Seminář 1: Úvod do iOS vývoje

Tvorba mobilních aplikací

Roman Vyjídáček

Kdo jsem

- Software engineer ve společnosti Enverus
- Absolvent doktorského studia informatiky na UP
- Mám cca 8 let zkušenosti s vývojem pro iOS
- Předchozí projekty:
  - UPlikace
  - Noc vědců

Organizační informace

- Úkoly budu kontrolovat pouze na semináři
- Na vypracování máte čas do následujícího semináře
- Konzultace po domluvě a ideálně online přes MS Teams
- Použití AI vám nezakazuju, ale pokud mi nedokážete vysvětlit řešení, je to automaticky nevypracovaný úkol
- Plagiátorství: V případě shodných řešení bude studentům odebrána možnost získat zápočet

Obsah semináře

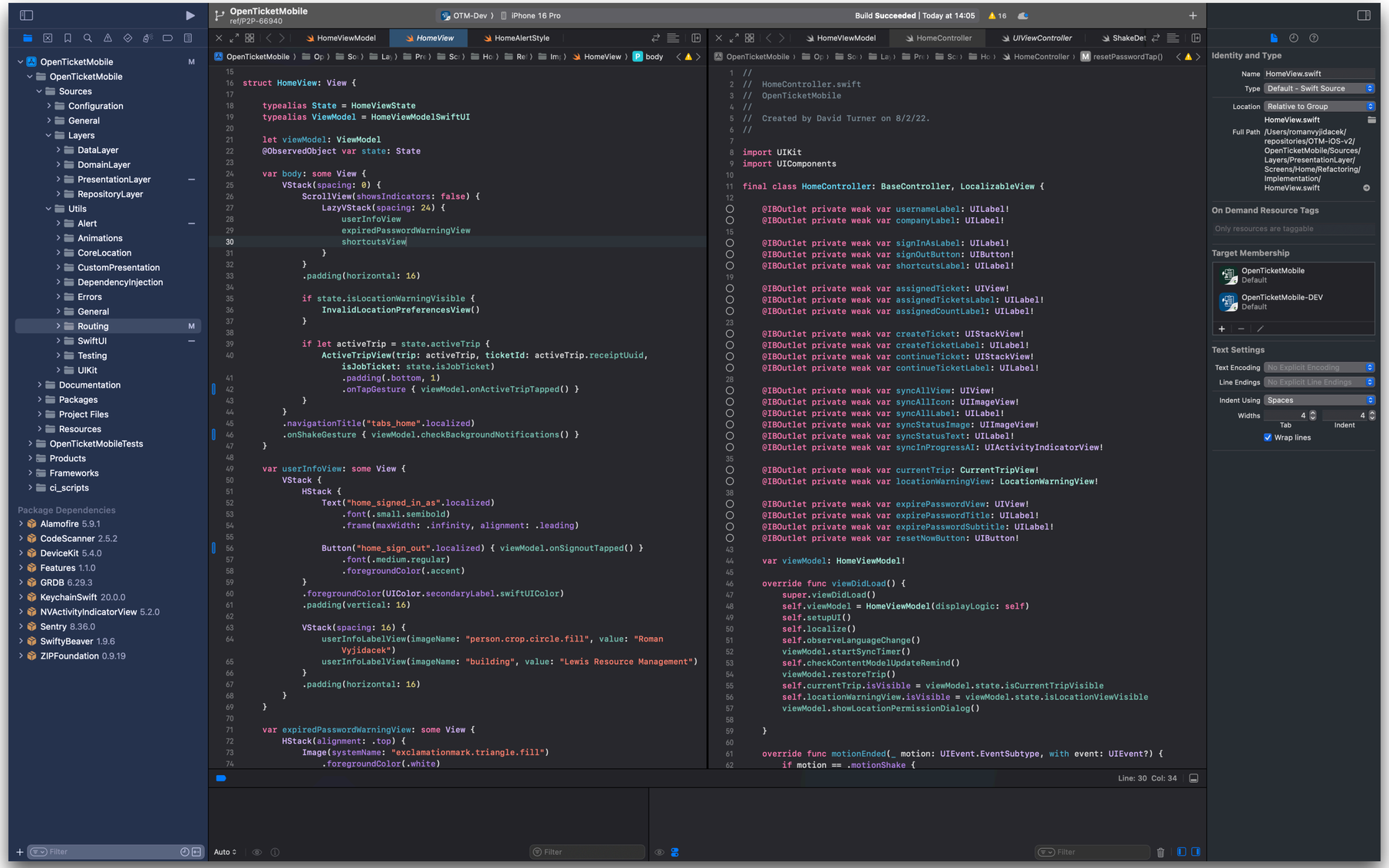
- Vývojové prostředí
- Základy jazyka Swift

Vývojové prostředí

Xcode

- Vývojové prostředí (IDE) pro vývoj aplikací na platformách firmy Apple
- Hlavní funkce zahrnují:
  - Návrh uživatelského rozhraní (UI)
  - Psaní kódu v jazyce Swift nebo Objective-C
  - Ladění aplikací
  - Testování na simulátorech nebo fyzických zařízeních

