4.1 ფრაგმენტზე ნაჩვენებია მანიფესტ ფაილში შესაბამისი ბრძანების გაწერა.

ფრაგმენტი 4.1

```
<service android:enabled="true"
android:name=".MvService"/>
```

იმაში დასარწმუნებლად, რომ სერვისის ჩართვის და მისი კონტროლის უფლება ექნება მხოლოდ ჩვენსავე აპლიკაციას, საჭიროა შემდეგი ბრძანება:

```
<service android:enabled="true"
    android:name=".MyService"
    android:permission="com.paad.MY_SERVICE_PERMISSION"/>
```

შედეგად, ნებისმიერი სხვა აპლიკაცია, რომელსაც დასჭირდება ზემოაღწერილი სერვისის გამოყენება, იძულებული იქნება თავის მანიფესტ-ფაილში გაწეროს ნებართვის მოთხოვნა სერვისზე წვდომისთვის.

4.2.3. სერვისის ამუშავება და მისი გადატვირთვის კონტროლი

რაიმე დავალების მისაცემად სერვისისათვის საჭიროა სერვისის კლასში აღიწეროს onStartCommand() მეთოდი. აქვე შეიძლება აღიწეროს სერვისის გადატვირთვის ფუნქციაც.

onStartCommand() მეთოდი აღიბრუნებს მაშინ, როდესაც მოხდება startService() მეთოდის გამოძახება. სერვისები ეშვება აპლიკაციის მთავარ არხზე, ასე რომ ის ფუნქცია რომელიც ჩაიწერება onStartCommand() მეთოდში, შესრულდება აპლიკაციის მთავარ არხზე. იმისათვის, რომ შევძლოთ დამალულ სხვა არხზე მუშაობა, საჭიროა გავაკეთოთ ახალი არხი და მას მივცეთ დავალება onStartCommand() მეთოდში.

4.3 ფრაგმენტზე ნაჩვენებია onStartCommand() მეთოდის აღწერა, რომელიც აბრუნებს საიდენტიფიკაციო მნიშვნელობას, რომლის საშუალებითაც სერვისის აღდგენა იქნება შესაძლებელი გადატვირთვის შემთხვევაში.

```
@Override
public int onStartCommand(Intent intent, int
flags, int startId) {
    startBackgroundTask(intent, startId);
    return Service.START_STICKY;
}
```

ამით საშუალება გვებლევა `onStartCommand()` მეთოდი დასრულდეს სწრაფად და გაკონტროლდეს სერვისის გადატვირთვის ფუნქცია დაბრუნებული მნიშვნელობით, რომელიც შეიძლება იყოს რამდენიმენაირი:

- `START_STICKY` - თუ ჩვენ დავაბრუნებთ ამ მნიშვნელობას, მაშინ ყოველ ჯერზე `onStartCommand()` მეთოდი იქნება გამოძახებული, როდესაც მოხდება სისტემის მიერ სერვისის გადატვირთვა;

- `START_NOT_TRICKY` - გამოიყენება მაშინ, როდესაც სერვისს ევალება სპეციფიკური დავალება. როგორც წესი, ასეთი სერვისები თვითონ ითიშება, როდესაც დავალებას დაასრულებს. ამ რეჟიმში სრულდება ისეთი დავალებები, როგორიცაა განახლებები ან ინტერნეტ-დავალებები. იმის მაგივრად, რომ რესურსის დეფიციტის გამო, ასეთი სერვისი განვაახლოთ, სჯობს დაველოდოთ შემდეგი დაგეგმილი შესრულების დროს;

- `START_REDELIVER_INTENT` - არის მომენტი, როდესაც საჭირო ხდება შემოწმდეს შესრულდა თუ არა დავალება, რომელიც სერვისს დავავალეთ. ეს რეჟიმი აერთიანებს ზემოთ ხსენებულ ორივე რეჟიმს. თუ სერვისი შეჩერებულია

სისტემის მიერ, მაშინ იგი გაეშვება იმ შემთხვევაში თუ მას დარჩენილი აქვს დავალება.

თითოეული რეჟიმის დროს საჭიროა შეჩერდეს სერვისი `stopService()` ან `stopSelf()` მეთოდის საშუალებით, ეს მეთოდები განხილულია მომდევნო პარაგრაფებში.

