**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Karlo Mikec**

**Implementacija koncepata interakcije s korisnikom u Swift-u**

**SEMINARSKI RAD**

**Varaždin, 2023.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Karlo Mikec**

**Matični broj: 0016151582**

**Studij: Informacijski i poslovni sustavi**

**Implementacija koncepata interakcije s korisnikom u Swiftu**

**SEMINARSKI RAD**

**Mentor:**

**Izv. prof. dr. sc. Zlatko Stapić**

**Varaždin, listopad 2023.**

**Sadržaj**

[1 Uvod 1](#_Toc149473266)

[2 Postupak kreiranja novog projekta 3](#_Toc149473267)

[3 Toast obavijest 6](#_Toc149473268)

[3.1 Implementacija Toast obavijesti 7](#_Toc149473269)

[4 Snackbar obavijest 10](#_Toc149473270)

[4.1 Implementacija Snack bar obavijesti 11](#_Toc149473271)

[5 Floating action button 13](#_Toc149473272)

[5.1 Implementacija Float Action Button-a 14](#_Toc149473273)

[6 Sistemske obavijesti 17](#_Toc149473274)

[6.1 Implementacija jednostavne sistemske obavijesti 18](#_Toc149473275)

[6.2 Implementacija složene sistemske obavijesti 21](#_Toc149473276)

[7 Korisnički dijalozi 25](#_Toc149473277)

[7.1 Implementacija korisničkog dijaloga 25](#_Toc149473278)

[8 Zaključak 29](#_Toc149473279)

[9 Popis svih slika 30](#_Toc149473280)

[10 Popis literature 31](#_Toc149473281)

# Uvod

Obavijesti su kratke poruke koje se javljaju na zaslonu pametnih uređaja kako bi obavijestili korisnike o nadolazećim događajima, informacijama ili aktivnostima. Obavijesti mogu uključivati poruke, e-poštu, upozorenja aplikacija, podsjetnike i slične informacije.

Obavijesti imaju važnu ulogu u svakodnevnom životu korisnika jer omogućuju korisnicima da budu na vrijeme informirani. Međutim prekomjerno slanje obavijesti može rezultirati ometanjem i smanjenom produktivnosti korisnika. Stoga, vrlo je važno da obavijesti budu relevantne i prilagođene potrebama korisnika.

Tvrtka Apple vodi strogu politiku slanja obavijesti kako bi se zaštitilo samo korisničko iskustvo, gdje traže od korisnika dozvolu prije slanja samih obavijesti. Sustav iOS omogućuje korisnicima kontrolu nad aplikacijama koje mogu slati obavijesti i kako će se obavijesti prikazivati. [[2]](#_Popis_literature)

Najnovije obavijesti mogu se pregledati na zaključanom zaslonu samog uređaja pritiskom na obavijest kako bi otvorili aplikaciju iz koje dolazi obavijest. Ako dođe više obavijesti iste aplikacije, one se spremaju u grupu obavijesti i način pregledavanja je isti kao i kod pojedinačnih obavijesti. [[1]](#_Popis_literature)

Obavijestima je također moguće pristupiti putem centra za obavijesti kojeg koriste iOS uređaji. Centar za obavijesti prikazuje povijest obavijesti i omogućuje korisnicima povratak kako bi vidjeli obavijest ako su je propustili. Centru za obavijesti moguće je pristupiti sa zaključanog zaslona na tako da se sa sredine zaslona prstom povuče prema vrhu zaslona.

Kako bi upravljali obavijestima na zaključanom zaslonu potrebno je prstom prijeđi ulijevo preko obavijesti i zatim odabrati „Opcije“ gdje se nudi nekoliko opcija od kojih su: [[1]](#_Popis_literature)

* Isključi zvuk na 1 sat
* Isključi zvuk za danas
* Prikaži postavke
* Isključi

Moguće je personalizirati i način prikazivanja obavijesti. Za to je potrebno otvoriti „Postavke“ pod stavkom „Obavijesti“. U odjeljku „Prikaži kao“ moguće je birati između tri načina prikazivanja obavijesti na zaključanom zaslonu: [[1]](#_Popis_literature)

* Broj: na dnu zaslona prikazuje ukupni broj obavijesti na dnu. Kako bi vidjeli obavijesti potrebno je pritisnuti na broj.
* Naslagano: na dnu zaslona prikazuje obavijesti naslagane jedna na drugu, s najnovijom obavijesti na vrhu.
* Popis: prikazuje obavijesti u obliku popisa.

Svi navedeni nazivi i moguće personalizacije dostupne su od verzije iOS 16 na dalje. Također u novijim verzijama iOS-a moguća je izmjena u načinu personalizacije i samim nazivima.

U svrhu ovog seminarskog rada bit će prikazano kako implementirati sljedeće načine obavijesti: Toast, Snack bar, FAB kontrole i Sistemske obavijesti.

A screen shot of a phone

Description automatically generated A screen shot of a phone

Description automatically generated

Slika 1: Otvaranje sa zaključanog zaslona.[[1]](#_Popis_literature) Slika 2: Upravljanje obavijestima [[1]](#_Popis_literature)

A screenshot of a phone

Description automatically generated

Slika 3: Promjena načina prikaza obavijesti [[1]](#_Popis_literature)

# Postupak kreiranja novog projekta

Kako bi mogli započeti sa implementacijom programskog rješenja, potrebno je pokrenuti Xcode IDE. Zatim odabrati opciju „*Create New Project*“ koja vodi do prozora gdje je potrebno izabrati template za novi projekt. Kako je tema ovog rada „*Implementacija koncepta interakcije s korisnikom u Swiftu*“ izabiremo platformu „*iOS*“ i opciju „*App*“. Na sljedećem prozoru pod stavkom „*Product Name*“ potrebno je specificirati naziv projekta. Kao „*Interface*“ biramo SwiftUI, dok pod stavkom „*Language*“ odabiremo Swift. Pritiskom na gumb „Next“ kreiramo projekt. Kada se otvori sam projekt s lijeve strane možemo vidjeti „*Project navigator*“ u kojem su sadržane sve datoteke koje su potrebne projektu. U sredini zaslona nalazi se mjesto za pisanje programskog rješenja. S desne strane nalazi se „*Interactive preview*“ gdje je odmah vidljivo korisničko sučelje prilikom kreiranja aplikacije. [[3]](#_Popis_literature)