Xcode

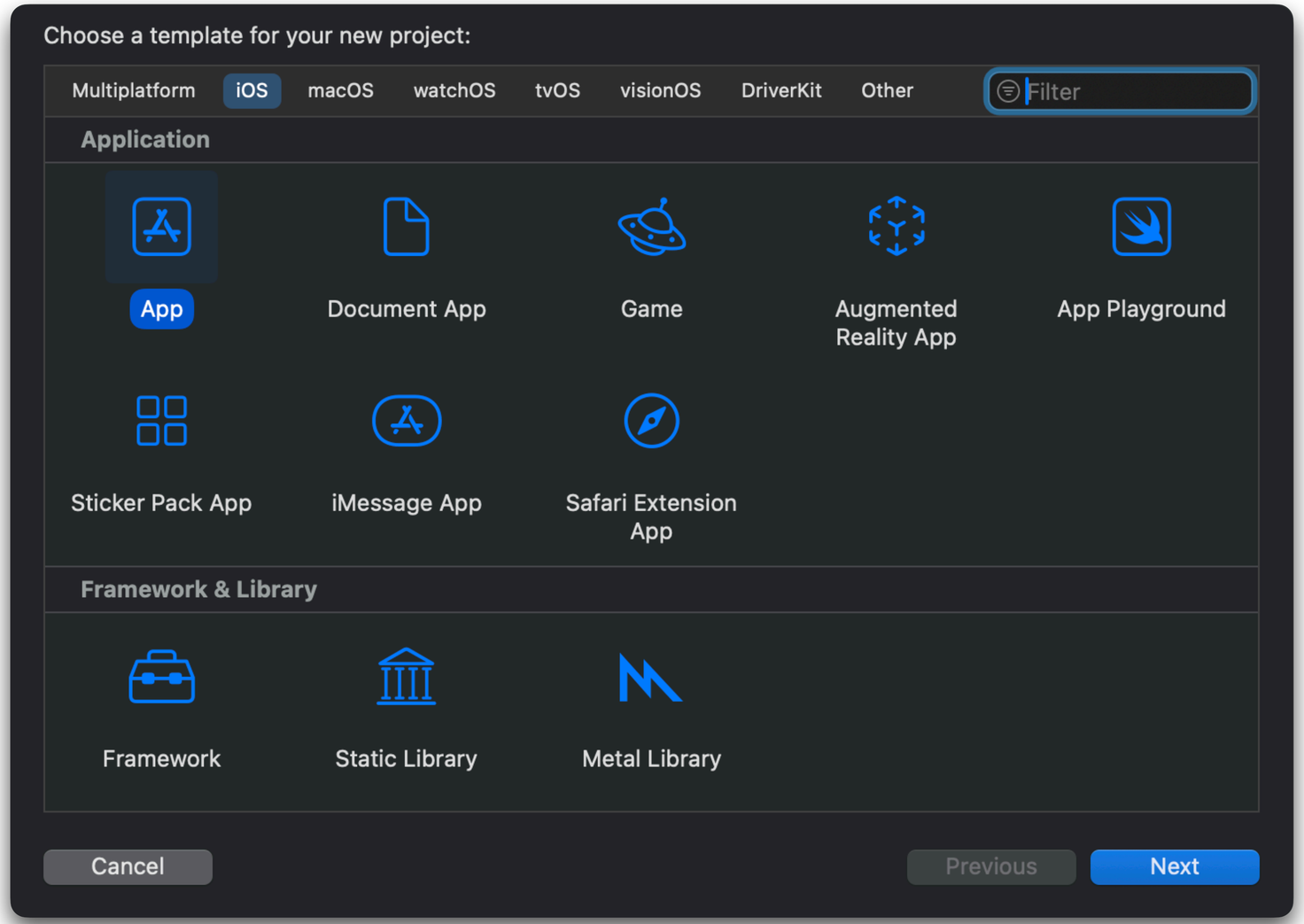

OpenTicketMobile
ref/P2P-66940
OTM-Dev
iPhone 16 Pro
Build Succeeded | Today at 14:05
HomeViewModel
HomeView
HomeAlertStyle
HomeController
UIViewController
Identity and Type
Name HomeView.swift
Type Default - Swift Source
Location Relative to Group
Full Path /Users/romanvyjidacek/repositories/OTM-iOS-v2/OpenTicketMobile/Sources/Layers/PresentationLayer/Screens/Home/Refactoring/Implementation/HomeView.swift
On Demand Resource Tags
Only resources are taggable
Target Membership
OpenTicketMobile
Default
OpenTicketMobile-DEV
Default
Text Settings
Text Encoding No Explicit Encoding
Line Endings No Explicit Line Endings
Indent Using Spaces
Widths 4 4
Tab Indent
Wrap lines