4.2.4. სერვისის ჩართვა და გათიშვა

სერვისის გასაშვებად საკმარისია გამოვიძახოთ `startService()` მეთოდი. Activity-ს მსგავსად, შესაძლებელია სერვისის ჩარვა მალულად, რეგისტრირებული Intent Receiver-ის საშუალებით. ან შეიძლება მისი ხილულად ჩართვა სერვისის კლასის საშუალებით. იმ შემთხვევაში, თუ აპლიკაციას არ აქვს მოთხოვნილი სერვისის შესაბამისი ნებართვა (მანიფესტ-ფაილში), მაშინ აპლიკაცია გამოიყენებს უსაფრთხოების გამონაკლისის კლასს `SecurityException`-ს [42].

ორივე შემთხვევაში `onStart()` მეთოდს უნდა მივაწოდოთ Intent, რომელსაც ექნება დამატებითი ინფორმაცია. 4.4 ფრაგმენტი ასახავს სერვისის მალულად და ხილულად ჩართვის მაგალითებს.

სერვისის შესაჩერებლად საჭიროა `stopService()` მეთოდის გამოძახება, რომელსაც გადაეცემა შეჩერების შესაბამისი Intent, როგორც ნაჩვენებია 4.5 ფრაგმენტში.

`stopService()` მეთოდის გამოძახება მთლიანად შეაჩერებს სერვისს, მიუხედავად იმისა, თუ რამდენჯერ არის `startService()` მეთოდი გამოძახებული.

ფრაგმენტი 4.4

```
private void explicitStart() {
    // Explicitly start My Service
    Intent intent = new Intent(this,
        MyService.class);
    // TODO Add extras if required.
    startService(intent);
}
private void implicitStart() {
    // Implicitly start a music Service
    Intent intent = new
        Intent(MyMusicService.PLAY_ALBUM);
    intent.putExtra(MyMusicService.ALBUM_NAME_
        EXTRA, "United");
    intent.putExtra(MyMusicService.ARTIST_NAME
        _EXTRA, "Pneonix");
    startService(intent);
}
```

ფრაგმენტი 4.5

```
// Stop a service explicitly.
stopService(new Intent(this, MyService.class));

// Stop a service implicitly.
Intent intent = new
    Intent(MyMusicService.PLAY_ALBUM);
stopService(intent);
```

4.2.5. თვითშეჩერებადი სერვისები

სერვისების მაღალი პრიორიტეტის გამო, როგორც წესი, სისტემა მათ არ ეხემა. ე.წ. თვითშეჩერებადი სერვისები ზრუნავს ჩვენი აპლიკაციისთვის გამოყოფილი მეხსიერების რესურსზე.

სერვისის შეჩერებით სისტემას ეძლევა საშუალება აღიდგინოს რესურსი, წინააღმდეგ შემთხვევაში იგი იძულებული იქნება გამოუყოს რესურსი სერვისს, რომელმაც დაასრულა მუშაობა.

როდესაც სერვისი დაასრულებს მუშაობას, აუცილებელია მის გასათიშად გამოვიძახოთ მეთოდი `stopSelf()`. სერვისის დაუყოვნებლივ შესაჩერებლად `stopSelf()` მეთოდისთვის საჭირო არაა რაიმე პარამეტრის გადაცემა. ან შეიძლება გადაეცეს `startId` პარამეტრი, რათა დავრწმუნდეთ, რომ სერვისმა დაასრულა მუშაობა.

4.2.6. სერვისის და Activity-ს დაკავშირება

შესაძლებელია სერვისისა და Activity-ის ერთმანეთთან დაკავშირება. შედეგად შევძლებთ სერვისის მიმთითებლის შენახვას Activity კლასში და შემდგომში მის სამართავად გამოყენებას. მათ დასაკავშირებლად საჭიროა შევასრულოთ კოდის შემდეგი ფრაგმენტი

ფრაგმენტი 4.6

```
@Override
public IBinder onBind(Intent intent) {
    return binder;
}

public class MyBinder extends Binder {
    MyMusicService getService() {
        return MyMusicService.this;
    }
}

private final IBinder binder = new MyBinder();
```

კავშირი სერვისსა და აპლიკაციის სხვა კომპონენტს შორის წარმოდგენილია როგორც ServiceConnection ობიექტი.

