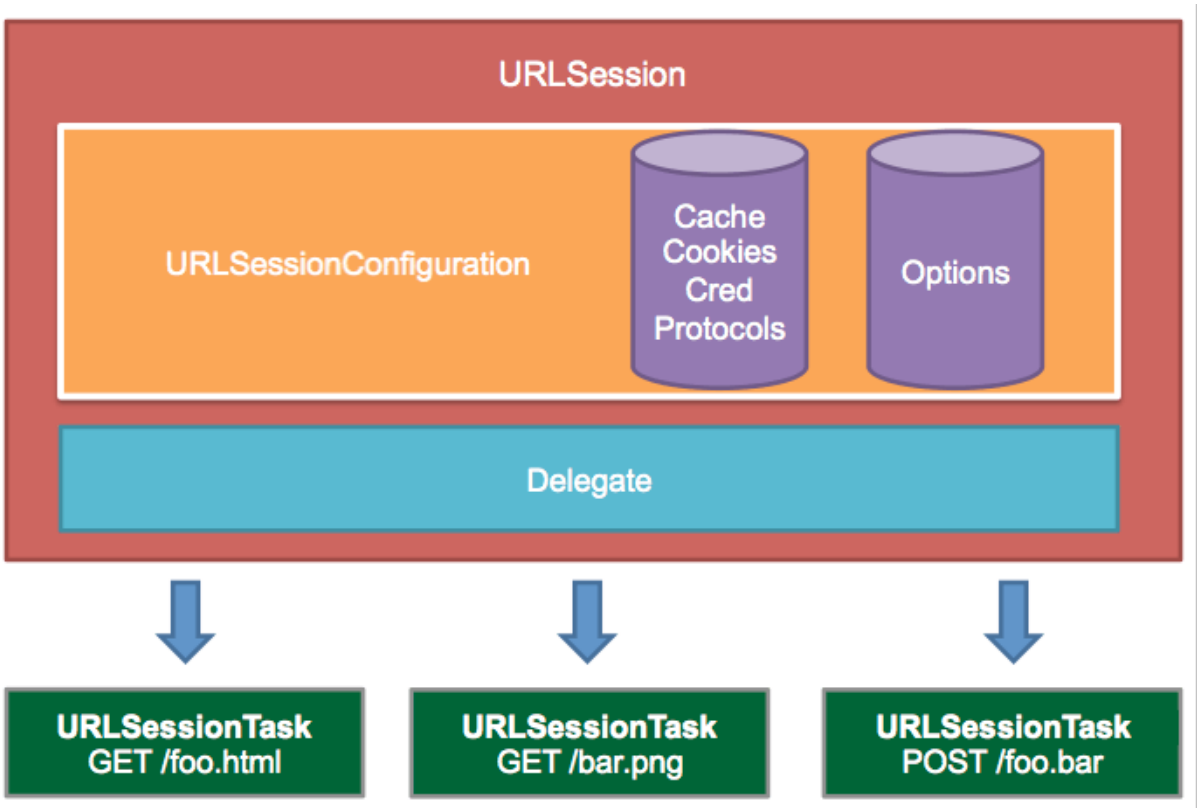
05 Работа с сетью

Для работы с сетевыми запросам Apple предоставляет API URLSession, который представляет из себя полный набор сетевых методов для загрузки и выгрузки контента через HTTP.

Технически URLSession это и класс и набор классов для обработки запросов, основанных на HTTP запросах.