A screenshot of a computer

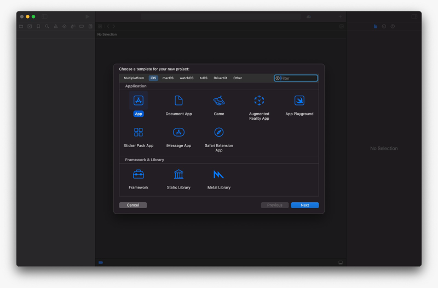
Description automatically generated

Slika 4: Prvi korak kreiranja novog projekta

A screenshot of a computer

Description automatically generated

Slika 5: Drugi korak kreiranja novog projekta



Slika 6: Treći korak kreiranja novog projekta

A screen shot of a cell phone

Description automatically generated

Slika 7: Četvrti korak kreiranja novog projekta

A screen shot of a cell phone

Description automatically generated

Slika 8: Peti korak kreiranja novog projekta

# Toast obavijest

Toast je vrsta obavijesti koja se pojavljuje na zaslonu mobilnog uređaja ili računala kako bi korisnika obavijestila o određenom događaju ili poruci. Ovakav način obavijesti karakterizira kratak sadržaj, kratko vrijeme prikazivanja i dizajnirane su tako da privuku pažnju korisnika pritom ne ometajući ga prilikom aktivnosti koju trenutno obavlja. [[4]](#_Popis_literature)

Toast obavijesti obično se pojavljuju na ekranu samo nekoliko sekundi, nakon čega nestaju. Na ovaj način nastoji se ne ometati korisnikovu trenutnu interakciju s uređajem.

Obavijesti su obično jednostavne, s kratkim tekstualnim porukama, ponekad ikonama ili malim slikama.

Obično se pojavljuju na dnu ekrana ako je riječ o mobilnim uređajima ili u kutu ekrana ako se radi o računalima, kako bi zadržale skromnost i bile manje ometajuće.

Cilj im je informiranje korisnika o važnim događajima, stanjima i porukama. Nastoje skrenuti pažnju korisniku na nešto važno bez ometanja trenutne radnje korisnika. [[5]](#_Popis_literature)

Primjer situacije u kojoj je može prikazati toast obavijest:

* Poruka i obavijest aplikacije – pristigne nam nova e-mail poruka i korisniku se prikaže toast obavijest koja ga o tome obavještava.
* Poziv na akciju – neke obavijesti mogu korisnika pozivati na određenu akciju, poput obavijesti o posljednjoj lokaciji uređaja povezanog na iCloud.

Važno je napomenuti kako se toast obavijesti moraju koristiti odgovorno kako ne bi preplavile korisnika nepotrebnim obavijestima i ometale njegovu produktivnost ili iskustvo. Obavijesti moraju biti relevantne i korisne za korisnika te se ne bi smjele zloupotrebljavati.

## Implementacija Toast obavijesti

Nakon što je kreiran novi projekt kako je objašnjeno u koraku 2, možemo krenuti s izradom Toast obavijesti. No, prije nego što krenemo potrebno je dodati novi paket uz pomoć kojeg ćemo kreirati toast obavijest. Potrebno je slijediti sljedeće korake za dodavanje novog paketa u kreirani projekt: File -> Add Package -> u tražilicu zalijepiti sljedeći link: <https://github.com/sanzaru/SimpleToast.git> -> Add Package. Paket koji smo dodali sadrži biblioteku s gotovim funkcijama koje ćemo iskoristiti u kreiranju toast obavijesti. Ako je „*simpletoast*“ paket uspješno dodan, možemo krenuti na izradu programskog rješenja.

A screenshot of a computer

Description automatically generated

Slika 9: Dodavanje simpletoast paketa

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | *// ContentView.swift*  *// ToastNotification*  *// Created by Karlo Mikec*  import SwiftUI  import SimpleToast  **struct** ContentView**:** View {  @State private var Toast **=** false    private let toastKarakteristike **=** SimpleToastOptions(  alignment: .bottom,  hideAfter: 3,  animation: .easeIn,  modifierType: .fade  )    var body**:** some View {  VStack (spacing**:** 20){  Image(systemName**:** "person")  .imageScale(.large)  .foregroundStyle(.tint)  Button("Show toast"){  Toast.toggle()  }  }  .simpleToast(isPresented**:** $Toast, options**:** toastKarakteristike){  HStack{  Image (systemName**:** "arrowshape.turn.up.right.fill")  Text ("The Toast is ready !").bold()  }  .padding(23)  .background(Color.gray.opacity(0.5))  .foregroundColor(Color.white)  .cornerRadius(14)  }  }  }  **#Preview {**  ContentView()  } |

Prvo su uključene dvije biblioteke: „SwiftUI“ koja se koristi za razvoj korisničkih sučelja i „*SimpleToast*“ koja omogućuje implementiranje jednostavnih obavijesti. Definirana je nova struktura „*ContentView*“ koja implementira „*View*“ protokol iz biblioteke „*SwiftUI*“. Zatim kreirana je privatna varijabla „*Toast*“ čija je vrijednost postavljena na false. Dekorator „*@State*“ prati kad će se stanje varijable promijeniti i tada će uzrokovati da se sučelje osvježi.

Definirana je konstanta „*toastKarakteristike*“ koja sadrži sve opcije za prikaz obavijesti. Opcije uključuju poravnanje na dnu ekrana, automatsko sakrivanje nakon 3 sekunde, te animaciju prilikom ulaska i tip animacije. U dijelu „*body*“ opisano je kako će se komponenta prikazati. Korišten je „*VStack*“ koji organizira komponente vertikalno s razmakom od 20 točaka između njih. Također, dodana je ikona koja prikazuje osobu pomoću „*Image(systemName: “person“)*“. Kreiran je gumb koji kad se pritisne, mijenja stanje variable „*Toast*“ iz true u false i obratno koristeći metodu „*toggle()*“. Korišten je „*.simpleToast*“ modifikator koji se nalazi u „*SimpleToast*“ biblioteki, a govori sljedeće: ako je „*Toast*“ postavljen na true, obavijest će se prikazati sa zadanim opcijama iz „*toastKarakteristike*“.

Sadržaj unutar same obavijesti uređen je na sljedeći način. Kombinira je slika strelice u desno i tekst unutar horizontalnog stoga „*HStack*“. Tekst je podebljan. Cijeli stog je okružen paddingom, ima sivu pozadinsku boju s 50% prozirnosti, bijelu prednju boju i zaobljene kuteve.