სერვისის დასაკავშირებლად აპლიკაციის სხვა კომპონენტთან საჭიროა ახალი ServiceConnection კლასის შვილი ობიექტის შექმნა, რომელსაც გადავუტვირთავთ მეთოდებს onServiceConnected() და onServiceDisconnected(), რათა მივიღოთ სერვისის მიმთითებელი, მას შემდეგ, რაც უკვე დამყარდება კავშირი, როგორც აქაა ნაჩვენებია:

ფრაგმენტი 4.7

```
// Reference to the service
private MyMusicService serviceRef;
// Handles the connection between the service
and activity
private ServiceConnection mConnection = new
ServiceConnection() {
public void onServiceConnected(ComponentName
className,
IBinder service) {
// Called when the connection is made.
serviceRef =
((MyMusicService.MyBinder) service).getService();
}
public void onServiceDisconnected(ComponentName
className) {
// Received when the service unexpectedly
disconnects.
serviceRef = null;
}
};
```

დასაკავშირებლად საჭიროა გამოვიძახოთ bindService() მეთოდი ჩვენს Activity კლასში. რომელსაც პარამეტრად გადაეცემა Intent, რომელსაც აქვს ინფორმაცია სერვისის შესახებ. ფრაგმენტი 4.8 წარმოგვიდგენს bindService() ნიმუშს:

```
// Bind to the service
Intent bindIntent = new Intent(MyActivity.this,
MyMusicService.class);
bindService(bindIntent, mConnection,
Context.BIND_AUTO_CREATE);
```

ანდროიდ 4.0 წარმოგვიდგენს flag-ების ჩამონათვალს, რომელიც გამოიყენება სერვისისა და აპლიკაციის გადაბმისას:

- **BIND_ADJUST_WITH_ACTIVITY** - აყენებს სერვისის პრიორიტეტს ისეთივეს, როგორიც აქვს იმ activity-ს, რომელთანაც არის დაკავშირებული. შედეგად სისტემა გაზრდის სერვისის პრიორიტეტს მაშინ, როდესაც activity არის აქტიურ რეჟიმში;

- **BIND_ABOVE_CLIENT** და **BIND_IMPORTANT** - ამ შემთხვევაში სერვისის პრიორიტეტი არის დაბალი ვიდრე მასთან დაკავშირებული კლიენტი. ასე, რომ სისტემა მას გათიშავს რესურსის „გასაჭირში“;

- **BIND_NOT_FOREGROUND** - ამ შემთხვევაში სერვისის პრიორიტეტი ვერსასდროს ვერ მივა ახლოს მასთან დაკავშირებული activity-ს პრიორიტეტთან;

- **BIND_WAIVE_PRIORITY** ამ შემთხვევაში სერვისის პრიორიტეტს არ ვეხებით.

როდესაც სერვისი დაკავშირდება, მას შემდეგ მისი ყველა public მეთოდი გახდება ხელმისაწვდომი serviceBinder ობიექტის საშუალებით, რომლის მიღება შესაძლებელია onServiceConnected() მეთოდში.

როგორც წესი, ანდროიდ აპლიკაციები არ ანაწილებს მათთვის გამოყოფილ მეხსიერებას, თუმცა ხანდახან საჭირო ხდება სხვის მიერ შექმნილი სერვისების გამოყენება. მათთან დასაკავშირებლად შეიძლება გამოვიყენოთ Broadcast Intent ან გამოვიყენოთ extras Bundle სერვისის ჩასართავად.

4.2.7. Foreground სერვისების შექმნა

ანდროიდი იყენებს დინამიკურ მიდგომებს რესურსების გასანაწილებლად, რამაც შეიძლება გამოიწვიოს აპლიკაციის კომპონენტის შეჩერება, გაფრთხილების ან მის გარეშეც კი.

როდესაც ხდება გამოთვლები, თუ აპლიკაციის რომელი კომპონენტი უნდა ამოიშალოს, ანდროიდის ოპერაციული სისტემა სერვისებს მიიჩნევს რიგით მეორე პრიორიტეტის მქონდე კომპონენტებად. მხოლოდ აქტიური activity კლასები ითვლება ყველაზე მაღალი პრიორიტეტის მქონედ.

თუ სერვისსა და აპლიკაციის მომხმარებელს აქვს ერთმანეთთან უშუალო კავშირი, მხოლოდ მაშინ შეგვიძლია ავწიოთ სერვისის პრიორიტეტი და გავუტოლოთ აქტიური activity-სას. ამის გაკეთება შეგვიძლია startForegroundService() მეთოდის გამოძახებით.

იმის გამო, რომ Foreground სერვისები ითვლება, რომ ისინი მუშაობს მომხმარებელთან პირდაპირ, startForegroundService() მეთოდში, უნდა აღვწეროთ ongoing ნოტიფიკაცია, როგორც ეს ნაჩვენებია 4.9 ფრაგმენტზე. ეს ნოტიფიკაცია გამოჩნდება და იქნება ნოტიფიკაციების ადგილზე მანამ, სანამ სერვისი იქნება აქტიური.

კარგი იქნება თუ მომხმარებელს ექნება სერვისის შეჩერების საშუალება. როგორც წესი, ასეთი საშუალება პირდაპირ ნოტიფიკაციების პანელშივე დევს ხოლმე.

