iOS alapú szoftverfejlesztés - Labor 08

A labor témája

A labor célja egy üzenetküldő alkalmazáson keresztül megismerni a SwiftUI egyedi nézeteinek használatát, illetve a beépített `List` működését.

SwiftUI néztek létrehozása

GeoMessenger

> Hozzunk létre egy `Single View App`ot **GeoMessenger** névvel a `Developer` könyvtárba!

> Az `Interface` legyen `SwiftUI`, a `Life Cycle` pedig `SwiftUI App`

> Mielőtt nekilátnánk a felhasználói felület fejlesztésének, hozzunk létre egy `Message.swift` nevű fájlt, ami az üzeneteiket fogja reprezentálni.

```
struct Message {
    var id: Int
    var sender: String
    var content: String
}
```

Hozzunk létre némi *mock*-olt üzenetet a teszteléshez. Ez nem túl elegáns, de a gyors layout teszteléshez megfelelő.

> Nyissuk meg a `GeoMessengerApp.swift` fájlt és a `@main` előtt adjuk hozzá az alábbi `messages` tömböt.

```
import SwiftUI

var messages: [Message] = [Message(id: 1, sender:"Joe", content: "First
Message"),
                           Message(id: 2, sender:"Joe", content: "Lorem
ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur."),
                           Message(id: 3, sender:"Teo", content: "Third
```

```
Message")]

@main
struct GeoMessengerApp: App {
```

Copy-paste barát gist link itt

> Lépjünk át a `ContentView.swift`-re és a `Text` helyén hozunk létre egy `List`-et. Adjunk hozzá az új `List`-ünkhöz 3 `Text`-et, aminek paraméterül átadjuk a `messages` tömb három elemét.

```
var body: some View {
    List(){
        Text(messages[0].content)
        Text(messages[1].content)
        Text(messages[3].content)
    }
}
```

Ez nagyon szép, statikus lista, de nekünk azért kellene valami dinamizmus.

> Adjuk át a `messages` tömböt a listának és a Closure-be vegyünk fel egy `message` nevű paramétert.

```
var body: some View {
    List(messages){ message in
        Text(message.content)
    }
}
```

Majdnem jó, de hibaüzenetet kapunk, mivel a `Message` nem valósítja meg az `Identifiable` protocol-t.

> Menjünk át a `Messages.swift`-re és adjuk hozzá az `Identifiable` protocol-t.

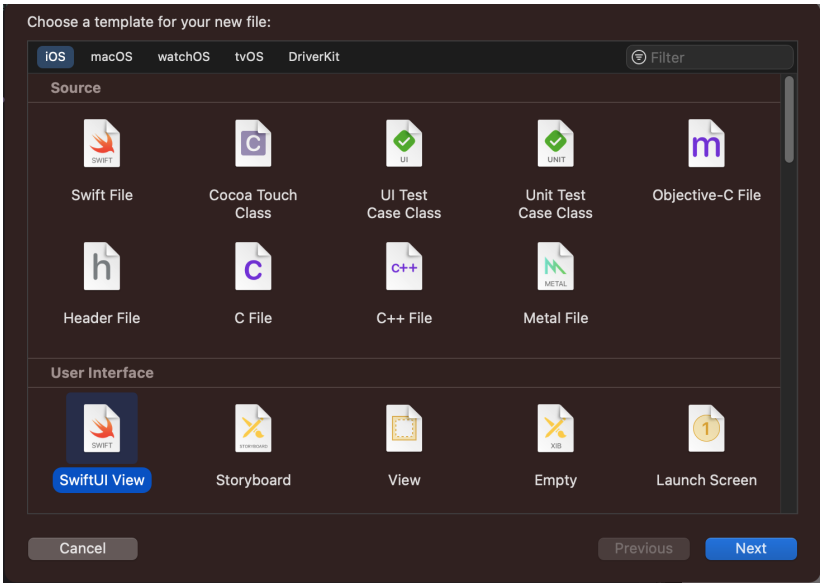
```
struct Message : Identifiable{
    var id: Int
    var sender: String
    var content: String
}
```

Ezzel meg is oldottuk a feladatot, mivel az `id` már szerepelt az osztályban, mint property.