U Xcode-u, SwiftUI nudi mogućnost da pregledamo izgled i ponašanje kreiranih komponenti bez potrebe za pokretanjem aplikacije na emulatoru, to se postiže kroz mehanizam „Canvas Preview“ i upravo iz tog razloga korišten je „*#Preview*“.

Ukratko, osnovna svrha ovog koda je prikazati korisničko sučelje s gumbom. Kada se taj gumb pritisne, korisniku se prikazuje obavijest na kratko vrijeme prije nego što nestane.

A black cell phone with a white screen

Description automatically generated A black cell phone with a white screen

Description automatically generated

Slika 10: Početni zaslon Toast obavijesti. Slika 11: Prikaz Toast obavijesti

# Snackbar obavijest

Snackbar obavijest još je jedan od načina obavještavanja korisnika. Snackbar također daje korisniku obavijesti o aplikaciji koje će se izvršiti u skorije vrijeme ili su se izvršile. Snackbar ne prekida korisnika u obavljanju njegovih radnji na samom uređaju. Snackbar obavijesti obično su jednostavne, s kratkim tekstualnim porukama i ponekad ikonama ili gumbom za zatvaranje. Ovakva vrsta obavijesti obično omogućuje korisnicima da na njih reagiraju, tako što ih korisnik može zatvoriti pritiskom na gumb za zatvaranje. Same snackbar obavijesti služe kako bi informirale korisnika o važnim događajima, porukama ili obavijestima. Njihov cilj je skrenuti korisnikovu pažnju na nešto važno bez ometanja trenutnog zadatka koji korisnik izvršava. Iako su snackbar obavijesti obično prikazane na dnu zaslona, mogu se prilagoditi za prikaz na drugim pozicijama zaslona ako je to potrebno. [[6]](#_Popis_literature) [[7]](#_Popis_literature)

Primjeri situacija u kojima se snackbar obavijesti uključuju:

* Poziv na akciju – snackbar obavijest može pozvati korisnika na određenu akciju, kao što je otvaranje određenog sadržaja ili izbor opcija
* Prikaz upozorenja – ako dođe do greške, snack bar obavijest može se koristiti kako bi obavijestila korisnika o tome i pružila upute za rješenje problema

Snackbar obavijesti su korisne jer pružaju informacije korisnicima na intuitivan način, bez prekidanja njihovog toka rada. Međutim, važno ih je koristiti s razumijevanjem i sukladno načelima dobre prakse kako ne bi postale previše napadne i izazvale frustracije kod korisnika.

## Implementacija Snack bar obavijesti

Kako bi implementirali Snack bar obavijest potrebno je kreirati novi projekt kako je opisano u kraku Postupak kreiranja novog projekta. Nakon što je novi projekt uspješno kreiran možemo krenuti sa implementacijom Snack bara.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | *// ContentView.swift*  *// SnackBar*  *// Created by Karlo Mikec*  import SwiftUI  **struct** ContentView**:** View {  @State private var Snackbar **=** false    var body**:** some View {  Button("Show SnackBar"){  Snackbar **=** true  }  .alert(isPresented**:** $Snackbar) {  Alert(  title: Text("No Internet Connection"),  message: Text("Make sure your device is connected to the internet."),  dismissButton: .**default**(Text("OK"))  )  }  }  } |

Prvo je uključen „*SwiftUI*“ biblioteka, kako bi mogli koristiti njene komponente. Definirana je nova struktura „*ContentView*“ koja implementira „*View*“ protokol. Deklarirana je privatna varijabla „*Snackbar*“ koja je tipa bool i postavljena je na vrijednost false. „*@State*“ koristi se za praćenje promjena u varijabli. Što znači, kada je promjena uočena, korisničko sučelje se ažurira. U ovom slučaju varijabla prati treba li se prikazati Snackbar, odnosno alert.

Svojstvo „*body*“ definira sadržaj koji će se prikazati kada se „*View*“ renderira. „*Button*“ kreiran novi gumb s tekstom „*Show SnackBar*“. Linija „*Snackbar* **=** true“ postavlja vrijednost na true, što rezultira prikazivanjem alerta, jer „*alert*“ reagira na promjenu ove vrijednosti. Uz pomoć „*.alert(isPresented: $Snackbar)*“ pridružuje se alert funkcija gumbu. Unutar dijela „*Alert()*“ definirano je što će se prikazati pritiskom na gumb. Postavljeni su naslov, poruka obavijesti i gumb OK za zatvaranje alerta.

Ukratko ovaj kod definira sučelje s jednim gumbom. Kada se gumb pritisne, pojavit će se alert s porukom o nedostatku internetske veze.

A cell phone with a white screen

Description automatically generated A screen shot of a phone

Description automatically generated

Slika 12: Početni zaslon Snackbar obavijesti Slika 13: Prikaz Snackbar obavijesti

# Floating action button

Floating Action Button (FAB) je interaktivni grafički element u sučeljima mobilnih aplikacija. Najčešće se koristi u aplikacijama koje su dizajnirane prema smjernicama Material Designa, koje je Google predstavio za stvaranje intuitivnih korisničkih iskustva.

FAB je obično mali, okrugli gumb koji lebdi iznad drugih elemenata sučelja i obično ima ikonu unutar sebe. Njegova svrha je istaknuti glavnu akciju koju korisnik može izvršiti u određenom kontekstu. Iako Apple nije implementirao FAB na svojim iOS uređajima, u svrhu ovog seminarskog rada osmišljena je jedna implementacija Float Action Buttona. [[8]](#_Popis_literature) [[9]](#_Popis_literature)

Prednosti FAB-a uključuju:

* **Istaknutost** – vizualno je privlačan i lako primjetan, čime korisnicima olakšava pristup glavnim akcijama.
* **Jasnoća** – na temelju oznake na FAB-u korisnici lako razumiju što će se dogoditi kada ga pritisnu.
* **Jednostavnost upotrebe** – FAB omogućuje brzu i jednostavnu interakciju, često s jednim dodirom.
* **Konzistencija** – korištenje FAB-a prema Material Design smjernicama pomaže održavanju dosljednog i prepoznatljivog korisničkog sučelja.