Vytvoření projektu

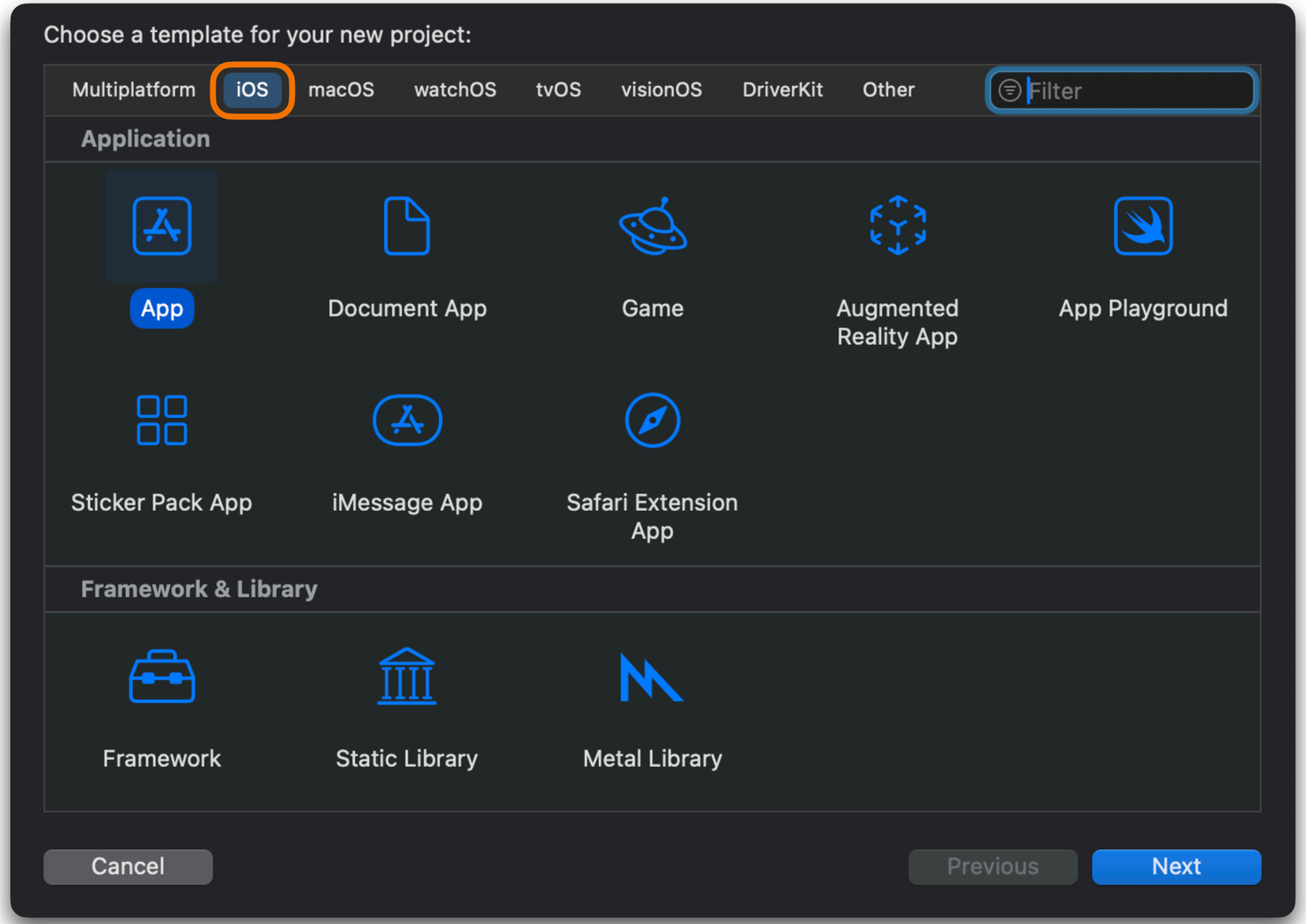
Vytvořit nový projekt

Vytvořit nový projekt

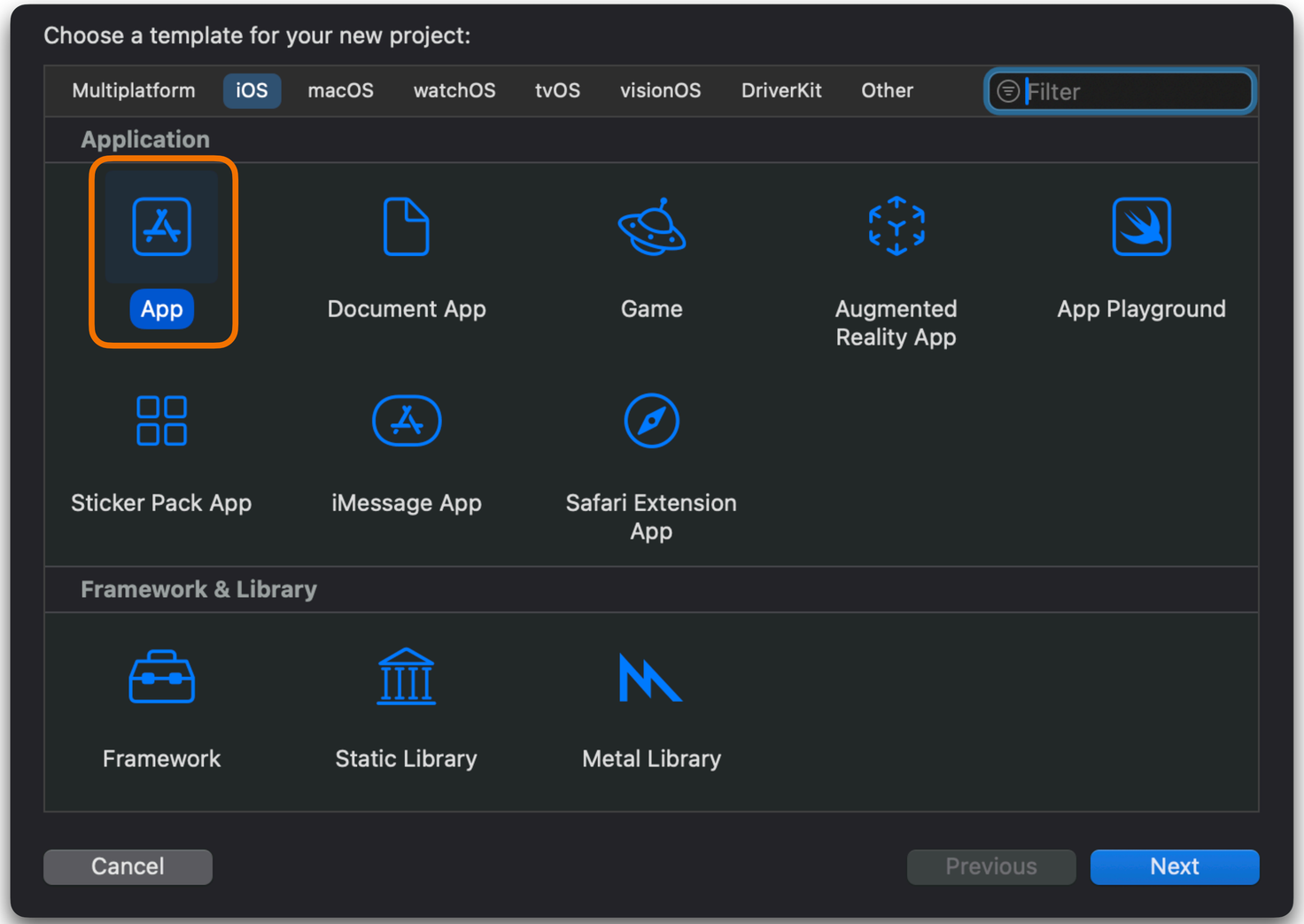
Vybereme typ aplikace



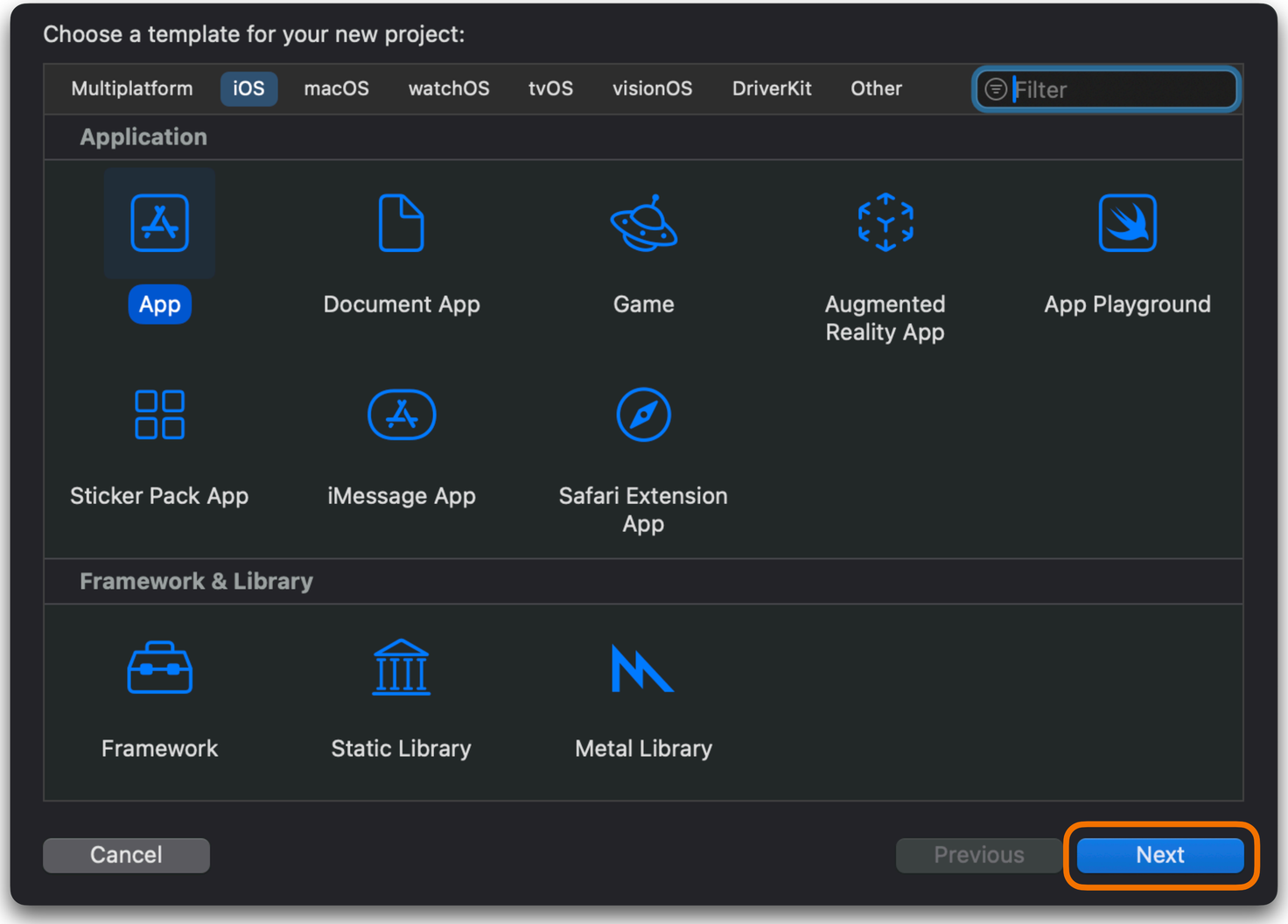