CustomView

A `List` gyakorlatilag tetszőleges nézetet képes egymás alatt megjeleníteni, így kicsit csinosítsuk ki a sorokat.

> Ehhez hozzunk létre egy új `SwiftUI View`-t `ListRowView` névvel az alábbi sablont használva.



> Hogy használni tudjuk az új "cellánkat", kellene egy property, ami az akutális `message`-et tartalmazza.

```
struct ListRowView: View {

    var message : Message
```

> Mivel a Preview-nál már probléma adódik, javítsuk meg, adjunk át egy elemet a `messages` tömbből

```
struct ListRowView_Previews: PreviewProvider {
    static var previews: some View {
        ListRowView(message: messages[0])
    }
}
```

> A `Text` szövegét cseréljük ki a `message content` értékére, hogy dinamikus tartalmat tudjunk megjeleníteni.

```
var body: some View {
    Text(message.content)
}
```

Térjünk vissza a `ContentView`-hoz és cseréljük le a `Text`-et a saját nézetünkkel.

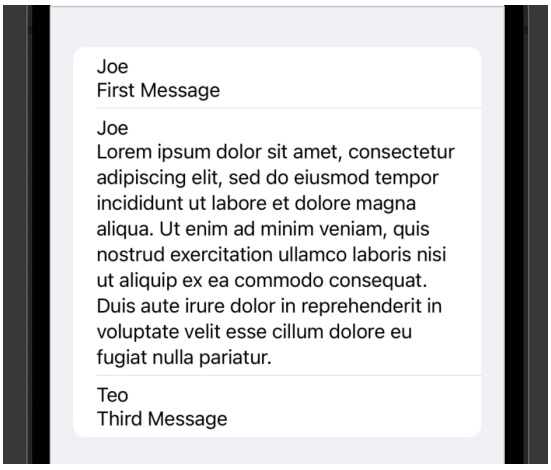
```
List(messages){message in
    ListRowView(message:message)
}
```

Micsoda hihetetlen változás! Pont úgy néz ki, mint eddig. Adjunk hozzá további részleteket.

> Térjünk vissza `ListRowView`-ba és ágyazzzuk be a `Text`-et egy `VStack`-be, ami legyen a `.leading` irányba igazítva majd adjunk hozzá egy másik `Text`-et, ami a `sender`-t fogja tartalmazni.

```
VStack (alignment: .leading) {
    Text(message.sender)
    Text(message.content)
}
```

Ezt kellene kapnunk.



NetworkManager

Most tegyük még dinamikusabbá a listánkat; töltsük le a netről!

> Hozzunk létre egy új `swift` fájlt `NetworkManager` névvel és benne hozzunk létre egy ugyanilyen nevű osztályt.

> Tegyük az osztályt `ObservableObject`-té és adjunk hozzá egy `@Published Message` tömböt `fetchedMessages` névvel.

```
class NetworkManager: ObservableObject {
    @Published var fetchedMessages = [Message]()

}
```

> Hozzuk létre az alábbi lekérdező metódust, amivel már korábban is találkoztunk

```
func fetchMessages() {
    let urlString =
"http://5glab.educationhost.cloud/igniter/public/messages"
    if let url = URL(string: urlString){
        let session = URLSession(configuration: .default)
        let task = session.dataTask(with: url) { (data, response, error)
in
            if error == nil{
                let decoder = JSONDecoder()
                if let data = data{
                    do{
                        let messages = try
decoder.decode(Array<Message>.self, from: data)
                        DispatchQueue.main.async {
                            self.fetchedMessages = messages
                        }
                    } catch{
                        print(error)
                    }
                }
            }
        }
        task.resume()
    }
}
```

Copy-paste barát gist link itt

Hibaüzenetet kapunk, mert a `Message` nem valósítja meg a `Codable` protocolokat.