ფრაგმენტი 4.9

```
private void startPlayback(String album,
                          String artist) {
    int NOTIFICATION_ID = 1;
    //Create an Intent that will open main Activity
    // if the notification is clicked.
    Intent intent = new Intent(this, MyActivity
                              .class);
    PendingIntent pi = PendingIntent
        .getActivity(this, 1, intent, 0);
    // Set the Notification UI parameters
    Notification notification = new
        Notification(R.drawable.icon,
        "Starting Playback",
        System.currentTimeMillis());
    notification.setLatestEventInfo(this, album,
        artist, pi);
    // Set the Notification as ongoing
    notification.flags = notification.flags |
    Notification.FLAG_ONGOING_EVENT;
    // Move the Service to the Foreground
    startForeground(NOTIFICATION_ID, notification);}
```

თუ სერვისს აღარ სჭირდება მაღალი პრიორიტეტი, შეიძლება იგი გადავიტანოთ როგორც background სერვისი და ავტომატურად გავაქროთ ნოტიფიკაციების პანელიდან stopForeground() მეთოდით (ფრაგმენტი 4.10).

ფრაგმენტი 4.10

4.2.8. Background Thread-ების გამოყენება

```
public void pausePlayback() {
    // Move to the background and remove the
    Notification
    stopForeground(true); }
```

კარგი ანდროიდის აპლიკაციის ერთ-ერთი შეფასების კრიტიკული კრიტერიუმია მისი რეაგირება მომხმარებლის ბრძანებებზე. იმაში რომ დავრწმუნდეთ, რომ ჩვენი აპლიკაცია არის რეზონანსული მომხმარებლის მოქმედებებზე და სისტემის მოვლენებზე, სასიცოცხლოდ მნიშვნელოვანია ყველა მძიმე ოპერაცია, მათ შორის Input/Output ოპერაციები, გადავიტანოთ აპლიკაციის მთავარ Thread-იდან რაიმე შვილობილ Thread-ზე.

ანდროიდში ნებისიერი activity, რომელიც არ რეაგირებს მაგალითად, ღილაკზე დაჭერიდან მაქსიმუმ 5 წამის განმავლობაში და Broadcast Receiver-ები, რომლებიც არ რეაგირებენ მაქსიმუმ 10 წამის მანძილზე, ისინი ითვლება როგორც არარეაგირებადი.

აპლიკაციის მუშაობისას შეყოვნებები ძალიან ცუდია, მომხმარებელი მათ მაშინაც კი ამჩნევს როდესაც ისინი არის რამდენიმე ასეული მილიწამის სიგრძის.

მნიშვნელოვანია, რომ ყველა არაპრიმიტიული ქმედება გადავიტანოთ background არხზე, თუ ისინი პირდაპირ კავშირში არაა მომხმარებლის ინტერფეისთან. განსაკუთრებით მნიშვნელოვანია ფაილური ოპერაციების, ინტერნეტ მიების, მონაცემთა ბაზის ტრანზაქციების და კომპლექსური გამოთვლების წარმოება background არხზე.

ანდროიდს აქვს ალტერნატიული საშუალებები დავალებათა გადასატანად background არხზე. შეიძლება გაკეთდეს საკუთარი Thread-ის იმპლემენტაცია და გამოყენებულმ იქნას სინქრონზაციის მეთოდები, რათა მოხდეს მომხმარებლის ინტერფეისის განახლება. ერთ-ერთი ალტერნატივაა AsyncTask, რომელიც საშუალებას იძლევა შესრულდეს

ოპერაციები background არხზე, დაკვირვება პროცესის პროგრესზე და განახლდეს მომხმარებლის ინტერფეისი.

4.2.9. AsyncTask ასინქრონული დავალებების შესასრულებლად

AsyncTask არის ერთ-ერთი საუკეთესო საშუალება აღწეროთ ასინქრონული დავალების მართვის პროცესი, ანუ ვამუშაოთ background არხზე მძიმე ოპერაციები და ვაცნობოთ მომხმარებლის ინტერფეისს პროგრესის ცვლილებებისა და სამუშაოს დამთავრების შესახებ.

AsyncTask თვითონ აგვარებს ყველა საჭირო მოქმედებას, როგორიცაა: ახალი thread-ის შექმნა, მენეჯმენტი და სინქრონიზაცია. ანუ გვამღევს საშუალებას თვალყური ვადევნოთ პროგრესს და განვაახლოთ სურვილისამებრ მომხმარებლის ინტერფეისი.