Floating Action Button je moćan dizajnerski element koji može značajno poboljšati korisničko iskustvo u mobilnim aplikacijama. Njegova sposobnost da istakne ključne akcije i olakša korisnicima pristup važnim funkcijama čini ga neprocjenjivim alatom za razvoj sučelja. Međutim, kao i svi dizajnerski elementi, treba ga pažljivo i promišljeno koristiti kako bi se postigao najbolji krajnji rezultat. S pravilnim pristupom, FAB će pomoći stvoriti intuitivna i ugodna korisnička iskustva te istaknuti funkcionalnosti koje su ključne za aplikaciju. [[10]](#_Popis_literature)

## Implementacija Float Action Button-a

Kao u svakoj implementaciji do sad, prvo je potrebno kreirati novi projekt kako je objašnjeno u poglavlju Postupak kreiranja projekta. Kada je novi projekt kreiran, možemo krenuti sa implementacijom Float action buttona.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | *// ContentView.swift*  *// FloatingActionButton*  *// Created by Karlo Mikec*  import SwiftUI  **struct** ContentView**:** View {  @State private var isFABVisible **=** false  @State private var isAlertPresented **=** false  var body**:** some View {  ZStack {  VStack {  Spacer()  }  VStack {  Spacer()  HStack {  Spacer()  **if** isFABVisible {  Button(action**:** {  isAlertPresented.toggle()  }) {  Image(systemName**:** "message")  .padding(20)  .background(Color.gray)  .foregroundColor(.white)  .clipShape(Circle())  .shadow(radius**:** 10)  }  .transition(.scale)  }  Button(action**:** {  isFABVisible.toggle()  }) {  Image(systemName**:** isFABVisible **?** "xmark" **:** "plus")  .padding(30)  .background(Color.gray)  .foregroundColor(.white)  .clipShape(Circle())  .shadow(radius**:** 10)  }  .transition(.scale)  .padding(20)  }  }  }  .alert(isPresented**:** $isAlertPresented) {  Alert(  title: Text("Confirmed"),  message: Text("You're all set to go"),  dismissButton: .**default**(Text("OK"))  )  }  }  }  **struct** YourApp**:** App {  var body**:** some Scene {  WindowGroup {  ContentView()  }  }  } |

Prvo je uključen „*SwiftUI*“ okvir, koji omogućuje izgradnju korisničkih sučelja. Deklarirana je struktura „*ContentView*“ koja implementira „*View*“ protokol. Zatim deklarirane su dvije privatne varijable koje su povezane sa „*@State*“ svojstvom i njihova vrijednost postavljena je na false. Ovo omogućuje praćenje promjene stanja ovih varijabli i automatsko ažuriranje sučelja kada se stanja ovih varijabli promijene.

Definirano je tijelo prikaza, unutar kojeg je opisano kako će sučelje zapravo izgledati i kako će reagirati na promjene. „*ZStack*“ je kontejner koji preklapa poglede jedan preko drugog. „*VStack*“ je vertikalni stog pogleda, koji sadrži „*Spacer*“ koji poravnava sve poglede prema dnu zaslona.

„*HStack*“ predstavlja horizontalni stog pogleda. Unutar njega provjerava se da li je varijabla „*isFABvisible*“ postavljen na true. Ako je, tada se prikazuje gumb koji mijenja vrijednost „*isAlertPresented*“ na true, kada se na gumb pritisne.

U dijelu „*Image(systemName: “message“)*“ definiran je izgled gumba kao ikona poruke unutar sive kružnice. Efekt „*.transition(.scale)*“ vizualno je uočljiv kod pojavljivanja i nestajanja gumba. Nadalje, kreiran je još jedan gumb, gdje se izgled gumba prikazuje na temelju vrijednosti „*isFABVisible*“. Ako je vrijednost postavljena na true, na gumbu je prikazan znak X, inače je prikazan znak +. Kao i kod prethodnog gumba dodan je tranzicijski efekt i još dodatan prostor oko samog gumba. Kad se napravi interakcija sa gumbom koji na sebi ima ikonu poruke, aktivira se „*..alert(isPresented: $isAlertPresented)*“ i na zaslonu se prikazuje prozor s obavijesti i gumbom OK, za zatvaranje te obavijesti.

Definirana je još struktura „*YourApp*“ koja definira točku ulaska aplikacije i određuje glavni pogled, u ovom slučaju „*ContentView*“ koji će se prikazati kada se aplikacija pokrene.

Ukratko ovaj kod definira sučelje s dva gumba, jedan koji se može prikazati i sakriti, i drugi koji kontrolira taj prvi gumb i također prikazuje upozorenje kada je prvi gumb pritisnut.

A black cell phone with a white screen

Description automatically generated A black cell phone with a white screen

Description automatically generated

Slika 14: Prikaz FAB-a Slika 15: Interakcija s FAB-om

A screenshot of a cell phone

Description automatically generated

Slika 16: Prikaz dodatne interakcije s FAB-om

# Sistemske obavijesti

Sistemske obavijesti na Appleovim iOS uređajima počele su se primjenjivati još od rane iOS 5 verzije, koja je objavljena krajem 2011. godine. Ova verzija donijela je brojne promjene u načinu na koji su obavijesti upravljanje na iOS platformi, uključujući uvođenje „Notification Center“ i obavijesti na zaključanom ekranu. Od tada, sistemske obavijesti razvijale su se kontinuirano sve do dan danas kada je u trenutku pisanja ovog seminarskog rada aktivna verzija iOS 17. Od tada su se sistemske obavijesti kontinuirano razvijale i poboljšavale kroz različite verzije iOS-a, nudeći korisnicima sve više prilagodljivih opcija i bolje iskustvo upravljanja obavijestima. [[11]](#_Popis_literature) [[12]](#_Popis_literature)

Sistemske obavijesti su informacije koje se prikazuju na uređaju kako bi korisniku pružile informacije o različitim događajima, stanjima ili zadacima koji se odvijaju na uređaju ili u pozadini aplikacija. Apple je implementirao sistemske obavijesti na svojim iOS uređajima kroz svoj mobilni operativni sustav kako bi poboljšao korisničko iskustvo.