> Menjünk át a `Message`-be és pótoljuk a hiányosságot. Továbbá egészítsük ki a `CodingKeys enum`-mal is.

```
struct Message : Identifiable, Codable{
    var id: Int
    var sender: String
    var content: String
```

```
    enum CodingKeys: String, CodingKey {
        case id = "message_id"
        case sender = "from_user"
        case content
    }
}
```

> Menjünk át a `ContentView`-ba és hozzunk létre egy `NetworkManager` `@ObjservedObject` property-t.

```
struct ContentView: View {
    @ObservedObject var networkManager = NetworkManager()
```

> Cseréljük ki a `List`-ben a `messages` tömböt az imént létrehozott `networkManager` `fetchedMessages` tömbjére.

```
var body: some View {
    List(networkManager.fetchedMessages){message in
        ListRowView(message:message)
    }
}
```

Most már csak arra van szükség, hogy frissüljön a lista, amikor betöltődik nézet

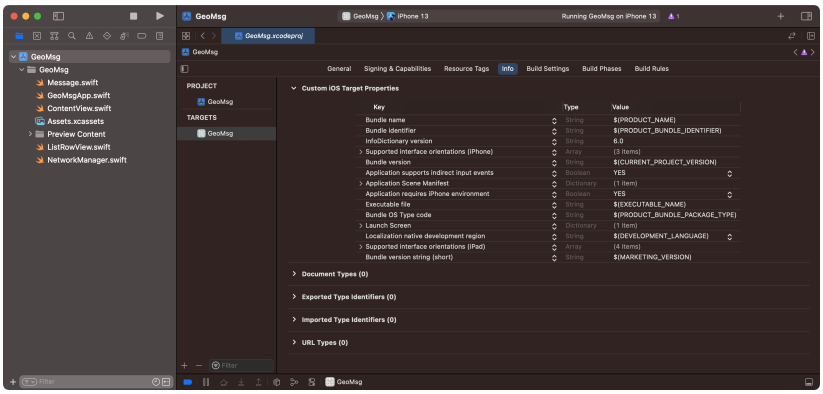
> Adjuk hozzá az `.OnAppear` modifier-t a `List`-hez, az alábbi tartalommal.

```
List(networkManager.fetchedMessages){message in
    ListRowView(message:message)
}
.onAppear {
    self.networkManager.fetchMessages()
}
```

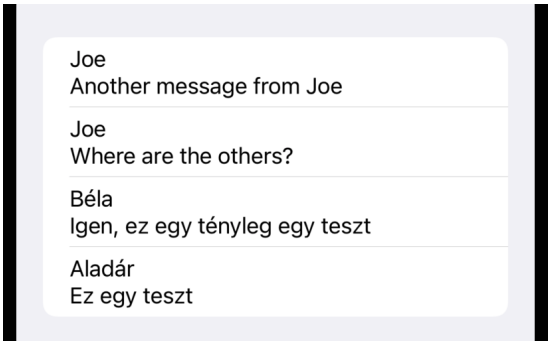
Teszteljük az alkalmazást szimulátoron futtatva!

Sajnos hibaüzenetet kapunk, mivel a szerver egyszerű http kapcsolaton keresztül kommunikál, így be kell állítani az `info.plist`-en belül `App Transport Security Settings`-ben az `Allow Arbitrary Loads` kulcsot `YES`-re

Mivel most nem látjuk az `info.plist`-et a `Project Navigator`-ban, így be kell menni a projektbeállításokon belül az `Info` tab-ra.



Most már működik a lekérdezés!



Önálló feladat

Üzenetküldés

> Hogy üzenetet tudjunk küldeni először bővítsük ki a `NetworkManager`-t egy `sendMessage`!
> Látható, hogy ha sikerült az üzenetküldés, automatikusan meghívódik a lekérdezés.

```
func sendMessage(message: String, sender: String){
    let message = Message(id:2, sender: sender, content: message)
    let encoder = JSONEncoder()

    guard let jsonData = try? encoder.encode(message) else { return }

    let url = URL(string:
"http://5glab.educationhost.cloud/igniter/public/messages/add")
    var request = URLRequest(url: url!)
    request.httpMethod = "POST"
    request.setValue("application/json", forHTTPHeaderField: "Content-
Type")

    let session = URLSession(configuration: .default)
    session.uploadTask(with: request, from: jsonData) { data, response,
error in
        if let error = error {
            print(error)
            return
        } else {
            self.fetchMessages()
```

```
        }
    }.resume()
}
```

Copy-paste barát gist link itt

Menjünk át a `ContentView`-ba és adjunk hozzá egy `TextField`-et, amibe az üzenetet fogjuk írni.