AsyncTask არის კარგი გამოსავალი შედარებით პატარა ოპერაციებისათვის, რომლებმაც ოპერაციის დასრულების შესახებ უნდა აცნობოს მომხმარებელს. თუმცა იგი არ იცოცხლებს თუკი ჩვენ შევცვლით activity-ს, რაც ნიშნავს, რომ მაგალითად, თუ გადავატრიალებთ ტელეფონს და ამ დროს activity შეიცვლება, მაშინ asyncTask მუშაობა შეწყდება.

გრძელვადიანი ოპერაციებისთვის, როგორიცაა ინტერნეტიდან რაიმეს გადმოწერა, უკეთესი გამოსავალია სერვისების გამოყენება.

ასევე Cursor კლასი კარგი გამოსავალია Content Provider-თან და მონაცემთა ბაზებთან სამუშაოდ.

4.2.10. AsyncTask-ის შექმნა

თითოეული AsyncTask აღწერისას უნდა გავწეროთ მისი პარამეტრის ტიპები, პროგრესის რეპორტი და უკან დასაბრუნებელი მნიშვნელობები. თუ არ გვჭირდება ან არ გვინდა შემაჯავლი პარამეტრები, რეპორტი ან უკან რაიმეს დაბრუნება, უბრალოდ შეგვიძლია გავწეროთ Void.

ასინქრონული ოპერაციის გასაკეთებლად საჭიროა შევქმნათ კლასი, რომელიც იქნება AsyncTask-ის შვილი. აღწეროთ პარამეტრის ტიპები ისე. როგორც ნაჩვენებია 4.11 ფრაგმენტზე.

ჩვენმა კლასმა უნდა აღწეროს აუცილებლად შემდეგი მეთოდები:

- `doInBackground` – ეს მეთოდი შესრულდება აუცილებლად `background` არხზე. ასე რომ ჩვენი მძიმე ოპერაცია აქ უნდა მოვათავსოთ და არ უნდა ვეცადოთ მომხმარებლის ინტერფეისთან კავშირის დამყარება ამ მეთოდიდან.

შეგვიძლია გამოვიყენოთ `progressUpdate()` მეთოდი რათაპროგრასის შესახებ ვანობოთ `onProgressUpdate()`, ხოლო როდესაც პროცესი დასრულდება უკან დავაბრუნებთ მნიშვნელობას რომლსაც დაიჭერს `onPostExecute()` მეთოდი, რომელსაც შეუძლია მომხმარებლის ინტერფეისის განახლება.

- `onProgressUpdate` – ეს მეთოდის გამოიყენება იმისთვის რომ მომხმარებლის ინტერფეისს ეცნობოს დავალების შესრულებული პროგრესის შესახებ. `progressUpdate()` მეთოდის საშუალებით იგი იღებს ინფორმაციას პროგრესის შესახებ `doInBackground()` მეთოდიდან.


```
private class MyAsyncTask extends
    AsyncTask<String, Integer, String> {
    @Override
    protected String doInBackground(String...
        parameter) {
//Moved to a background thread.
String result = "";
int myProgress = 0;
int inputLength = parameter[0].length();
// Perform background processing task, update
//myProgress]
for (int i = 1; i <= inputLength; i++) {
    myProgress = i;
    result = result +
        parameter[0].charAt(inputLength-i);
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) { }
    publishProgress(myProgress);
}
// Return the value to be passed to
//onPostExecute
return result; }
@Override
protected void onProgressUpdate(Integer...
    progress) {
//Synchronized to UI thread. Update progress //
bar, Notification, or other UI elements
asyncProgress.setProgress(progress[0]); }
@Override
protected void onPostExecute(String result) {
// Synchronized to UI thread.
// Report results via UI update, Dialog, or
//notifications
asyncTextView.setText(result); }
}
```

ეს მეთოდი სინქრონიზებულია მომხმარებლის ინტერფეისთან, რაც საშუალებას გვაძლევს უსაფრთხოდ შევცვალოთ მომხმარებლის ინტერფეისის ელემენტები.

- `onPostExecute` – როდესაც `doInBackground()` მეთოდი დაასრულებს მუშაობას, მის მიერ დაბრუნებული მნიშვნელობა აისახება ამ მეთოდში.

შეიძლება მომხმარებლის ინტერფეისის ელემენტების უსაფრთხო განახლება ოპერაციის დასრულების შემდეგ.

4.2.11. ასინქრონული ოპერაციების გაშვება

`AsyncTask`-ის იმპლემენტაციის შემდეგ შეგვიძლია შევექმნათ კლასის ახალი ობიექტი და გამოვიძახოთ მეთოდი `execute()`, როგორც ეს ნაჩვენებია 4.12 ფრაგმენტზე. შესაძლებელია რამდენიმე იმ ტიპის პარამეტრის გადაცემა, რაც იმპლემენტაციისას განვსაზღვრეთ.