Neke od ključnih karakteristika obavijesti na iOS uređajima su:

* **Prikazivanje informacija na zaključanom zaslonu** – sistemske obavijesti omogućuju korisnicima da brzo pregledaju važne informacije, poput poruka, e-pošte, obavijesti o aplikacijama bez otključavanja uređaja. To štedi vrijeme i omogućuje brz pristup ključnim informacijama
* **Personalizacija i prilagodljivost** – iOS omogućuje korisnicima da prilagode postavke obavijesti prema svojim potrebama. Moguće je odabrati koje aplikacije smiju prikazivati obavijesti.
* **Obavijesti uživo** – sistemske obavijesti na iOS uređajima omogućuju obavijesti uživo, što znači da se informacije kontinuirano ažuriraju, kao što su promjene vremena ili cijene dionica, bez potrebne za ručnim osvježavanjem
* **Grupiranje obavijesti** – da bi se spriječila navala obavijesti, iOS grupira obavijesti prema aplikacijama ili temama. To olakšava pregled i upravljanje obavijestima.
* **Apple Siri** – glasovni asistent Siri također koristi sistemsku obavijest kako bi korisnicima predložio akcije, kao što su dodavanje događaja u kalendar ili slanje poruka.

Prednosti sistemskih obavijesti na iOS uređajima uključuju povećanje produktivnosti, olakšavanje praćenja važnih informacija, personalizaciju korisničkog iskustva i smanjenje ometanja. Korisnici mogu prilagoditi obavijesti prema potrebama i postaviti prioritete za važne kontakte i aplikacije, čime se osigurava da ne propuste važne informacije, a istovremeno filtriraju manje važne obavijesti. Sistemske obavijesti također omogućuju brz pristup ključnim informacijama bez potrebe za otključavanjem uređaja, što čini korisničko iskustvo bržim i učinkovitijim. [[14]](#_Popis_literature) [[13]](#_Popis_literature)

## Implementacija jednostavne sistemske obavijesti

Kako bi započeli s implementacijom jednostavne sistemske obavijesti, prvo je potrebno kreirati novi projekt čiji je postupak detaljno objašnjen u poglavlju Postupak kreiranja projekta.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | import SwiftUI  import UserNotifications  **struct** ContentView**:** View {  @State private var showNotification **=** false    var body**:** some View {  VStack {  Button("Show notification"){  scheduleNotification()  }  }  }    func scheduleNotification(){  let content **=** UNMutableNotificationContent()  content.title **=** "HOMEASSISTANT"  content.body **=** "Motion detected in the living room."    let trigger **=** UNTimeIntervalNotificationTrigger(timeInterval**:** 5, repeats**:** false)  let request **=** UNNotificationRequest(identifier**:** UUID().uuidString, content**:** content, trigger**:** trigger)    UNUserNotificationCenter.current().add(request){  error **in**  **if** let error **=** error {  print("Error scheduling notification: \(error)")  } **else** {  print("The notification has been successfullz scheduled.")  }  }  }  }  **struct** MyApp**:** App {  @UIApplicationDelegateAdaptor(AppDelegate.self) var appDelegate    var body**:** some Scene {  WindowGroup {  ContentView()  }  }  }  class AppDelegate**:** NSObject, UIApplicationDelegate, UNUserNotificationCenterDelegate {  func application(\_ application**:** UIApplication, didFinishLaunchingWithOptions launchOptions**:** [UIApplication.LaunchOptionsKey **:** Any]**?** **=** nil) **->** Bool {  UNUserNotificationCenter.current().delegate **=** self  UNUserNotificationCenter.current().requestAuthorization(options**:** [.alert, .sound, .badge]) { granted,  error **in**  **if** granted {  print("Premission to display notification is granted.")  } **else** **if** let error **=** error {  print("Error requesting notification premission: \(error)")  }  }  **return** true  }  } |

Na samome početku uključene su dvije biblioteke „*SwiftUI*“ koja se koristi za izradu korisničkog sučelja i „*UserNotifications*“koja se koristi za rad s obavijestima na iOS uređajima. Zatim, definirana je „*ContentView*“struktura koja implementira „*View*“ protokol koji se koristi za prikaz korisničkog sučelja. Uz pomoć „@State“ kreirana je privatna varijabla „*showNotification*“ čija vrijednost je postavljena na false i time je onemogućeno prikazivanje sistemske obavijesti odmah prilikom učitavanja aplikacije.

Unutar „*body*“ dijela definirano je sve što će se prikazati u sučelju. „*VStack*“ je kontejner koji sve svoje pod elemente slaže vertikalno. Također, kreiran je gumb s tekstom „*Show notification*“. Kada korisnik pritisne gumb, pozvat će se funkcija „*scheduleNotification()*“.

Unutar funkcija „*scheduleNotification()*“ osmišljen je način kako je planirana obavijesti.

Stvoren je sadržaj obavijesti i postavljeni su naslov i tijelo obavijesti. Uz pomoć varijable „*trigger*“ stvoren je okidač koji će aktivirati obavijest nakon 5 sekundi i neće je ponavljati.

U varijabli „*request*“ kreira se zahtjev za obavijest s jedinstvenim identifikatorom, prethodno definiranim sadržajem „*content*“ i prethodno definiranim okidačem „*trigger*“. „*UNUserNotificationCenter*“ iskorišten je kako bi se zahtjev za obavijest dodao trenutnom centru za obavijesti. Varijabla „*error*“ korištena je kao pomoć prilikom debagiranja aplikacije, a radi na sljedeći način: ako postoji greška kod planiranja obavijesti u konzolu ispisuje se pogreška. Inače, ispisuje se poruka o uspješno zakazanom planiranju obavijesti.

Unutar „*MyApp*“ definirana je glavna aplikacija. „*@UIApplicationDelegateAdapto*“ iskorišten je kako bi se „*AppDelegate*“ povezao s aplikacijom. Unutar „*body*“ dijela definirano je glavno korisničko sučelje aplikacije. Kad se aplikacija pokrene prikazat će se „*ContentView*“.

Kreirana je posebna klasa „*AppDelegate*“ koja implementira protokole „*UIApplicationDelegate*“ i „UNUserNotificationCenterDelegate“. Unutar te klase nalazi se funkcija „*application*“ koja se poziva kad se aplikacija pokrene, postavlja trenutnu instancu klase kao delegat za centar obavijesti. Prema standardima tvrtke Apple, šalje se zahtjev korisniku za dopuštenje prikaza obavijesti. Kao još jedna pomoć u debagiranju kreirana je provjera da li je korisnik dao dozvolu za primanje obavijesti. Ako je dozvoljeno u konzolu ispisat će se poruka da je dozvola za prikaz obavijesti dozvoljena, u suprotnom ispisat će se da je dozvola za slanje obavijesti odbijena.