Vybereme typ aplikace



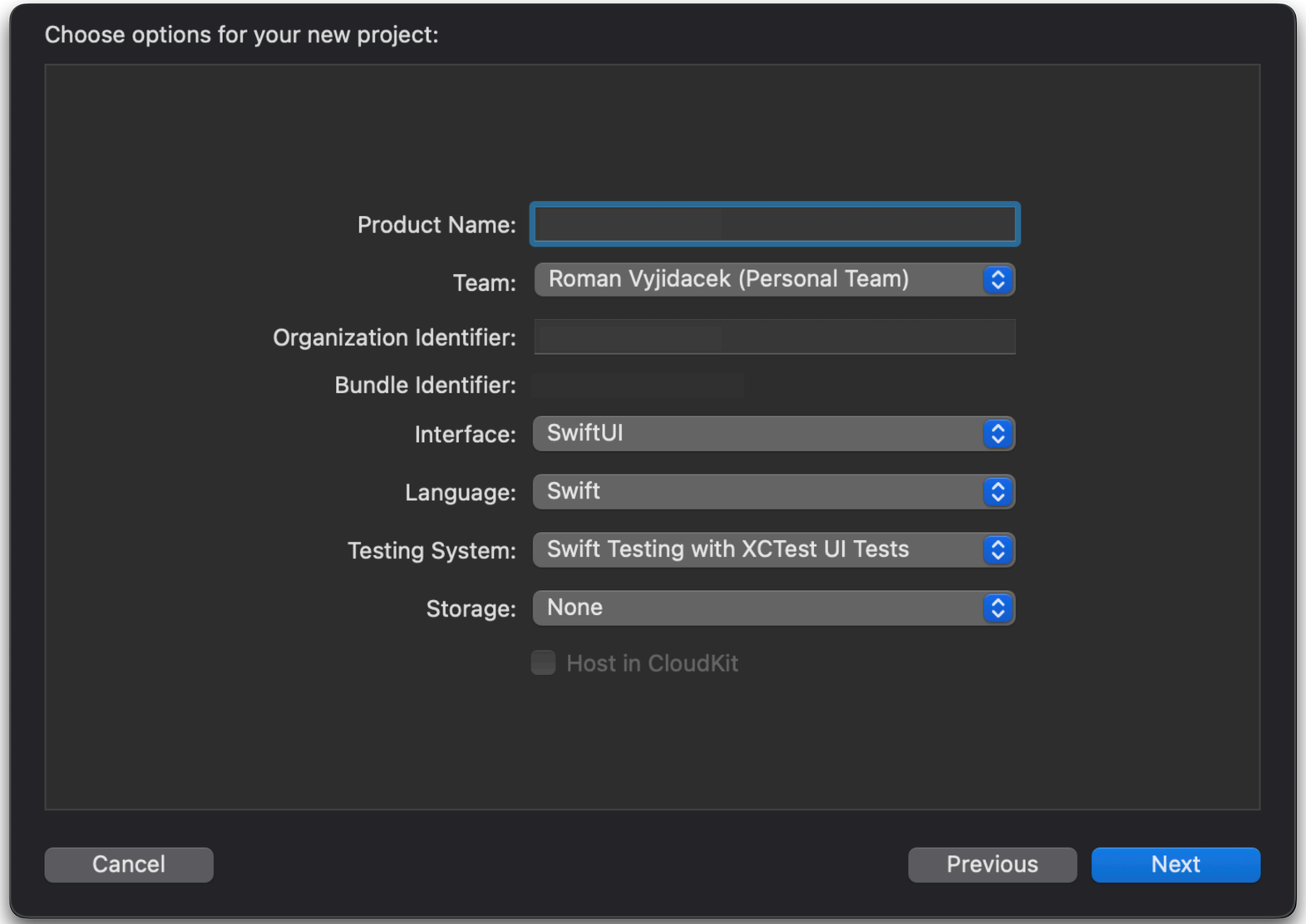
Vybereme typ aplikace



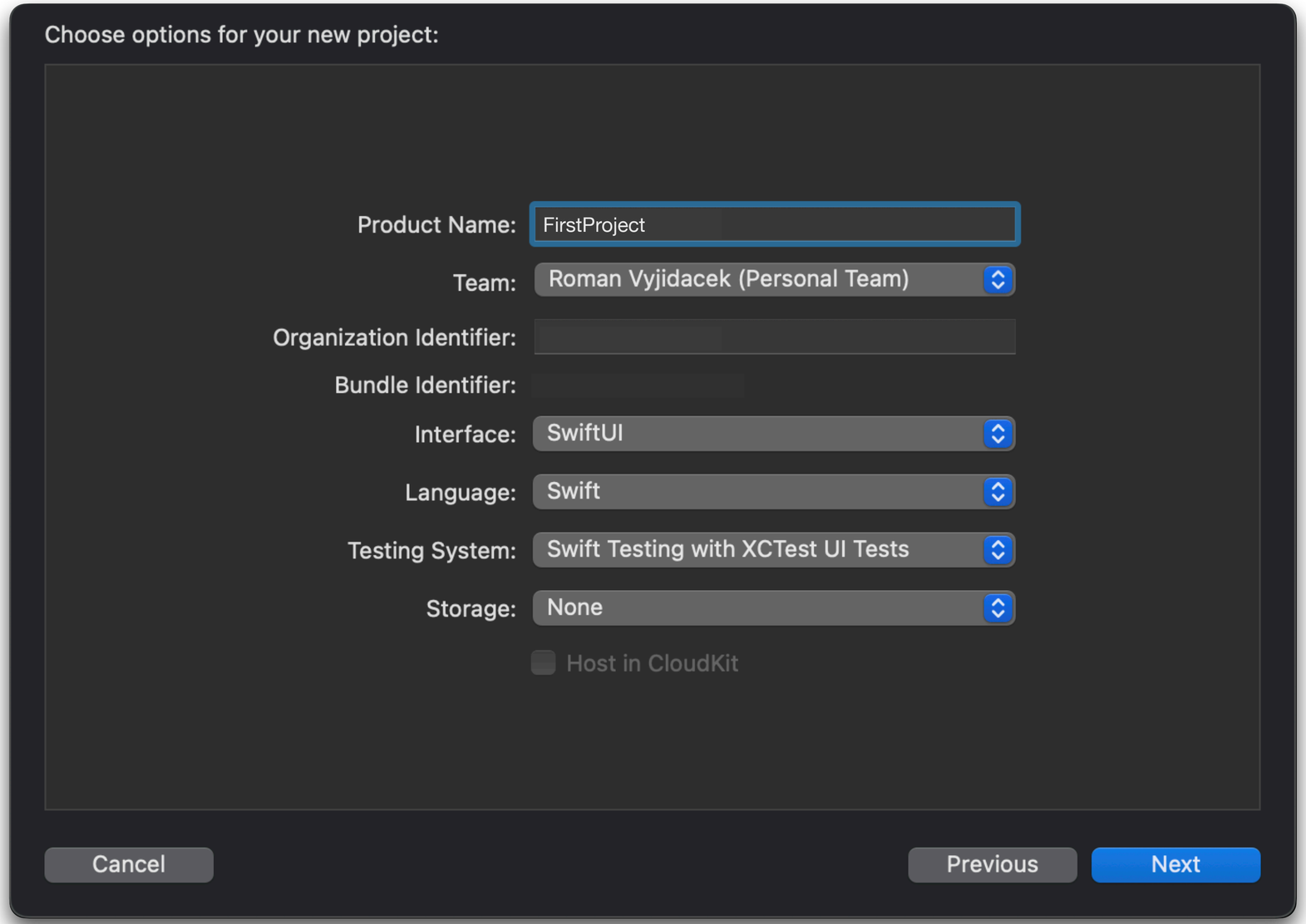
Vybereme typ aplikace



Nastavení projektu