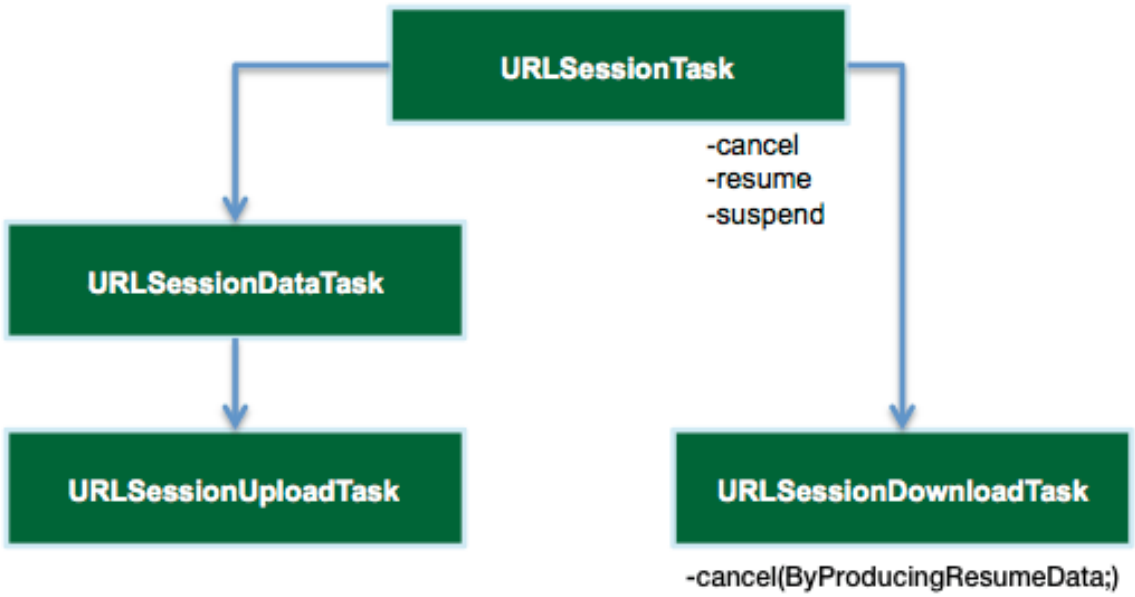
URLSession является ключевым компонентом всего стека, отвечающим за отправку и прием HTTP запросов. Для настройки экземпляра `URLSession` используется класс `URLSessionConfiguration`, который позволяет сконфигурировать параметры сессии. Всего доступно три параметра:

• **default**: Создает объект дефолтных настроек, использующих сохраняемый на диске глобальный кэш, учетные записи и хранилище куки.

• **ephemeral**: этот вариант подобен настройкам по умолчанию, за исключением того, что все данные, связанные с сеансом, сохраняются в оперативной памяти. Это можно представить как "частную" сессию.

• **background**: Позволяет сессии выполнить задание по загрузке и выгрузке данных в фоновом режиме. Передача данных продолжается даже тогда, когда приложение приостановлено или его работа прекращена.

URLSessionConfiguration также позволяет настроить свойства сессии, такие как: величину таймаута, политику кэширования и дополнительных заголовков HTTP.



URLSessionTask является абстрактным классом, который обозначает объект задания. Сеанс создает задание, который выполняет всю работу поиска данных и загрузки и выгрузки файлов на сервер.

URLSessionUploadTask: Используется для выгрузки файлов с устройства на удаленный сервер, работает по принципу методов HTTP

URLSessionDownloadTask: данная задача используется для загрузки файлов с удаленного сервера, которые сохраняются, как временные файлы в соответствующей директории.

Вы также можете приостанавливать, возобновлять и отменять задания. **URLSessionDownloadTask** имеет дополнительную возможность приостановки с последующим возобновлением работы.

URLSession возвращает данные двумя способами: с помощью замыкания, когда задание заканчивается либо успешно или с ошибкой, или путем вызова методов делегата, который вы установили при создании сеанса.

Разберем приложение для загрузки изображения из сети.

Для начала делаем проверку на валидность URL адреса:

```
guard let url = URL(string: "https://applelives.com/wp-content/
uploads/2016/03/iPhone-SE-11.jpeg") else { return }
```

Если URL адрес валидный, то создаем экземпляр `URLSession`

```
let session = URLSession.shared
```

еперь обращаемся к нашей сессии и вызываем метод `dataTask`, который создает задачу на получение содержимого по указанному URL-адресу, а затем вызывает блок, принимающий три аргумента:

- опциональный экземпляр структуры `Data`
- опциональный экземпляр класса `URLResponse`
- оптимальный экземпляр протокола `Error`

```
session.dataTask(with: url) { (data, response, error) in

}
```

Далее мы пробуем извлечь опциональное значение свойства **data** и если у нас это получается, то пробуем присвоить это значение объекту класса **UIImage**.

```
@IBAction func getImagePressed(_ sender: Any) {

    guard let url = URL(string: "https://applelives.com/wp-
content/uploads/2016/03/iPhone-SE-11.jpeg") else { return }

    let session = URLSession.shared
    session.dataTask(with: url) { (data, response, error) in
        if let data = data, let image = UIImage(data: data) {

        }
    }

}
```