Ukratko, ovaj kod predstavlja jednostavnu aplikaciju koja korisniku omogućuje pritiskom na gumb da zakaže obavijest koja će se prikazati nakon 5 sekundi.

A black cell phone with a white screen

Description automatically generated A screen shot of a cell phone

Description automatically generated

Slika 17: Jednostavna sistemska obavijest (1). Slika 18: Jednostavna sistemska obavijest (2)

A cell phone with a screen

Description automatically generated

Slika 19: Jednostavna sistemska obavijest (3)

## Implementacija složene sistemske obavijesti

Za implementaciju složene sistemske obavijesti korišten je jednim dijelom kod iz poglavlja Implementacija jednostavne sistemske obavijesti. Stoga većina programskog rješenja objašnjena je u prijašnjem poglavlju.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | import SwiftUI  import UserNotifications  **struct** ContentView**:** View {  var body**:** some View {  Home()  }  }  **struct** Home**:** View {  @StateObject var delegate **=** NotificationDelegate()  var body**:** some View {  Button("Notification", action**:** notifyMe)  .onAppear {  UNUserNotificationCenter.current().requestAuthorization(options**:** [.alert, .sound, .badge]) { \_, \_ **in** }  UNUserNotificationCenter.current().delegate **=** delegate  }  }  func notifyMe() {  let content **=** UNMutableNotificationContent()  content.title **=** "HOMEASSISTANT"  content.subtitle **=** "Movement detection at living room!"  content.categoryIdentifier **=** "ACTIONS"  let image **=** "motion-sensor"  **if** let imageURL **=** Bundle.main.url(forResource**:** image, withExtension**:** "png") {  let attachment **=** try**!** UNNotificationAttachment(identifier**:** "kuca", url**:** imageURL)  content.attachments **=** [attachment]  }  let trigger **=** UNTimeIntervalNotificationTrigger(timeInterval**:** 5, repeats**:** false)  let request **=** UNNotificationRequest(identifier**:** "identifier", content**:** content, trigger**:** trigger)  let close **=** UNNotificationAction(identifier**:** "CLOSE", title**:** "Close", options**:** .destructive)  let category **=** UNNotificationCategory(identifier**:** "ACTIONS", actions**:** [close], intentIdentifiers**:** [], options**:** [])  UNUserNotificationCenter.current().setNotificationCategories([category])  UNUserNotificationCenter.current().add(request) { \_ **in** }  }  }  class NotificationDelegate**:** NSObject, ObservableObject, UNUserNotificationCenterDelegate {  @Published var alert **=** false    func userNotificationCenter(\_ center**:** UNUserNotificationCenter, willPresent notification**:** UNNotification, withCompletionHandler completionHandler**:** @escaping (UNNotificationPresentationOptions) **->** Void) {  completionHandler([.badge, .banner, .sound])  }  } |

U ovom programskom rješenju prvo su uključeni moduli „*SwiftUI*“ i „*UserNotifications*“ koji predstavljaju skup alata za grafičko sučelje aplikacije i skup alata za rad s obavijestima na uređaju. Zatim definirana je struktura „*ContentView*“ unutar koje se nalazi „*body*“ gdje će se prikazivati zadani sadržaj. Definirana je još jedna struktura pod nazivom „*Home*“ koja će se prikazati kao sadržaj unutar „*ContentView*“. Kreiran je gumb s tekstom „*Notification*“ i kada korisnik pritisne na zadani gumb, poziva se funkcija „*notifyMe*“. Iskorišten je modifikator „*.onAppear*“ koji omogućuje da se kod izvrši kada se pogled prvi put prikaže na ekranu. Pošto tvrtka Apple vodi strogu politiku o zaštiti korisničkih podataka, od aplikacija traže da zatraže dopuštenje od korisnika prije nego, aplikacija krene slati obavijesti. Tako je i ovdje u linijama 16 i 17 implementirano da aplikacija traži dozvolu od korisnika za slanje obavijesti. Zatražene su 3 vrste dozvola: „*.alert*“ traži dozvolu za prikazivanje upozorenja ili bannera, „*.sound*“ traži dozvolu za reprodukciju zvuka kada stigne obavijest i „*.badge*“ traži dozvolu za ažuriranje brojača na ikoni aplikacije.

Postavljen je delegat za centar obavijesti. Ideja je bila da se određena funkcija delegira drugom objektu. U ovom primjeru objekt „delegate“ koji je instanca klase „*NotificationDelegate*“ postavljen je da upravlja događajima vezanim za obavijesti. Kada se dogodi neka radnja vezana uz obavijesti (npr. dođe obavijest kad je aplikacija otvorena), metode unutar objekta „delegate“ zadužene su za obradu tih događaja.

Kreirana je funkcija „*notifyMe()*“ unutar koje je stvoren objekt tipa „*UNMutableNotificationContent*“ koji se koristi za definiranje sadržaja obavijesti koju želimo poslati, u ovom slučaju to su naslov, podnaslov i tekst poruke. U sljedećem koraku definirali smo ime slike koju želimo priložiti obavijesti. Dohvatili smo putanju do slike „*motion-sensor.png*“ koja se nalazi unutar glavnog paketa aplikacije. Ako tražena slika postoji tada se ona dodaje sadržaju obavijesti „*content.attachments = [attachment]*“.

Kreiran je „*trigger*“ za obavijesti. Ovaj brojač šalje obavijest nakon što prođe određeni vremenski interval. U konkretnom primjeru, obavijest će biti poslana 5 sekundi nakon što je postavljena, s „*repeats: false*“ obavijest je ograničena da se samo jednom prikaže, a ne svakih 5 sekundi.

Sada kada je obavijest kreirana i kada se pokaže nakon 5 sekundi, stvorena je akcija koja će biti dostupna korisniku kada vidi obavijest. Ova akcija ima naslov „*Close*“ i kad korisnik pritisne ovu akciju, obavijest će biti maknuta. „*options: .destructive*“ skreće pažnju da će akcija biti naznačena crvenom bojom i nakon pritiska na nju da će se obavijest skloniti.