Nastavení projektu



Product Name v Xcode by měl být jedinečný, krátký, bez mezer a speciálních znaků, a začínat velkým písmenem.

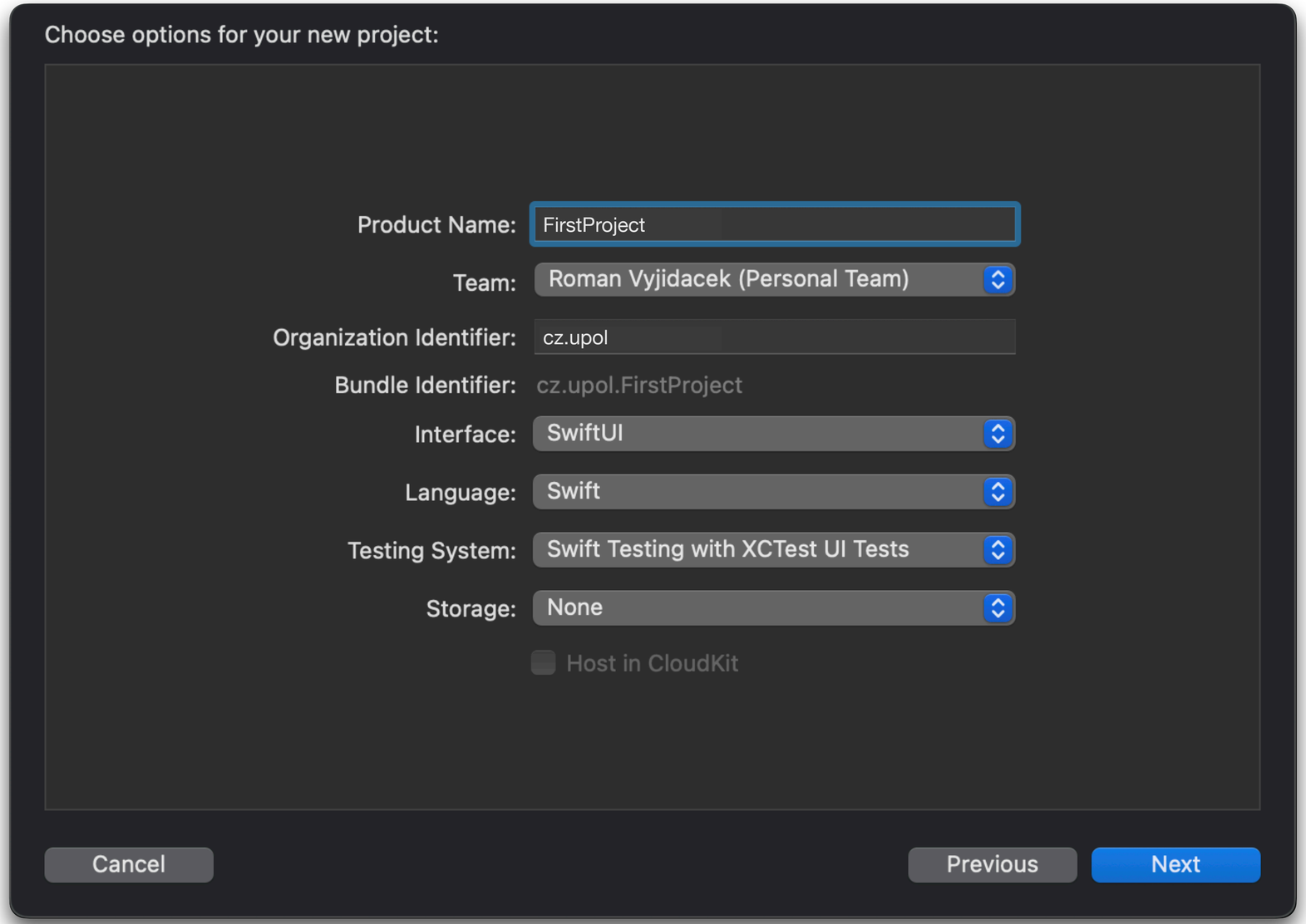
Příklad správných názvů:

