Seminář 5: Síťové operace

Tvorba mobilních aplikací

Roman Vyjídáček

Obsah semináře

- Serializace a Deserializace dat
- Práce s API

Serializace a Deserializace dat

Proč nás zajímá serializace dat?

- Nástroj pro mapování JSON dat na modely a zpět.
- Použití Plist, Data, UserDefaults.
- Bezpečné & typově kontrolované: compiler hlídá, že vše, co uložíme, umíme i načíst.

Protokol	Účel	Směr
Encodable	Převedeme Swift hodnotu do externí podoby	Swift → Data/JSON
Decodable	Převede externí data do Swift hodnoty	Data/JSON → Swift
Codable	Typealias pro Encodable & Decodable	Obousměrný

Základní použití

- Automaticky se mapují názvy vlastností ↔ JSON klíče.
- Získáme Data, které můžeme poslat na API nebo uložit do disku.

```
struct User: Codable {
    let id: Int
    let name: String
}

let user  = User(id: 1, name: "Alice")
let data  = try JSONEncoder().encode(user)
let clone = try JSONDecoder().decode(User.self,
                                     from: data)
```

Vlastní mapování klíčů

- CodingKeys = jemné doladění názvů nebo ignorování polí.
- Strategie pro Date, Data, keyDecodingStrategy(.iso8601)

```
struct Article: Codable {
    let title: String
    let publishedAt: Date
    let teaser: String?

    enum CodingKeys: String, CodingKey {
        case title
        case publishedAt = "published_at"
        case teaser = "short_description"
    }
}

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .iso8601
let article = try decoder.decode(
    Article.self, from: jsonData
)
```

Komplexní JSON

Scénář

- API vrací vnořený objekt user a časový údaj v milisekundách od 1970.
- V aplikaci ale chceme mít plošší model s fullName a klasickým Date.

```
{
  "id": 123,
  "user": {
    "first_name": "Alice",
    "last_name":  "Walker"
  },
  "created": 1713878245000
}
```

```
struct Ticket: Codable {
    let id:       Int
    let fullName: String
    let created:  Date
}
```

Vlastní decoding

```swift
init(from decoder: Decoder) throws {
    // ROOT { … }
    let container = try decoder.container(keyedBy: CodingKeys.self)
    id = try container.decode(Int.self, forKey: .id)

    // "user": { first_name, last_name }
    let user = try container.nestedContainer(keyedBy: UserKeys.self,
                                             forKey: .user)
    let first = try user.decode(String.self, forKey: .firstName)
    let last  = try user.decode(String.self, forKey: .lastName)
    fullName = "\(first) \(last)"

    // "created": epoch-time v milisekundách
    let millis = try container.decode(Double.self, forKey: .created)
    created = Date(timeIntervalSince1970: millis / 1_000)
}

private enum CodingKeys: String, CodingKey { case id, user, created }
private enum UserKeys:  String, CodingKey {
    case firstName = "first_name"
    case lastName  = "last_name"
}
```

Vlastní encoding

```swift
func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)
    try container.encode(id, forKey: .id)

    // rozbije fullName na křestní + příjmení
    let parts = fullName.split(separator: " ", maxSplits: 1)
    let first = parts.first.map(String.init) ?? ""
    let last  = parts.dropFirst().first.map(String.init) ?? ""

    var user = container.nestedContainer(keyedBy: UserKeys.self,
                                         forKey: .user)
    try user.encode(first, forKey: .firstName)
    try user.encode(last,  forKey: .lastName)

    // Date -> epoch-ms
    let millis = created.timeIntervalSince1970 * 1_000
    try container.encode(millis, forKey: .created)
}

private enum CodingKeys: String, CodingKey { case id, user, created }
private enum UserKeys:  String, CodingKey {
    case firstName = "first_name"
    case lastName  = "last_name"
}
```

Optionals

```
struct Person: Codable {
    let id: Int
    let name: String
    var nickname: String?

    enum CodingKeys: String, CodingKey { case id, name, nickname }

    init(from decoder: Decoder) throws {
        let c = try decoder.container(keyedBy: CodingKeys.self)
        id = try c.decode(Int.self,    forKey: .id)
        name = try c.decode(String.self, forKey: .name)
        nickname = try c.decodeIfPresent(String.self, forKey: .nickname) ?? “(bez přezdívky)”
    }

    // Kódování: „nickname“ zapíšeme jen tehdy, pokud není nil
    func encode(to encoder: Encoder) throws {
        var c = encoder.container(keyedBy: CodingKeys.self)
        try c.encode(id, forKey: .id)
        try c.encode(name, forKey: .name)
        try c.encodeIfPresent(nickname, forKey: .nickname)
    }
}
```

Práce s API

Základy HTTP požadavků

- URL = adresa + volitelné query parametry (např. ?page=2&limit=20).
- Metoda určuje záměr klienta: nejčastěji GET (čtení) nebo POST (zápis).
- Hlavičky (headers) nesou metadata: Content-Type, Accept, Authorization, User-Agent…
- Tělo (body) je nepovinná datová nálož – JSON, multipart, formulář…
- Odpověď obsahuje status kód (200 OK, 404 Not Found…), hlavičky a případné tělo s daty nebo chybou.