> Ehhez először egy `VStack`-be kell beletenni a `List`-ünket, illetve az `.onAppear` modifier-t is érdemes egy szinttel kijjebb tenni.

```
VStack {
    List(networkManager.fetchedMessages){message in
        ListRowView(message:message)
    }
    TextField("New message", text: $newMessage)
}.onAppear {
    self.networkManager.fetchMessages()
}
```

> Ha pedig `TextField`-et hozunk létre, akkor kell egy `@State` property is, ami tárolja majd az üzenetet.

```
@State var newMessage = ""
```

> Valósítsuk meg az üzenetküldést a `TextField .onSubmit` modifier-ének megvalósításával.

```
TextField("New message", text: $newMessage)
.onSubmit {
    self.networkManager.sendMessage(message: newMessage, sender: "
[USERNAME]")
    self.newMessage = ""
}
```

> Mielőtt kipróbálnánk, cseréljük le a "[USERNAME]"-et, a sajátunkra.

Akkor is érdemes lenne a listát frissíteni, ha nem küldünk üzenetet. Erre iOS 15-től egy egyszerű lehetőségünk van, a `List`-ben elérhető `.refreshable` modifier segítségével.

> A `.refreshable` modifier segítségével hívjuk meg a lista frissítését, ha a felhasználó "túlhúzza" a listát.

```
.refreshable {
    self.networkManager.fetchMessages()
}
```

UI csinosítása

Nem valami "karakteres" az alkalmazásunk, amin változtatni kellene.

> Állítsunk be először is a `TextField` stílusát a `.textFieldStyle` modifier segítségével `roundedRect`-re, majd állítsunk be némi `padding`-et is.

Ha azt szeretnénk, hogy oldalanként eltérő legyen a padding, akkor egy `EdgeInsets` objektumot kell átadni neki az alábbi módon:

```
.padding(EdgeInsets(top: 0, leading: 7, bottom: 2, trailing: 7))
```

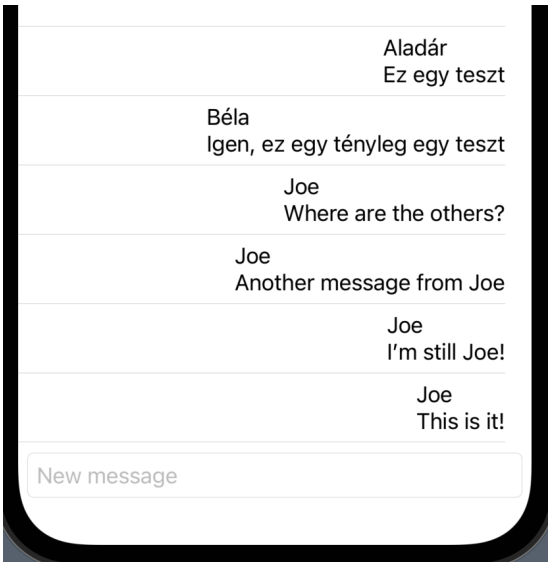
Mivel a `List` a legfelső elemhez rakja a legújabb üzenetet, nem igazán hasonlít egy szokásos chatprogramra az alkalmazásunk.

> Fordítsuk meg a `List`-et a következő trükkel: A `.rotationEffect` modifier segítségével forgassuk el 180 fokban a listát, majd ugyanezt tegyük meg külön a `ListRowView`-kra is

```
List(networkManager.fetchedMessages){message in
    ListRowView(message:message)
        .rotationEffect(.degrees(180))
}
    .rotationEffect(.degrees(180))
```

Mivel most fejjel lefelé van a lista, így nem lefelé kell húzni a listát, hogy frissüljön, hanem felfelé.

> Mivel kicsit kusza lett a látvány, állítsuk be a `List` stílusát a `.listStyle` modifier segítségével `.plain`-re.

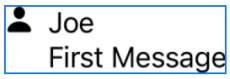


Kicsit talán jobb, de még nem az igazi...

