Main Actor:

Original:

func populate() {

Webservice().getAllTodos(url: URL(string: "")) { result in

DispatchQueue.main.async {

self.todos = todos.map(TodoViewModel.init)

}

}

APPROACH 1:

@MainActor

class TodoListViewModel: ObservableObject {

@Published var todos: [TodoViewModel] = []

func populateTodos() {

Webservice().getAllTodosAsync(url: URL(string: "https:// ")) { result in

Task {

await MainActor.run {

self.todos = todos.map(TodoViewModel.init)

}

}

}

APPROACH 2: Adding main actor on closure:

func getAllTodos(url: URL, completion: **@MainActor** @escaping (Result<[Todo], NetworkError>) -> Void) {

URLSession.shared.dataTask(with: url) { data, \_, error in

guard let data = data, error == nil else { return }

let todos = try? JSONDecoder().decode([Todo].self, from: data)

Task {

await completion(.success(todos))

}

}

}.resume()

}

func populateTodos() {

Webservice().getAllTodosAsync(url: URL(string: "https:// ")) { result in

switch result {

case .success(let todos):

todos.map(TodoViewModel.init)

} } }

Approach 3: Recommended: Update closure callback with async await:

|  |  |
| --- | --- |
| Closure callback | Async await |
| func getAllTodos(url: URL, completion: @escaping (Result<[Todo], NetworkError>) -> Void) {  URLSession.shared.dataTask(with: url) { data, \_, error in  guard let data = data, error == nil else {  completion(.failure(.badRequest))  }  guard let todos = try? JSONDecoder().decode([Todo].self, from: data) else {  completion(.failure(.decodingError))  }  completion(.success(todos))  }.resume()  } | func getAllTodosAsync(url: URL) async throws -> [Todo] {  let (data, \_) = try await URLSession.shared.data(from: url)  let todos = try? JSONDecoder().decode([Todo].self, from: data)  return todos ?? []  } |
| func populate() {  Webservice().getAllTodos(url: URL(string: "")) { result in  DispatchQueue.main.async {  self.todos = todos.map(TodoViewModel.init)  }  } | func populateTodos() async {  do {  let todos = try await Webservice().getAllTodosAsync(url: url)  self.todos = todos.map(TodoViewModel.init)  } catch {  print(error)  }} |

CONTINUATION:

Calling standard callback function using async:

func getTodosAsync() async -> [Todos] {

return await withCheckedContinuation { continuation in

getAllTodos { result in

……….

continuation.resume(returning: todos)

}

Async {

let todo = await getTodosAsync {

print(todo)

}

STRUCTURED CONCURRENCY:

* Async let -> allows to run multiple concurrent tasks

func getData(number: Int) -> Double {

async let (data, \_) = URLSession.shared.data(from: url) //no try await.

async let (data2, \_) = URLSession.shared.data(from: url2) //no try await.

let todos = try? JSONDecoder().decode([Todo].self, from: try await data) //await while decoding

let todos2 = try? JSONDecoder().decode([Todo].self, from: try await data2)

return calculatedData(todos, todos2)

}

Let ids = [1,2,3,4,5]

Async {

for id in ids {

let dataReceived = await getData(number: id)

print(dataReceived)

}

}

* Task cancellation:

do {

for id in ids {

try Task.cancellation()

let dataReceived = await getData(number: id)

print(dataReceived)

}

} catch { print(error)

* Task group -> allows you to run multiple child tasks based on the groups.

func group async {

try await withThrowingTaskGroup(of: Void.self) { group in

for id in ids {

group.async {

let dataReceived = try await getData(number:id)

print(dataReceived)

}

}

}

* Async Sequence:

Let endpointURL = URL(string: )

Task {

for try await line in endpointURL.allLines() {

print(line)

}

}

Pre-built in: url, urlsession, notifications can use following:

Map, flatMap, compactMap, reduce, zip, allSatisfy, max(by:), prefix, prefix(while:), joined, max , min, min(by:), filter, contains, dropFirst, joined