✅ WeatherTracker | ✅ ToDoApp

Příklad špatných názvů:

❌ 123MyApp | ❌ My App

Nastavení projektu



Product Name v Xcode by měl být jedinečný, krátký, bez mezer a speciálních znaků, a začínat velkým písmenem.

Příklad správných názvů:

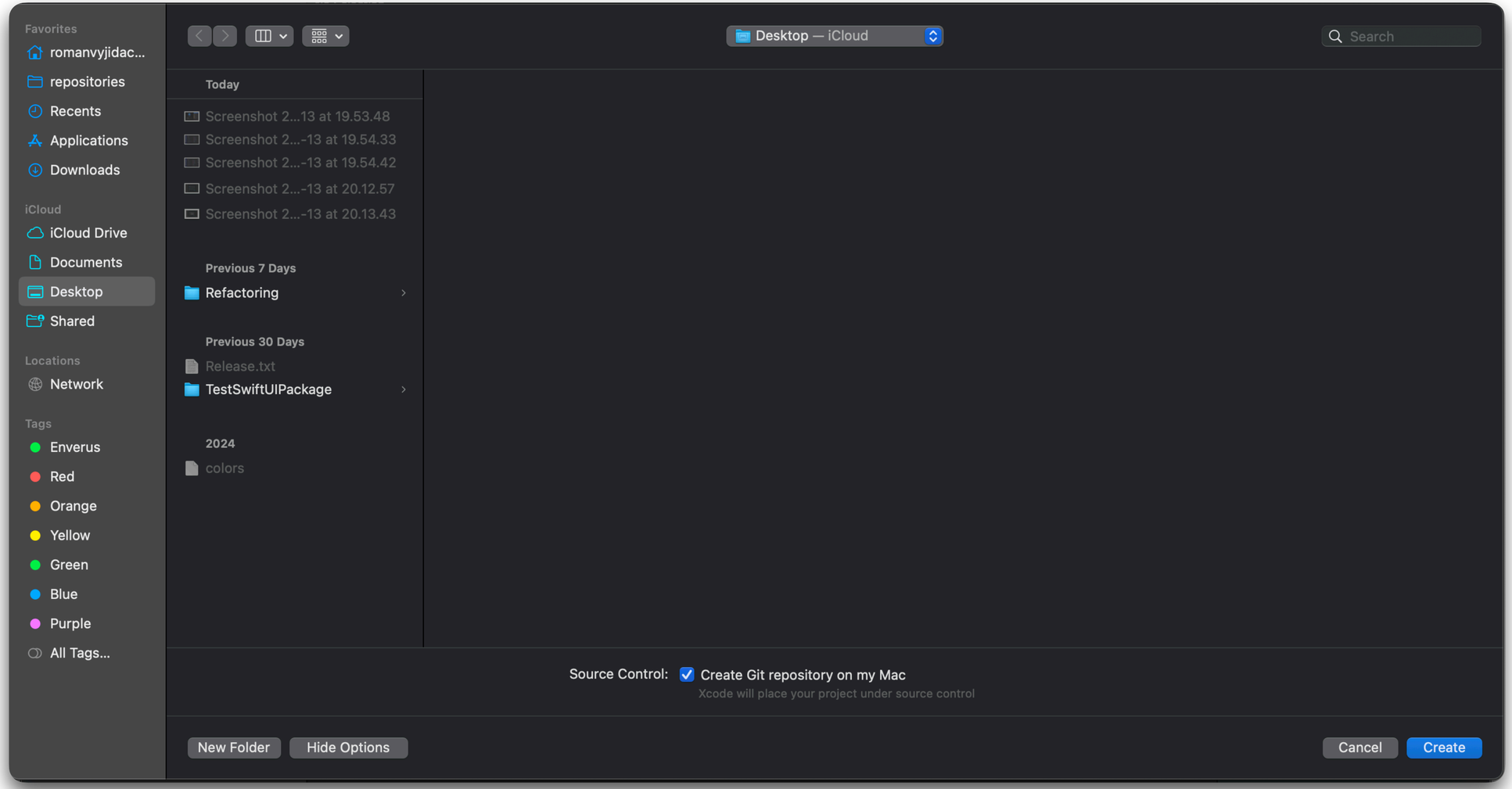
✅ WeatherTracker | ✅ ToDoApp

Příklad špatných názvů:

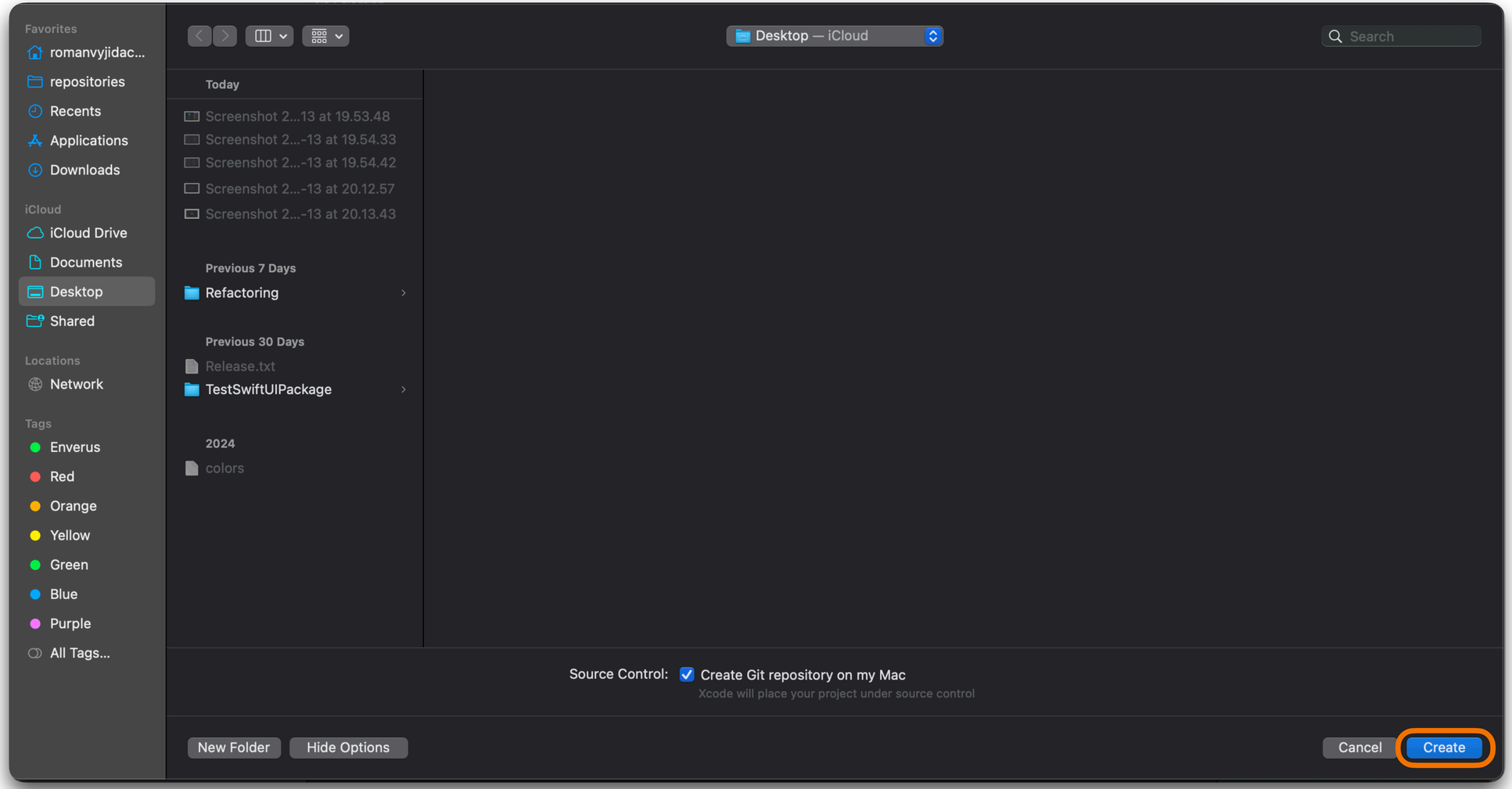
❌ 123MyApp | ❌ My App

Organization Identifier je součástí Bundle Identifier, který jednoznačně identifikuje aplikaci v App Store a na zařízeních. Používá se v reverse domain notation.

Vybereme adresář



Vybereme adresář



Základy jazyka Swift

Swift

- Kompletně nový jazyk
- Uveden 6/2014, vývoj od 2010
- Každý rok nová major verze
- Zpočátku radikální vývoj rozbíjel kompatibilitu od verze 2 open source pod Apache licencí
- Kompletní dokumentace: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/

Balíčkovací systémy

- Swift Package Manager
  - Nativní řešení, získává na oblibě, decentralizovaný
  - Správa přes UI
- Cocoapods
  - Nejrozšířenější správce závislostí
  - Centralizované úložiště, používá **Podfile** (soubor s deklarací závislostí)
- Carthage
  - Lehká a flexibilní alternativa
  - Decentralizovaný přístup, generuje frameworky namísto integrace kódu.

Hello World!

- Začneme klasicky s programem “Hello World!”
- Spuštení programu:
  - Terminál: `swift main.swift`
  - Swift Playground: https://developer.apple.com/swift-playground/
  - Online: https://www.programiz.com/swift/online-compiler/

```
// Vytiskne řetězec + \n
print("Hello, World!")

// Vytiskne pouze řetězec
print("Hello World!",
      terminator: "")
```

Proměnné a konstanty

- `var` definuje proměnnou, jejíž hodnota se může změnit
- `let` definuje konstantu, která se nemůže změnit
- Pro názvy používáme camelCase
- Není nutné používat ;
- Swift používá typovou inferenci
- Swift vyžaduje inicializaci proměnné před prvním použitím

```
var name = “Alice"

let age = 25
```

Datové typy

- Swift má silnou typovou kontrolu.
- Nemůžeme tedy například provádět aritmetické operace mezi `Int` a `Double`
- Hlavní typy: `Int`, `Double`, `Bool`, `String`
- Při definici proměnné můžeme (a je to běžné) i datový typ

```
var number: Int = 10

var price: Double = 99.99

var isSwiftFun: Bool = true

var message: String = “Hi"
```

Podmínky

- Podmínky používají relační operátory: ==, !=, <, >, <=, >=
- Nejsou potřeba závorky (…) kolem podmínky
- Složitější podmínky je možné vytvářet pomocí logických operátorů && (AND) a || (OR)

```
let age = 18
let isCarAvaliable = false

if age >= 18 {
    print("Můžeš řídit.")
} else {
    print("Nemůžeš řídit.")
}

// Složitější podmínka
if age >= 18 && isCarAvaliable {
    print("Můžeš řídit.")
} else {
    print("Nemůžeš řídit.")
}
```

Switch

- Místo zanořených `if-else` bloků, kde pouze vybíráme jednu hodnotu je lepší využít konstrukce `switch`
- Je možné “přepínat” všechny základní datové typy + výčtové datové typy
- `Switch` musí pokrýt všechny možné případy nebo mít default větev
- Zde je prezentován pouze základní výraz → více v dokumentaci

```
let grade = "B"

switch grade {
case "A":
    print("Výborně!")
case "B":
    print("Dobrá práce!")
default:
    print("Zkus to lépe.")
}
```

Cykly (for, while)