ფრაგმენტი 4.12

```
String input = "redrum ... redrum";
new MyAsyncTask().execute(input);
```

4.2.12. Intent სერვისი

`Intent` სერვისი მოსახერხებელი კლასია, რომელიც ითვლება საუკეთესო პრაქტიკად `background` სერვისის აღწერისას [48]. აპლიკაციის სხვა კომპონენტები იყენებენ `Intent`-ს, რომლებსაც აქვს საჭირო პარამეტრები სერვისის ჩასართველად. `Intent` სერვისი რიგში აყენებს `Intent`-ებს და ასრულებს მათ თანმიმდევრულად ასინქრონულ `background` არხზე. როგორც კი ყველა `Intent` შესრულდება, `Intent` სერვისი ავტომატურად განადგურდება.

Intent სერვისი ავტომატურ რეჟიმში აგვარებს ყველა საჭირო კომპლექსურობას Intent-ების რიგში ჩაყენებიდან, მათ შესრულებამდე, background სერვისის შექმნაში და მომხმარებლის ინტერფეისის სინქრონიზაციაში.

იმისათვის რომ აღვწეროთ სერვისი, როგორც Service Intent, საჭიროა იმპლემენტაცია გავუკეთოთ IntentService კლასს და აღვწეროთ onHandleIntent() მეთოდი. იგი შესრულებულ იქნება worker thread-ზე,

ფრაგმენტი 4.13

```
import android.app.IntentService;
import android.content.Intent;
public class MyIntentService extends
    IntentService {
    public MyIntentService(String name) {
        super(name);
    }
    //TODO Complete any required constructor tasks }
    @Override
    public void onCreate() {
        super.onCreate();
    }
    //TODO: Actions to perform when service is created
    }
    @Override
    protected void onHandleIntent(Intent intent) {
    // This handler occurs on a background thread.
    // TODO The time consuming task should be
    // implemented here.
    // Each Intent supplied to this IntentService
    //will be
    // processed consecutively here. When all
    //incoming Intents
    //have been processed the Service will
    //terminate itself.
    }}
```

4.2.13. Alarm-ების გამოყენება

Alarm ნიშნავს Intent-ის გამოძახებას წინასწარ განსაზღვრულ დროს [49]. Timer-ისგან განსხვავებით, Alarm მუშაობს აპლიკაციის გარეთ, ასე რომ აპლიკაციის დახურვის შემდეგაც კი შეგვიძლია აპლიკაციის მოვლენების დაჭერა და მათი გამოწვევა. ისინი უფრო მძლავრია მაშინ, როდესაც მათ ვიყენებთ Broadcast Receiver-ებთან ერთად.

Alarm-ები ცოცხალია მაშინაც კი, თუ მოწყობილობა „მიძინებულია“ და მათ შეუძლია მისი გამოღვიძება. თუმცა ყველა Alarm ითიშება მოწყობილობის გადატვირთვისას.

Alarm მართვა შესაძლებელია AlarmManager კლასის საშუალებით [50]. მისი მოპოვება ხდება შემდეგნაირად:

ფრაგმენტი 4.14

```
AlarmManager alarmManager =  
(AlarmManager) getSystemService(Context.ALARM_SERVICE);
```

4.2.14. Alarm-ის შექმნა კონფიგურირება და გაუქმება

ერთჯერადი Alarm-ის შესაქმნელად საჭიროა გამოვიძახოთ set() მეთოდი და განვსაზღვროთ Alarm ტიპი, შესრულების დრო და Intent, რომელსაც შეასრულებს როდესაც დადგება ამის დრო. თუ მიუთითებთ წარსულ დროს მაშინ Alarm შესრულდება მყისიერად.

არსებობს Alarm-ების 4 ტიპი:

- RTC_WAKEUP – გამოაღვიძებს მოწყობილობას როცა დადგება შესრულების დრო;

- RTC – შეასრულებს დავალებას და არ გამოადვიძებს მოწყობილობას როდესაც დადგება შესრულების დრო;
- ELAPSED_REALTIME – შეასრულებს დავალებას მოწყობილობის ჩატვირთვიდან გასული დროის მიხედვით, მაგრამ მოწყობილობას არ გააღვიძებს;
- ELAPSED_REALTIME_WAKEUP – შეასრულებს დავალებას მოწყობილობის ჩატვირთიდან გასული დროის მიხედვით და მოწყობილობას გააღვიძებს.

