Краткое пояснение к диаграмме последовательностей:

1. Открытие карточки пациента

Врач загружает снимок МРТ в систему

1. Обработка снимка

Врач загружает снимок МРТ в систему загрузки снимков МРТ

Из системы загрузки снимка МРТ снимок поступает на обработку в Систему распознавания медицинских объектов

1. Результат

Система распознавания отправляет врачу результат работы

Данная модель демонстрирует взаимодействие между Актором (Врачом) и контейнером Cистемой распознавания.

Краткое описание к модели базы данных:

1. Пользователь

Учётная запись

В зависимости от типа доступа выдаются разные полномочия

1. Тип пользователя

Позволяет распределить права доступа между пользователями

1. Администратор

Вносит пользователей в систему и выдаёт права доступа

Редактирует данные пользователей

1. Врач

Может загружать МРТ снимки в систему

Согласовывать результат с результатом системы

1. Пациент

Просмотр результатов обследования МРТ снимка

**Код с учетом принципов KISS, YAGNI, DRY и SOLID.**

import UIKit

protocol Service {

associatedtype T

var data: T { get set }

func fetchData(id: Any, completion: @escaping (T?, Error?) -> Void)

}

struct Route {

// Реализация структуры согласно бд

}

struct PointOfInterest {

// Реализация структуры согласно бд

}

class RouteService: Service {

typealias T = [Route]

var data: [Route] = []

func fetchData(id: Any, completion: @escaping ([Route]?, Error?) -> Void) {

// Логика получения снимка МРТ

// ...

// В случае успеха

completion(data, nil)

// В случае ошибки

completion(nil, error)

}

}

class PointOfInterestService: Service {

typealias T = [PointOfInterest]

var data: [PointOfInterest] = []

func fetchData(id: Any, completion: @escaping ([PointOfInterest]?, Error?) -> Void) {

// Логика получения пациента

// ...

// В случае успеха

completion(data, nil)

// В случае ошибки

completion(nil, error)

}

}

class MainViewControler: UIViewController {

var service: (any Service)?

let link = "http://url.com/route"

let routeId = 123

func getRoute() {

service = RouteService()

// Получение снимка

service?.fetchData(id: link) { (routes, error) in

if let routes = routes {

// Обработка снимка

} else if let error = error {

// Обработка ошибки

}

}

}

func getPointsOfInterest() {

service = PointOfInterestService()

// Получение результата обработки снимка МРТ

service?.fetchData(id: routeId) { (pointsOfInterest, error) in

if let pointsOfInterest = pointsOfInterest {

// Обработка обработки снимка МРТ

} else if let error = error {

// Обработка ошибки

}

}

}

}

* **Принцип KISS (Keep It Simple, Stupid):**  
  Классы RouteService и PointOfInterestService предоставляют простые методы для получения данных МРТ и пациента. Они скрывают сложность внутренней реализации, обеспечивая клиентскому коду простоту взаимодействия.
* **Принцип единственной ответственности (SOLID):**
  + S: Каждый из этих классов имеет одну основную ответственность - предоставление данных о пациенте. Это облегчает поддержку, расширение и изменение кода.
  + O: Используем протокол Service, который позволяет добавлять новые переменные и методы, не изменяя существующий код.
  + L: Наследование не используется, не продемонстровано. Swift, на котором пишу код, больше не про ООП, а протокольно-ориентированный.
  + I: В данном коде нет классов-клиентов, которые нереализовывали бы какие-либо методы протоколов. Все протоколы содержат лишь необходимые методы.
  + D: Создаём объект типа Service и можем ему подсовывать любые сервисы, выполняющие этот протокол.
* **Принцип DRY (Don't Repeat Yourself):**  
  Логика обработки ответа от сервисов вынесена в замыкания обработчиков (completion), что позволяет избежать повторения кода при обработке результатов.
* **Принцип YAGNI (You Aren't Gonna Need It):**  
  Реализована только минимально необходимая функциональность для получения данных пациента и снимках МРТ. Ненужные функции или избыточные детали отсутствуют.