> Térjünk át a `ListRowView`-ra és csinoítsuk ki azt is

> Az üzenet mellé tegyünk be egy kis ikont (amit majd később a szervertől fogunk lekérdezni). Ehhez először ágyazzuk be a `VStack`-et egy `HStack`-be, aminek felülre legyen rendezve.

```
HStack (alignment: .top) {
   VStack (alignment: .leading) {
```



> Adjunk hozzá a külső `HStack`-hez egy `Image`-t, aminek egy rendszer által biztosított ikont adunk át a `systemName` paraméter segítségével.

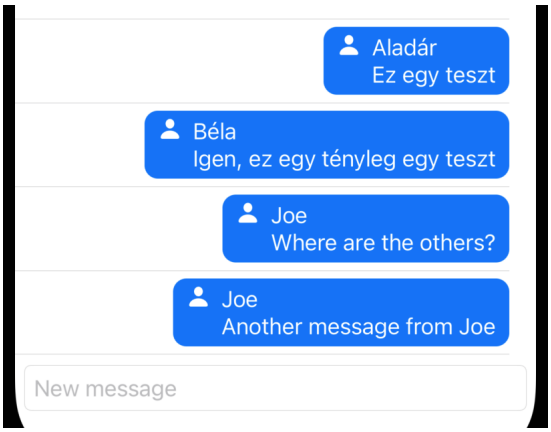
```
HStack (alignment: .top) {
    Image(systemName: "person.fill")
```

> Állítsunk be egy kék `RoundedRectangle`-t a `HStack` hátteréül. És ha már ezt tesszük állítsuk át a szövegek és a kép színét fehérre.

```
.background(
    RoundedRectangle(cornerRadius: 10)
        .fill(Color.blue)
)
.foregroundColor(.white)
```

> Érdemes némi padding-et is állítani hozzá egy `EdgeInsets` átadásával: Függőleges irányokba 5-5 képpont legyen, vízszintes irányokban pedig 10-10 képpont.

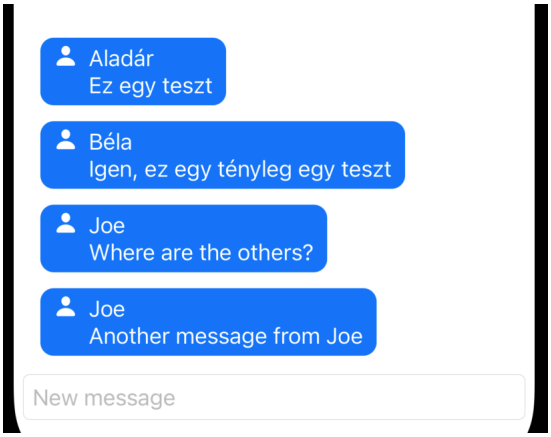
Figyeljünk oda, hogy a `.background` beállítása előtt kell a `.padding`-et beállítani!



Majdnem jó a látvány, de az összes üzenet - a forgatás miatt - a rossz oldalra rendeződik. Ráadásul a szeparátor is látszik.

> A rendezés megoldására ágyazzuk be a külső `HStack`-et egy másik `HStack`-be és adjuk a "belső" `HStack` után egy `Spacer`-t.

> Végezetül a legkülső `HStack`-en állítsuk be a `.listRowSeparator` segítségével a szeőarátor láthatóságát `.hidden`-re.



Szorgalmi feladat

Egészítsük ki az egyedi `ListRowView` nézetünket, hogy ha a saját üzeneteinket jelenítjük meg (egyezik a nevünk a `sender`-rel), akkor balra legyen rendezve az üzenet és legyen zöld a háttér.

> Tipp #1: A leading/trailing irány felcserlését az `.environment(\.layoutDirection,.rightToLeft/.leftToRight)` modifier hívással lehet elérni.

> Tipp #2: Ha feltételhez szeretnénk kötni, hogy egy elem megjelenjen, akkor nyugodtan lehet feltételes szerkezetbe tenni.

```
if message.sender == "Aladár" {
    Spacer(minLength: 40)
}
HStack (alignment: .top){
```

> Tipp #3: Az egyes értékek beállítása szintén lehet feltételvizsgálat eredménye.

```
RoundedRectangle(cornerRadius: 10)
    .fill(message.sender == "Aladár" ? Color.green : Color.blue)
```