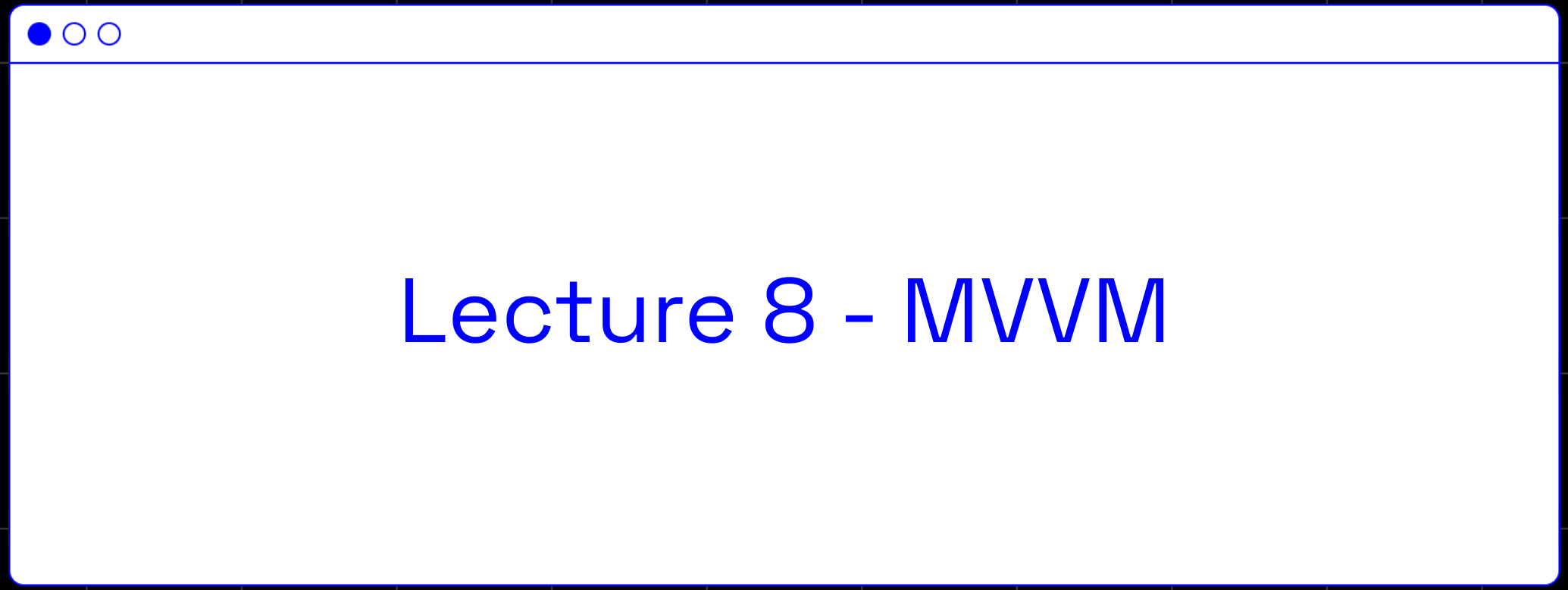
ackee edu

BI-IOS

Igor Rosocha, Jakub Olejník, Rostislav Babáček


Lecture 8 - MVVM

Architektura aplikace

> Softwarová architektura popisuje rozdělení aplikace na komponenty a vztahy mezi nimi.

⚠️ Klíčové rozhodnutí v počáteční fázi

Příklady

- MVC (Model-View-Controller)
- **MVVM (Model-View-ViewModel)**
- TCA (The Composable Architecture)
- Clean Architecture

MVVM *(Model-View-ViewModel)*

Jednoduchá a flexibilní architektura

Rozdělení kódu tak, aby `View` *nedělalo všechno*

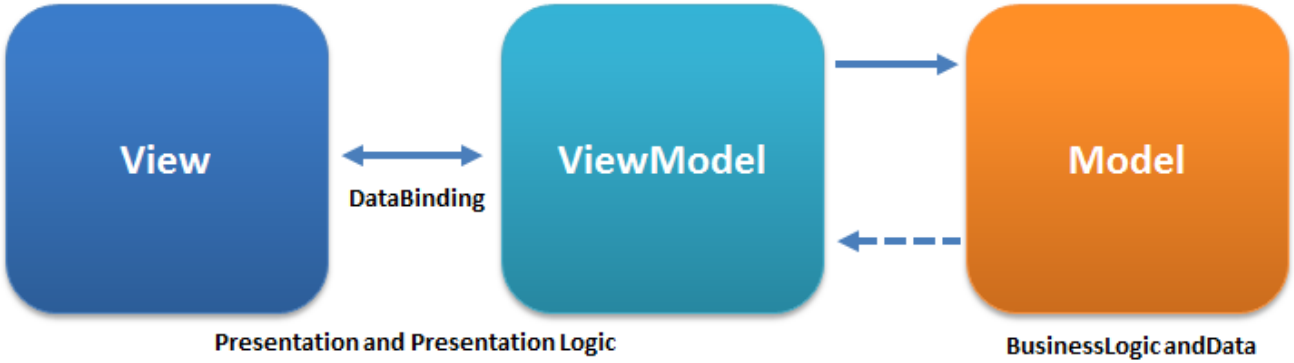
Zlepšuje/umožňuje znovupoužitelnost

Odstiňuje `View` od toho, jak se získávají data

Zvyšuje testovatelnost

👎🏽 State Management může být tricky se SwiftUI

MVVM



`Model` - logika + perzistence, networking apod.