Uz sve do sad navedeno, kreirana je kategorija obavijesti s identifikatorom „*ACTIONS*“ koja je ranije postavljena u „*content.categoryIdentifier*“. Ova kategorija sadrži prethodno definiranu akciju „*Close*“. Nadalje, novokreirana kategorija obavijesti dodana je u trenutni centar obavijesti. Tako sustav može prepoznati kategoriju i povezane akcije kad obavijest stigne. Sada je preostalo da dodamo zahtjev za obavijest koji sadrži sve prethodno definirane informacije u trenutni centar obavijesti. Time je postignuto da se obavijest pošalje korisniku nakon 5 sekundi kako je i prethodno definirano triggerom. Ovo je postignuto sljedećom linijom koda „*UNUserNotificationCenter.Current().add(request)*“, time je ujedno i završena funkcija „*notifyMe()*“.

Ostalo je pojasniti još par linija koda. Klasa „*NotificationDelegate*“ nasljeđuje od „*NSObject*“ i implementira dva protokola „*ObservableObject*“ i „*UNUserNotificationCenterDelegate*“. Protokol „*ObservableObject*“ dolazi iz SwiftUI-a koji omogućuje korisnicima da obavještavaju svoje View-ove o promjenama u podacima, odnosno omogućuje ažuriranje sučelja. Drugi spomenuti protokol omogućuje rukovanje obavijestima, poput definiranja kako se obavijesti prikazuju dok je aplikacija aktivna. „*@Published*“ omogućuje osvježavanje podataka kada se vrijednost ove varijable promijeni. Kada se „*alert*“ promijeni, svi SwiftUI pogledi koji ovise o toj varijabli automatski će se osvježiti.

A black cell phone with a white screen

Description automatically generated A screen shot of a cell phone

Description automatically generated

Slika 20: Složena sistemska obavijest (1) Slika 21: Složena sistemska obavijest (2)

A screen shot of a cell phone

Description automatically generated

Slika 22: Složena sistemska obavijest (3)

# Korisnički dijalozi

Dijalog je najlakše zamisliti kao mali prozor koji traži od korisnika da donese neku vrstu odluke. Može zatražiti korisnika da unese neke informacije. Dijalog se ne prikazuje preko cijelog zaslona, već se prikazuje u obliku pop-up prozora koji zahtijeva od korisnika da poduzme određenu radnju prije nego nastavi dalje. Alert je izvrstan primjer dijaloga između uređaja i korisnika, jer daje upozorenje korisniku i skreće mu pažnju na važne informacije, a s druge strane korisnik mora pritisnuti ili neki gumb ili unesti neke podatke ovisno o obavijesti, kako bi potvrdio da je primio važne informacije. [[15]](#_Popis_literature) [[16]](#_Popis_literature)

## Implementacija korisničkog dijaloga

Za implementaciju korisničkog dijaloga, prvo je potrebno kreirati novi projekt prema uputama u poglavlju Postupak kreiranja projekta.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import SwiftUI  **struct** ContentView**:** View {  @State private var isAlert **=** false  @State private var showErrorAlert **=** false  @State private var Username**:** String **=** ""  @State private var Password**:** String **=** ""    private let correctUsername **=** "Kmikec20"  private let correctPassword **=** "12345"    var body**:** some View {  Button("Sign In") {  isAlert **=** true  }  .alert("Sign In", isPresented**:** $isAlert, actions**:** {  TextField("Username", text**:** $Username)  SecureField("Lozinka", text**:** $Password)    Button("Sign In", action**:** {  **if** Username **==** correctUsername **&&** Password **==** correctPassword {  print("Successful login!")  } **else** {  showErrorAlert **=** true  }  })    Button("Cancel", role**:** .cancel, action**:** {})  }, message**:** {  Text("Enter correct user credentials!")  })  .alert("Wrong credentials!", isPresented**:** $showErrorAlert) {}  }  } |

Prvo je uključena biblioteka „*SwiftUI*“ za izradu korisničkih sučelja. Definirana je struktura „*ContentView*“ koja implementira „*View*“ protokol. Kreirana je privatna varijabla „*isAlert*“ čija je vrijednost postavljena na false. Varijabla prati kada se treba prikazati dijaloški okvir za prijavu. Privatna varijabla „*showErrorAlert*“ čija je vrijednost također postavljena na false, prati kada se treba prikazati upozorenje o pogrešnom unosu podataka. Privatne varijable „*Username*“ i „Password“ kreirane su s ciljem da prate trenutačnu vrijednost unesenog korisničkog imena i lozinke u zadano polje. Također, unaprijed su definirani korisničko ime s varijablom „*correctUsername*“ i lozinka s varijablom „*correctPassword*“.Ove dvije varijable definiraju točno korisničko ime i lozinku koje se koriste za provjeru kod prijave.

Unutar „*body*“ dijela, definiran je gumb „*Sign In*“ koji kada se pritisne pokreće prikaz dijaloškog okvira za prijavu. Modifikator „*.alert*“ definira dijaloški okvir s naslovom „*Sign In*“ koji će se prikazati kada je vrijednost „*isAlert*“ postavljena na true. Element „*TextField*“ predstavlja tekstualno polje gdje korisnik unosi svoje korisničko ime. Dok, „SecureField“ predstavlja sigurno polje za unos gdje korisnik unosi svoju lozinku i tekst je skriven radi sigurnosti. Unutar dijaloškog okvira kreiran je još jedan gumb „*Sign In*“ i akcija koja će provjeriti da li su uneseni korisničko ime i lozinka jednaki. Da li je „*Username*“ jednaka vrijednosti „*correctUsername*“ i da li je „*Password*“ jednaka vrijednosti „*correctPassword*“. Ako su korisničko ime i lozinka točni, ispisat će se poruka u konzolu „*Successful login!*“ inače pojavit će se novi okvir s porukom o unesenim pogrešnim podacima.

Ukratko, ovo programsko rješenje predstavlja gumb s tekstom „*Sign In*“. Kada korisnik pritisne gumb, aplikacija provjerava da li su uneseni podaci ispravni. Ako jesu, ispisat će se poruka u konzolu. Ako nisu, korisnik će vidjeti upozorenje na zaslonu uređaja.