4.15 ფრაგმენტზე ნაჩვენებია Alarm-ის შექმნის პროცესი.

ფრაგმენტი 4.15

```
// Get a reference to the Alarm Manager
AlarmManager alarmManager =
(AlarmManager) getSystemService(Context.ALARM_SERVICE);
// Set the alarm to wake the device if sleeping.
int alarmType =
    AlarmManager.ELAPSED_REALTIME_WAKEUP;
// Trigger the device in 10 seconds.
long timeOrLengthofWait = 10000;
// Create a Pending Intent that will broadcast
//and action
String ALARM_ACTION = "ALARM_ACTION";
Intent intentToFire = new Intent(ALARM_ACTION);
PendingIntent alarmIntent =
    PendingIntent.getBroadcast(this, 0,
        intentToFire, 0);
// Set the alarm
alarmManager.set(alarmType, timeOrLengthofWait,
    alarmIntent);
```

4.3. მეოთხე თავის დასკვნა

Google Maps for Android v2 ხელმისაწვდომია დეველოპერებისათვის, რომლებიც არეგისტრირებენ API Key-ს. რუქების გამოყენება ანდროიდ აპლიკაციებში მას ხდის უფრო ინფორმაციულად მდიდარს და საინტერესოს. დეველოპერები, აპლიკაციის მოთხოვნებიდან გამომდინარე, რუქას ათავსებენ საკუთარ აპლიკაციაში, ან ჩაშენებულ რუქის აპლიკაციას გახსნიან მოწყობილობაში. Android Location Services API, Google Location Services API და Google Maps Android v2 services API გვთავაზობს მდიდარ შესაძლებლობას, რათა განვაკითვროთ ჩვენი ანდროიდ აპლიკაციები.

ლიტერატურა:

1. კაკაშვილი გ. მობილური ოპერაციული სისტემების გამოწვევები და შესაძლებლობები. ჟურნ. „ინტელექტი“. ISSN 1512-0333. 1(66). საქ. მეცნ. და საზოგ. განვით. ფონდი. თბ., 2020. გვ. 34-37
2. Mobile Operating System Market Share Worldwide. Internet resource: <https://gs.statcounter.com/os-market-share/mobile/worldwide>. 24.11.20
3. Android vs iOS: Which mobile OS is right for you? Internet resource: <https://www.itpro.co.uk/mobile/30409/android-vs-ios-which-mobile-os-is-right-for-you>. 24.11.2020
4. Developed by Google Developer Training Team. Android Developer Fundamentals Course – Practicals. Dec.2016. 56 p. <https://google-developer-training.github.io/android-developer-fundamentals-course-practicals/en/android-developer-fundamentals-course-practicals-en.pdf>
5. Android version history. Internet resource: https://en.wikipedia.org/wiki/Android_version_history. 2.11.20
6. Android 10. Internet resource: https://en.wikipedia.org/wiki/Android_10. 2.11.20
7. Key Android Concepts. Internet resource: <https://www.macadamian.com/learn/7-key-android-concepts/>
8. Android 10. Internet resource: <https://www.android.com/android-10/> 10.03.2020
9. Mobile App Development Process: A Step-by-Step Guide. By Team Invonto. 01.2020.

10. ჩოგოვაძე გ., სურგულაძე გ., გულიტაშვილი მ., დოლიძე ს. პროგრამული აპლიკაციების ხარისხის მართვა: ტესტირება და ოპტიმიზაცია. ISBN 978-9941-20-629-2. სტუ. „IT-კონსალტინგ ცენტრი“. თბ., 2020. -363 გვ.

11. <https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html>

12. DiMarzio J.F. Android a Programmers Guide. McGraw Hill Education; 1st edition. New York 2008

13. Install Android Studio. Internet resource: <https://developer.android.com/studio/install>

14. Cardle P. Android App Development in Android Studio - Java plus Android edition for beginners. 2017. 202 p. Internet resource: <https://www.pdfdrive.com/android-app-development-in-android-studio-java-plus-android-edition-for-beginners-e186456208.html>

15. Java vs. Kotlin: which is the better option for Android App Development ? <https://clearbridgemobile.com/java-vs-kotlin-which-is-the-better-option-for-android-app-development/>

16. Meier R. Professional Android™ 4 Application Development. ISBN: 978-1-118-10227-5. Copyright © 2012 by John Wiley & Sons, Inc., Indianapolis, Indiana. 868 p.

