

## Codable

Protokol + code gen sloužící ke zakódování / dekódování objektů do / z dat

Pokud je API hezké, výsledek je snadný a zadarmo

Několik možností, jak nasadit na "ošklivé" API

typealias Codable = Decodable & Encodable

### Decodable

```
{
    "color": "red",
    "value": "#f00"
}
```

```
struct Color: Decodable {
   let color: String
   let value: String
}
```

#### CodingKeys

```
{
    "color_name": "red",
    "value": "#f00"
}
```

```
struct Color: Decodable {
    enum CodingKeys: String, CodingKey {
        case name = "color_name"
        case value
    }
    let name: String
    let value: String
}
```

#### Vlastní init

Pro velmi ošklivá API

```
struct Color: Decodable {
    enum CodingKeys: String, CodingKey {
        case name = "color_name"
    let name: String
    init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        self.name = try container.decode(String.self, forKey: .name)
```

#### JSONDecoder

Zodpovědný za převedení JSONu do objektu

```
struct Model: Decodable { }
let decoder = try JSONDecoder().decode(Model.self, from: data)
```

### Encodable

Zakódování objektu do dat (opačný směr než Decodable )

Opět se používají CodingKeys pro parametry, které jsou různě pojmenované

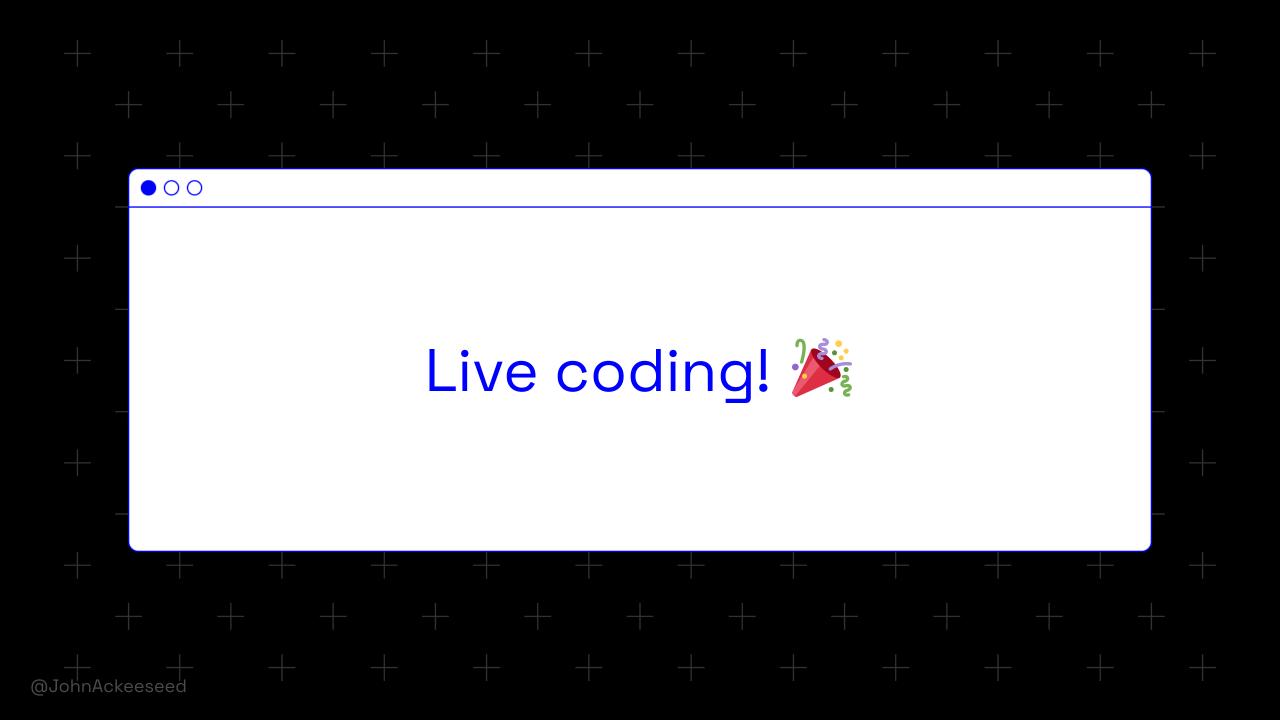
#### Encodable

```
struct Color: Encodable {
    enum CodingKeys: String, CodingKey {
        case name = "color_name"
    let name: String
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(name, forKey: .name)
```

```
struct Color: Codable {
    enum CodingKeys: String, CodingKey {
        case name = "color_name"
    let name: String
    init(from decoder: Decoder) throws {
        let container = decoder.container(keyedBy: CodingKeys.self)
        name = try container.decode(String.self, forKey: .name)
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(name, forKey: .name)
```

# Codable

https://www.raywenderlich.com/3418439-encoding-and-decoding-in-swift https://www.hackingwithswift.com/articles/119/codable-cheat-sheet



