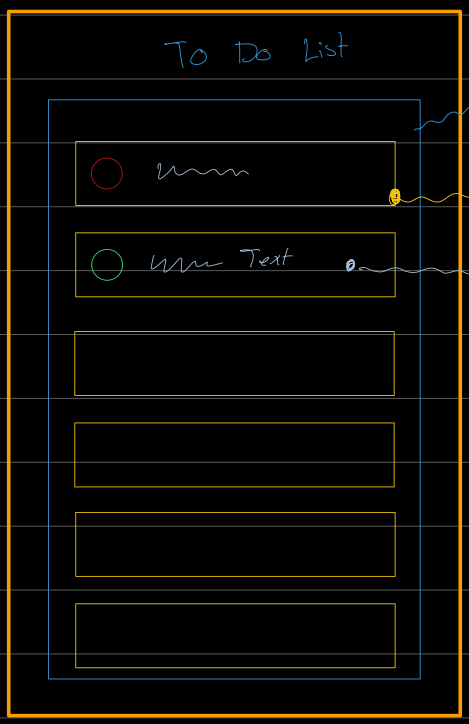

intial thoughts:

- one view for the list
- one swift for calling API
- Maybe a class for data recieved
  - must see schema
- one class for view component!
- State managment
  - Done? Not Done?

---

① check schema

How to even call API?

Store Data?

Schema?

The API supports Delete, Add, get Random.

Create Task ⇝ maybe Async later? not needed rn.

Error!

Value of URL? must be unwrapped to URL

put '!' for URL ⇝ its safe, hard coded

How to test run in terminal?

Did not have tools installed.

`xcode -select --install`

FetchData()

handle fetch Data errors and output to console for Debug

TodoResponseModel

- schema Looks like:

1- todos: [Todo]
2- total: Int
3- skip: Int (what is this??)
4- limit: Int

[
1- id: Int
2- todo: String
3- completed: Bool
4- userId: Int
]

FetchData()

@escaping: allows closure to escape function
Like "out" in C*?

↳ (Result<[todo], Error>) → Void

↳ enum → either todo [] or Error

This is async!

↓ .Resume() ~ probably Like .Result in .Net

URLSession.shared.dataTask(with: ...)

↳ Task that just fetch data of URL!

↳ { data, response, error

↳ completion handlers!

response => This is HTTPURLResponse

↳ swift type.

Completion has (.failure) ⇝ Log error

(.success) ⇝ data

if let error = error { ⇝ if error is not null (nil)

completion(.failure)

if let httpResponse = response as? HTTPURLResponse

↳ Try to cast to

guard let data = data else {

completion(.failure(NSError(domain: "", code: -1,

userInfo: nil))

}

↳ Swift class for error handling.

do { } catch { } => Literally try catch block

try

Read more weird setup.

Type.

try JSONDecoder().decode(TodoResponseModel.self, from: data)

↳ if throws error put this

Completion(todoResponse.todos)

↳ unpack

↳ on success returns the Array[] todos

↑ I need to now get this into view!

class TodoViewModel : ObservableObject {}

↳ swift

↳ combine lib

↳ should tell view when data changed

This is basically bff call the backend make view!

do an onUpdate

@Published ⇝ ObservableObject

Combine library

↳ changes to this should be sent to Anyone subscribed to this

↳ reactive programming paradigm.

Later in View use:

@ObservableObject var viewModel = TodoViewModel()

subscribes to published.

DispatchQueue.main.async { → main Queue is like JS main loop. The main thread

self?.todos = todos

}

main Queue Responsible for updating the View UI

makes sure on main thread!