GET požadavek

- Slouží výhradně k načítání zdrojů – „dej mi data“.
- Nemá měnit stav serveru → je bezpečný a idempotentní (opakované volání = stejný výsledek, žádný vedlejší efekt).
- Data se posílají jen v URL (cesta + query). Tělo se oficiálně ignoruje.
- Může být kešovatelný (CDN, prohlížeč, URLCache).
- Omezen velikostí URL (řádově kilobajty), takže není vhodný pro dlouhé parametry či citlivá data.

POST požadavek

- Slouží k vytváření nebo modifikaci zdrojů – „pošli data“.
- Tělo je hlavní nositel obsahu: JSON (application/json), formulář (application/x-www-form-urlencoded), soubor (multipart).
- Není idempotentní – dvě identická volání mohou vytvořit dva záznamy.
- Nekešuje se (případné opakování řeší klient – např. retry).
- Často vyžaduje autentizaci (token v hlavičce Authorization) a zabezpečení přes HTTPS.

Praktické zásady pro iOS vývojáře

- Model URL: cestu držte čistou (/users/123) a filtry v query (?sort=name).
- Validate status kódy: 2xx = úspěch, 4xx = chyba klienta, 5xx = chyba serveru.
- Timeouty a retry – mobilní síť není spolehlivá; myslete na exponential back-off.
- JSON only: sjednoťte Content-Type i Accept na application/json a svá data vždy serializujte přes Codable.
- Bezpečnost: vždy používejte HTTPS, nikdy neukládejte citlivé tokeny do kódu; pro OAuth apod. využijte Keychain.

Vytvoření URL

- Určete základní URL (https://api.example.com) a cestu (např. /articles).
- Rozmyslete si query parametry; ty přidáte do URLComponents.

```
var comps = URLComponents(
    string: "https://api.example.com/articles"
)!

comps.queryItems = [
    URLQueryItem(name: "page", value: "1")
]

let url = comps.url!
// => https://api.example.com/articles?page=1
```

Sestavení URLRequest

- Vyplňte HTTP metodu (GET, POST, …).
- Přidejte hlavičky: Content-Type, Accept, případně Authorization.
- Pokud posíláte data, zakódujte je do Data (typicky JSON) a přiřaďte do httpBody.

```
var request = URLRequest(url: url)
request.httpMethod = "POST"
request.setValue(
    "application/json",
    forHTTPHeaderField: "Content-Type"
)

request.httpBody = try JSONEncoder().encode(article)
request.timeoutInterval = 15
```

Síťové volání

- Použijte URLSession.shared.data(for:request) uvnitř async funkce.
- Získejte data a URLResponse.
- Přetypujte odpověď na HTTPURLResponse.
- Zpracujte potencionální chyby.

```
let (data, response) = try await
    URLSession.shared.data(for: request)

guard let http = response as? HTTPURLResponse
//    Neznama chyba
}

// zpracujeme jednotlive chybove stavy
```

Authorizace s heslem

```
func makeBasicAuthRequest(url: URL,
                          username: String,
                          password: String) throws -> URLRequest {

    // 1. username:password → String
    let credential = "\(username):\(password)"

    // 2. UTF-8 → Data  •  3. Base-64
    guard let credentialData = credential.data(using: .utf8) else {
        throw URLError(.badURL)
    }
    let base64Credential = credentialData.base64EncodedString()

    // 4. URLRequest + hlavička Authorization
    var request = URLRequest(url: url)
    request.httpMethod = "GET"
    request.setValue("Basic \(base64Credential)",
                     forHTTPHeaderField: "Authorization")
    request.setValue("application/json",
                     forHTTPHeaderField: "Accept")
    return request
}
```

Authorizace

```swift
func makeAuthorizedRequest(_ url: URL,
                           method: String = "GET",
                           jsonBody: Encodable? = nil) async throws -> URLRequest {

    var req = URLRequest(url: url)
    req.httpMethod = method
    req.setValue("application/json", forHTTPHeaderField: "Accept")

    if let body = jsonBody {
        req.setValue("application/json", forHTTPHeaderField: "Content-Type")
        req.httpBody = try JSONEncoder().encode(body)
    }

    if let token = await AuthState.shared.accessToken {
        req.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
    }
    return req
}
```