17. Telephony.Sms.Intents. Internet resource: <https://developer.android.com/reference/android/provider/Telephony.Sms.Intents.html>

18. Pohjolainen J. Android Telephony Manager and SMS.

19. Android. Telephony Manager. Internet resource: <https://developer.android.com/reference/android/telephony/TelephonyManager.html>

20. Android. Connectivity Manager. Internet resource: <https://developer.android.com/reference/android/net/ConnectivityManager.html>

21. Sms Manager. Internet resource: <https://developer.android.com/reference/android/telephony/SmsManager.html>

22. Run apps on the Android Emulator. Internet resource: <https://developer.android.com/studio/run/emulator>

23. Set GPS Location in Emulator using Android Studio. Internet resource: <https://stackoverflow.com/questions/18972114/set-gps-location-in-emulator-using-android-studio>

24. How to emulate GPS location in the Android Emulator? Internet resource: <https://stackoverflow.com/questions/2279647/how-to-emulate-gps-location-in-the-android-emulator/2279827>

25. Johnson D. How to update or fix Google Play Services on your Android, and keep all your apps running correctly. 2020, Internet resource: <https://www.businessinsider.com/how-to-update-google-play-services-android-troubleshooting>

26. Vogel L. Android Location API with the fused location provider - Tutorial. Vogella GmbH. Version 4.2, 2016

27. Android Google Services: "Location APIs". <http://d.android.com/google/play-services/location.html>

28. Android Training: "Making Your App Location-Aware". <http://d.android.com/training/location/index.html>

29. Android API Guides: "Location and Maps". <http://d.android.com/guide/topics/location/index.html>

30. Android API Guides - Location Strategies. 2016. Internet resource: https://topic.alibabacloud.com/a/android-api-guides-font-coloredlocationfont-strategies_1_21_32525242.html

31. Android Google Services: "Google Maps Android API v2". <http://d.android.com/google/play-services/maps.html>

32. Google Developers: "Google Maps Android API v2". <http://developers.google.com/maps/documentation/android>

33. Android Google Services Reference documentation on the com.google.android.gms.maps package. 2019. Internet resource: <https://developers.google.com/android/reference/com/google/android/gms/maps/package-summary>

34. Maps SDK for Android: Get an API Key. <https://developers.google.com/maps/premium/apikey/maps-android-apikey>

35. Maps SDK for iOS: Get an API Key. <https://developers.google.com/maps/premium/apikey/maps-ios-apikey>

36. European emergency number 112. Internet resource: <https://eena.org/about-112/whats-112-all-about/>

37. „112“ (emergency telephone number). [https://en.wikipedia.org/wiki/112_\(emergency_telephone_number\)](https://en.wikipedia.org/wiki/112_(emergency_telephone_number))

38. Guidelines to select Emergency Number for public telecommunications network. Temporary document. Geneva, 6-15 May 2008

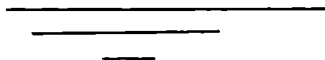
39. Authorities warn against dialing 112 for emergencies. <http://archive.jsonline.com/news/wisconsin/authorities-warn-against-dialing-112-for-emergencies-0h8vucj-194260201.html>

40. professional_android_4_application_development.
https://www.amazon.com/gp/product/1118102274/ref=oh_aui_detailpage_o06_s00?ie=UTF8&psc=1
41. Advanced Android Application Development (4th Edition) (Developer's Library). https://www.amazon.com/gp/product/0133892387/ref=oh_aui_detailpage_o06_s00?ie=UTF8&psc=1
42. Android security exception for permission listed in manifest file. Internet resource: <https://stackoverflow.com/questions/37000294/android-security-exception-for-permission-listed-in-manifest-file>. 10.10.20
43. Annuzzi J., Darcey Jr.L. Shane Conder. Advanced Android Application Development (4th Edition) (Developer's Library).
44. Creating Background Services. Internet resource: <https://developer.android.com/training/run-background-service/create-service.html>
45. Running in a Background Services. Internet resource: <https://developer.android.com/training/run-background-service/index.html>
46. Services. Internet resource: <https://developer.android.com/guide/components/services.html>
47. Foreground Services (Android). Internet resource: <https://developer.android.com/guide/components/foreground-services>

48. Create a background service. Android Developers. Internet resource: <https://developer.android.com/training/run-background-service/create-service>

49. Scheduling repeated alarms. Internet resource: <https://developer.android.com/training/scheduling/alarms.html>

50. AlarmManager. Internet resource: <https://developer.android.com/reference/android/app/AlarmManager.html>



გადაეცა წარმოებას 2.11.2020, ხელმოწერილია დასაბეჭდად 12.11.2020. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 11. ტირაჟი 100 ეგზ.



სტუ-ს „IT კონსალტინგის ცენტრი“
(თბილისი, მ.კოსტავას 77)