- `For-in` iteruje přes sekvence (`Array`, `Range`)
- `While` se opakuje, dokud platí podmínka
- `Repeat-while` nejprve provede blok kódu, poté kontroluje podmínku
- Použijeme `for-in`, pokud víš, kolikrát se má smyčka opakovat. Použijeme `while`, pokud to nevíme předem.

```
// Čísla 1,2,3,4,5
for number in 1...5 {
    print("Číslo: \(number)")
}

// Čísla 1,2,3,4
for number in 1..<5 {
    print("Číslo: \(number)")
}

var count = 3

while count > 0 {
    print(count)
    count -= 1
}
```

Funkce

- `func` definuje funkci
- Pro názvy používáme camelCase
- Parametry mají pevně daný typ
- Funkce mohou mít výchozí hodnoty parametrů
- Pokud chceme předat hodnotový typ odkazem je nutné definovat parametr jako `inout`

```
func greet(name: String) -> String {
    return "Ahoj, \(name)!"
}

greet(name: "Petr")

func sayHello(name: String = "světe") {
    print("Ahoj, \(name)!")
}

sayHello()
sayHello(name: “Jana")

func changeValue(number: inout Int) {
    number = Int.random(in: 0...100)
}

var number = 2
changeValue(number: &number)
```

Kolekce

- Pole je datový typ předávaný hodnotou
- Ve skutečnosti však Swift předává kolekce odkazem a při zápisu vytvoří kopii
- Přes kolekce lze snadno iterovat pomocí `for-in` cyklu
- Slovník ukládá data ve formátu klíč-hodnota

```
var fruits = ["Jablko", “Banán",
              "Hruška"]
// Přístup k prvnímu prvku
fruits[0]
// Přidání prvku
fruits.append("Broskev")
// Odstranění druhého prvku
fruits.remove(at: 1)
// Počet prvků
fruits.count

var person = ["Sheldon": "Dr.",
              "Wolowitz": "Mr."]
// Získáme hodnotu
person["Sheldon"]
// Odstraníme hodnotu
person["Sheldon"] = nil
```

Struktury

- `Struct` je hodnotový typ – při přiřazení nebo předání funkci se kopíruje
- Používá se pro jednoduchá data, jako jsou modely, souřadnice, nastavení atd
- `Mutating` umožňuje měnit vlastnosti struktury uvnitř metody.
- Neumožňuje dědičnost – každý struct je samostatný
- Struktury jsou thread-safe

```
struct Point {
    var x: Int
    var y: Int

    mutating func moveBy(x: Int, y: Int) {
        self.x += x
        self.y += y
    }
}

var p = Point(x: 3, y: 4)
p.moveBy(x: 2, y: -1)
print(p) // Point(x: 5, y: 3)
```

Třídy

- `Class` je referenční typ – při přiřazení se nepřekopíruje, ale předává se reference.
- Umožňuje dědičnost (inheritance) – může mít nadřazenou třídu.
- Změny v objektu se projeví všude, kde je reference na instanci.

```
class Animal {
    var name: String

    init(name: String) {
        self.name = name
    }

    func makeSound() {
        print("Nějaký zvuk")
    }
}

class Dog: Animal {
    override func makeSound() {
        print("Haf haf!")
    }
}

let dog = Dog(name: "Rex")
dog.makeSound() // Haf haf!
```

Shrnutí struct vs. class

Vlastnost	struct (Struktura)	class (Třída)
Typ	Hodnotový (Value Type)	Referenční (Reference Type)
Kopírování	Vytvoří novou kopii	Předává referenci
Dědičnost	❌ Nepodporuje	✅ Podporuje
Změna vlastností	Je potřeba `mutating`	Není potřeba `mutating`
Thread-safe	Ano – každá instance je unikátní	Možné problémy – reference mohou být sdílené

Computed properties

- Používáme pro hodnoty, které můžeme vypočítat z ostatních vlastností
- `var area` není ukládána do paměti, ale vypočítává se při každém přístupu
- Může mít také set blok pro aktualizaci hodnoty
- Computed properties používáme místo funkcí, pokud se hodnota logicky vztahuje k objektu.

```
class Rectangle {
    var width: Double
    var height: Double

    var area: Double {
        return width * height
    }

    var perimeter: Double {
        get { return 2 * (width + height) }
        set { width = newValue / 4
              height = newValue / 4 }
    }

    init(width: Double, height: Double) {
        self.width = width
        self.height = height
    }
}

let rect = Rectangle(width: 5, height: 10)
print(rect.area) // 50.0
```

Optionals

- Umožňují proměnným nemít žádnou hodnotu (`nil`)
- Swift neumožňuje nil v běžných proměnných – musí být Optional
- Pokud chceme hodnotu “rozbalit” tak použijeme !
- Nikdy nepoužívejte ! bez kontroly, zda hodnota existuje!
- Pomocí operátoru ?? Můžeme definovat výchozí hodnotu pokud je hodnota `nil`

```swift
var name: String? = nil
name = "Alice"
print(name) // Optional("Alice")

// Alice (ale může způsobit
crash, pokud je nil!)
print(name!)

var username: String? = nil
let user = username ?? "Vader"
print(user) // "Vader"
```

Bezpečné rozbalení Optional

- `If let` bezpečně rozbalí Optional a použije jeho hodnotu
- Pokud je hodnota `nil`, provede se else (je-li přítomna) blok
- `Guard let` Slouží ke kontrole podmínek na začátku funkce
- Pokud podmínka není splněna, kód okamžitě opustí funkci (return)

```swift
var name: String? = "Alice"

if let unwrappedName = name {
    print("Ahoj, \(unwrappedName)")
} else {
    print("Žádné jméno")
}

func greet(name: String?) {
    guard let unwrappedName = name else {
        print("Jméno chybí!")
        return
    }
    print("Ahoj, \(unwrappedName)")
}
```

Výčtové typy

- Výčtový typ (enum) umožňuje definovat sadu souvisejících hodno
- Každá hodnota (case) je unikátní a typově bezpečná
- Enum může obsahovat asociované hodnoty a výchozí hodnoty (raw values)
- V enum je možné definovat metody
- Samostudium: rawValues, switch pro výčtové typy

```
enum Direction {
    case north
    case south
    case east
    case west
}

let move = Direction.north
var current: Direction = .east

enum Barcode {
    case upc(Int, Int, Int, Int)
    case qrCode(String)
}

let upc = Barcode.upc(8,85909,51226,3)
let qr = Barcode.qrCode("ABC123XYZ")
```

Úkoly

https://vyjidacek.cz/tmai/seminar1.php