A screen shot of a cell phone

Description automatically generated A screen shot of a phone

Description automatically generated

Slika 23: Prikaz korisničkog dijaloga (1) Slika 24: Prikaz korisničkog dijaloga (2)

A screen shot of a phone

Description automatically generated

Slika 25: Prikaz korisničkog dijaloga (3)

# Zaključak

Ovim seminarskim radom objašnjeni su načini obavještavanja u iOS operativnom sustavu. Pažnja je skrenuta na sljedeće načine obavještavanja: Toast, Snackbar, Floating action button, Sistemske obavijesti, te Korisničke dijaloge između korisnika i sustava. Sve spomenute cjeline uz teorijsku obradu, potkrijepljene su i implementacijom programskog rješenja uz prikaz slikovitih materijala. Iako se sve navedene obavijesti ne pojavljuju u iOS okruženju, u svrhu ovog seminarskog rada su implementirane i prikazane kako bi one izgledale u iOS okruženju. Sistemske obavijesti su uočljive u centru za obavijesti, na zaključanom zaslonu, pa čak i na početnom zaslonu. Na taj način korisniku su dostupne sa više različitih strana. Gdje god se nalazio u bilo kojem trenutku na uređaju, lako može pristupiti obavijestima. Korisničke dijaloge možemo razmatrati kao višu razinu komunikacije operacijskog sustava prema korisniku, jer daju upozorenja korisniku i očekuju direktnu interakciju od strane korisnika. Svime, prethodno spomenutim u ovome radu obavijesti su sastavna i neizbježna komponenta svakog operacijskog sustava. Možemo reći da putem obavijesti sustav komunicira s korisnicima i korisnici daju povrate informacije sustavu. Obavijesti znatno olakšavaju svakodnevno korištenje različitih uređaja i vrlo su važna komponenta svakog korisničkog iskustva.

# Popis svih slika

[Slika 1: Otvaranje sa zaključanog zaslona 2](#_Toc149427936)

[Slika 2: Upravljanje obavijestima 2](#_Toc149427937)

[Slika 3: Promjena načina prikaza obavijesti 2](#_Toc149427938)

[Slika 4: Prvi korak kreiranja novog projekta 3](#_Toc149427939)

[Slika 5: Drugi korak kreiranja novog projekta 4](#_Toc149427940)

[Slika 6: Treći korak kreiranja novog projekta 4](#_Toc149427941)

[Slika 7: Četvrti korak kreiranja novog projekta 5](#_Toc149427942)

[Slika 8: Peti korak kreiranja novog projekta 5](#_Toc149427943)

[Slika 9: Dodavanje simpletoast paketa 7](#_Toc149427944)

[Slika 10: Početni zaslon Toast obavijesti 9](#_Toc149427945)

[Slika 11: Prikaz Toast obavijesti 10](#_Toc149427946)

[Slika 12: Početni zaslon Snackbar obavijesti 13](#_Toc149427947)

[Slika 13: Prikaz Snackbar obavijesti 14](#_Toc149427948)

[Slika 14: Prikaz FAB-a 18](#_Toc149427949)

[Slika 15: Interakcija s FAB-om 18](#_Toc149427950)

[Slika 16: Prikaz dodatne interakcije s FAB-om 19](#_Toc149427951)

[Slika 17: Jednostavna sistemska obavijest (1) 23](#_Toc149427952)

[Slika 18: Jednostavna sistemska obavijest (2) 24](#_Toc149427953)

[Slika 19: Jednostavna sistemska obavijest (3) 24](#_Toc149427954)

[Slika 20: Složena sistemska obavijest (1) 27](#_Toc149427955)

[Slika 21: Složena sistemska obavijest (2) 28](#_Toc149427956)

[Slika 22: Složena sistemska obavijest (3) 28](#_Toc149427957)

[Slika 23: Prikaz korisničkog dijaloga (1) 31](#_Toc149427958)

[Slika 24: Prikaz korisničkog dijaloga (2) 32](#_Toc149427959)

[Slika 25: Prikaz korisničkog dijaloga (3) 32](#_Toc149427960)

\* Slike 1 – 3 preuzete su sa [stranice](https://support.apple.com/hr-hr/HT201925)

\* Slike 4 – 25 samostalna su izrada

# Popis literature

[[1]](#_Uvod) [Upotreba obavijesti na iPhone ili iPad uređajima](https://support.apple.com/hr-hr/HT201925)

[[2]](#_Uvod) [Notifications - Apple Developer Documentation](https://developer.apple.com/design/human-interface-guidelines/notifications)

[[3]](#_Postupak_kreiranja_novog) [Creating an Xcode project fo an app - Apple Developer Documentation](https://developer.apple.com/documentation/xcode/creating-an-xcode-project-for-an-app)

[[4]](#_Toast_obavijest) [Alternatives to using toasts](https://developer.android.com/guide/topics/ui/notifiers/toasts)

[[5]](#_Toast_obavijest) [Toast Notifications](https://design-system.hpe.design/templates/toast-notifications)

[[6]](#_Snack_bar_obavijest) [Snackbars](https://m2.material.io/components/snackbars/ios)

[[7]](#_Snackbar_obavijest) [Alerts - Apple Developer Documentation](https://developer.apple.com/design/human-interface-guidelines/alerts)

[[8]](#_Floating_action_button) [Buttons: Float action button](https://m2.material.io/components/buttons-floating-action-button)

[[9]](#_Floating_action_button) [Floating Action Buttons](https://guides.codepath.com/android/floating-action-buttons)

[[10]](#_Floating_action_button) [Floating Action Button in UX Design](https://uxplanet.org/floating-action-button-in-ux-design-7dd06e49144e)

[[11]](#_Sistemske_obavijesti) [User Notifications - Apple Developer Documentation](https://developer.apple.com/documentation/usernotifications)

[[12]](#_Sistemske_obavijesti) [Managing notifications - Apple Developer Documentation](https://developer.apple.com/design/human-interface-guidelines/managing-notifications)

[[13]](#_Sistemske_obavijesti) [Handling notifications and dotification-related actions](https://developer.apple.com/documentation/usernotifications/handling_notifications_and_notification-related_actions)

[[14]](#_Sistemske_obavijesti) [iOS 17](https://www.apple.com/hr/newsroom/2023/06/ios-17-makes-iphone-more-personal-and-intuitive/)

[[15]](#_Korisnički_dijalozi) [Asking permission to use notifications - Apple Developet Documentation](https://developer.apple.com/documentation/usernotifications/asking_permission_to_use_notifications)

[[16]](#_Korisnički_dijalozi) [Use the Show Alert action in a shortcut on iPhone](https://support.apple.com/en-mt/guide/shortcuts/apdb9661c761/ios)