# Async/await

Nový přístup k psaní asynchronního kódu

Řeší spoustu věcí za programátora

Jednoduchý nativní styl psaní

```
let request = URLRequest(...)
// Old way
let task = URLSession.shared.dataTask(with: request) { data, response, error in
    if let data {
        do {
            let model = try JSONDecoder().decode(Model.self, from: data)
        } catch {
            print(error)
task.resume()
// New way
do {
    let (data, response) = try await URLSession.shared.data(for: request)
    let model = try JSONDecoder().decode(Model.self, from: data)
} catch {
    print(error)
```

### Async funkce

```
func methodName() async -> ReturnType {
    ...
}
```

Klíčové slovo async za argumenty

Potřeba zavolat z async kontextu, nelze volat přímo



Vytvoření asynchronního kontextu = naplánování async funkce

Není potřeba na něj držet referenci, systém ho odklidí když skončí

# Actor

Koncepčně podobné ako class

Atomický prístup

Bezpečné použití v konkuretních prostředích

Není potřeba řešit race-conditions/locks

## @MainActor

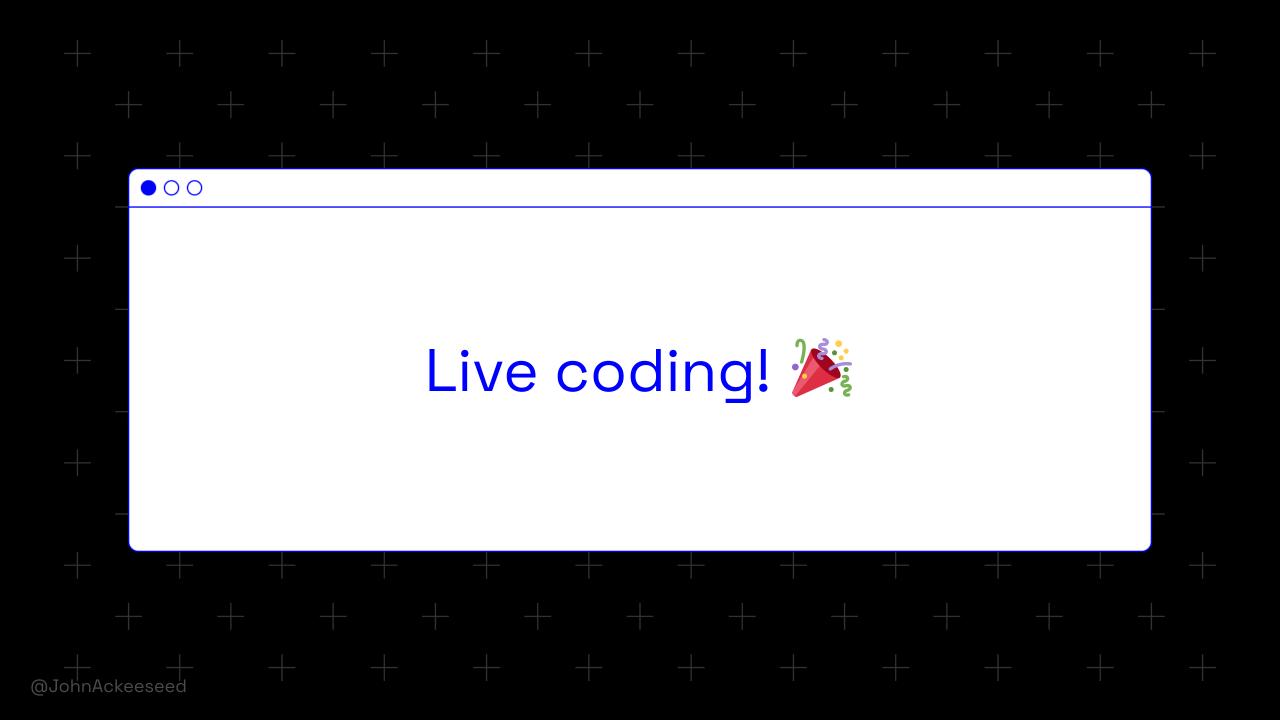
Označení části kódu, která bude vždy spuštěna / použita na hlavním vlákně

Můžeme označit jak funkce, tak objekty

Opět všechno za nás na pozadí vyřešeno

# Concurrency

https://www.hackingwithswift.com/quick-start/concurrency



#### 1. domácí úkol

Udělat detail postu

- 🖵 může mít více fotek ( 💡 PageView )
- nějaké hezké Ul (fotky jsou vidět celé)

Načtení a zobrazení komentářů včetně nějakého UI (třeba nějak přes fotku)

Zobrazení informací o uživateli / postu s nějakým pěkným Ul

Bonus: Po tapnutí na nějaké tlačítko se skryje všechno kromě fotek