`ViewModel` - příprava dat pro `View` + propagace akcí do modelu

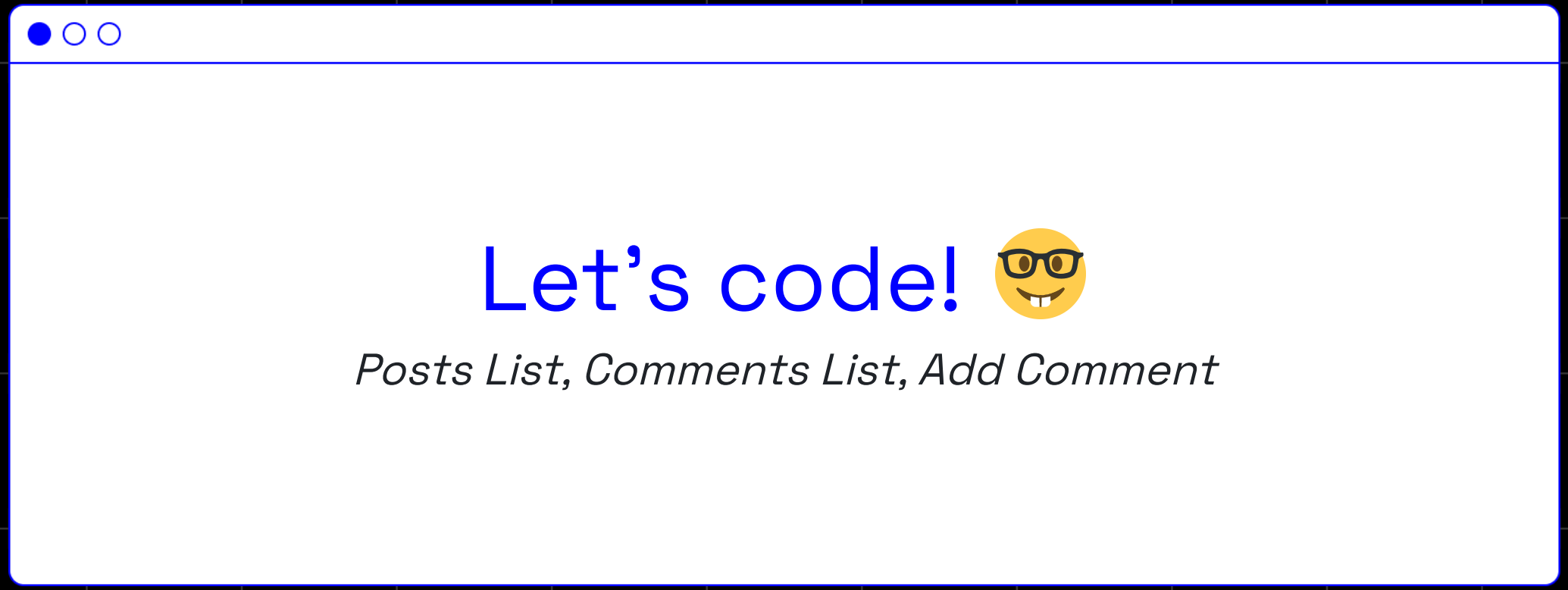
`View` - zobrazení dat z VM + předávání UI interakcí do VM

ViewModel

Obecně může mít vlastní modelové objekty, na které bude konvertovat objekty z modelu

V našem případě bude zastupovat i roli modelu

Přijímá z view akce typu ”přidej komentář” a překládá je na akce typu ”pošli tenhle request na tenhle endpoint a s odpovědí udělej tohle”


Let’s code! 🤓
Posts List, Comments List, Add Comment

Otázky ?

UI updates iOS 13+

- ❏ `ObservableObject`
- ❏ `@StateObject` *(View created the object)*
- ❏ `@ObservedObject` *(View is just watching, passed by reference)*
- ❏ `objectWillChange.send()` x `@Published`

⚠️ Musíme označit co má vyvolat překreslení View

```
class CommentViewModel: ObservableObject {
    @Published var user: User = ...
    @Published var text: String = ...
    @Published var date: Date = ...
    var likes: [User] = ...

    func addLike(like: Like) {
        objectWillChange.send()
        likes.append(like)
    }
}

struct CommentView: View {
    @StateObject var viewModel: CommentViewModel
}
// OR
struct CommentView: View {
    @ObservedObject var viewModel: CommentViewModel
}
```

> Important
> Even for a configurable state object, you still declare it as private. This ensures that you can't accidentally set the parameter through a memberwise initializer of the view, because doing so can conflict with the framework's storage management and produce unexpected results.

> Add the @ObservedObject attribute to a parameter of a SwiftUI View when the input is an ObservableObject and you want the view to update when the object's published properties change. You typically do this to pass a StateObject into a subview.

```
class CommentViewModel: ObservableObject {
    @Published var user: User = ...
    @Published var text: String = ...
    @Published var date: Date = ...
    @Published var likes: [User] = ...
} // @Published, @Published, @Published, @Published, @Published 🤯

@StateObject var viewModel: CommentViewModel // or @ObservableObject? 🤯
```

```
@Observable class CommentViewModel {
    var user: User = ...
    var description: String = ...
    var date: Date = ...
    var likes: [User] = ...
}

var viewModel: CommentViewModel
```

UI updates iOS 17+

- ❏ `@Observable`
- ❏ `@ObservationIgnored`
- ❏ `@Bindable`

❗ By default **vše co k čemu se přistupuje** vyvolává překreslení View

```
import Observation
```

Migrating from the Observable Object protocol to the Observable macro

> Note
> If you need to store a value type, like a structure, string, or integer, use the State property wrapper instead. Also use State if you need to store a reference type that conforms to the Observable() protocol.

Otázky ?

Thank you very much,
see you next week! 🙌