В случае успешного создания объекта класса **UIImage**, нам надо передать задачу по обновлению интерфейса в основной поток, создав для этого асинхронную очередь:

```
@IBAction func getImagePressed(_ sender: Any) {

    guard let url = URL(string: "https://applelives.com/wp-
content/uploads/2016/03/iPhone-SE-11.jpeg") else { return }

    let session = URLSession.shared
    session.dataTask(with: url) { (data, response, error) in
        if let data = data, let image = UIImage(data: data) {
            DispatchQueue.main.async {
                self.label.isHidden = true
                self.getImageButton.isEnabled = false
                self.imageView.image = image
            }
        }
    }

}
```

Созданная нами задача по загрузке изображения находится в подвешенном состоянии и не будет выполнена до тех пор, пока не получит соответствующую команду. Для того, что бы запустить выполнение задачи нужно вызвать метод `resume()`:

```
session.dataTask(with: url) { (data, response, error) in
    if let data = data, let image = UIImage(data: data) {
        DispatchQueue.main.async {
            self.label.isHidden = true
            self.getImageButton.isEnabled = false
            self.imageView.image = image
        }
    }
} .resume()
```

JSON

JSON – JavaScript Object Notation. Это текстовый формат обмена данными, основанный на JavaScript. Формат JSON был разработан Дугласом Крокфордом.

Пример работы с джейсоном в котором хранятся данные по одному объекту https://swiftbook.ru//wp-content/uploads/api/api_course:

Создаем новый проект:

1. Переносим на вью контроллер кнопку и называем её Send request. Закрепляем констрейнтами.
2. Создаем ай би экшин для кнопки
3. Для преобразования полученных данных нам понадобится структура. Создаем новый файл Course
4. Переходим в новый файл и создаем структуру

```
struct Course {
    let id: Int
```

```
    let name: String
    let link: String
    let imageUrl: String
    let number_of_lessons: Int
    let number_of_tests: Int
}
```

5. Переходим в класс ViewController
6. В методе sendRequest создаем свойство с адресом нашей строки

```
let jsonUrlString = "https://swiftbook.ru//wp-content/uploads/
api/api_course"
```

7. Создаем URL

```
guard let url = URL(string: jsonUrlString) else { return }
```

8. Создаем сессию

```
URLSession.shared.dataTask(with: URL, completionHandler: (Data?,
URLResponse?, Error?) -> Void)
```

12. Подставляем `url` в запрос:

```
URLSession.shared.dataTask(with: url) { (data, responce, error)
in

    guard let data = data else { return }

    do {
        let  course = try JSONDecoder().decode(Course.self, from:
data)
        print(course.name)
    } catch let jsonError {
        print("Error serializing json", jsonError)
```

```
    }
}.resume()
```

Пример с джейсоном, который содержит массив данных: https://swiftbook.ru//wp-content/uploads/api/api_courses Т.к. джейсон содержит массив данных, то и в коде мы должны обращаться к массиву. Так же не забываем поменять ссылку на новый джейсон

```
@IBAction func getData(_ sender: Any) {

    let jsonUrlString = "https://swiftbook.ru//wp-content/
uploads/api/api_courses"

    guard let url = URL(string: jsonUrlString) else { return }

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

        guard let data = data else { return }

        do {

            let courses = try JSONDecoder().decode([Course].self,
from: data)
            print(courses)

        } catch let jsonError {
            print("Error serializing json", jsonError)
        }

    }.resume()
```

Джейсон с именем и описанием: https://swiftbook.ru//wp-content/uploads/api/api_website_description

1. Создадим новый файл со структурой WebsiteDescription

```
struct WebsiteDescription {
    let name: String
```

```
    let description: String
    let courses: [Course]
}
```

2. Переходим в класс SecondViewController
3. Меняем ссылку для свойства jsonUrlString
4. Переходим к отправке запроса и меняем код в соответствии с новыми условиями

```
let websiteDescription = try
JSONDecoder().decode(WebsiteDescription.self, from: data)
```

5. Присваиваем тип Decodable структуре WebsiteDescription

```
struct WebsiteDescription: Decodable {
    let name: String
    let description: String
    let courses: [Course]
}
```

6. Возвращаемся обратно и выводим на консоль данные об имени и описание

```
print(websiteDescription.name, websiteDescription.description)
```

Джейсон – это текстовый формат данных и как и в любом текстовом формате, в нем могут содержаться ошибки или пропущенные поля, как в последне примере: https://swiftbook.ru//wp-content/uploads/api/api_missing_or_wrong_fields

1. Меняем ссылку для свойства jsonUrlString
2. Меняем логику в айбиэкшине

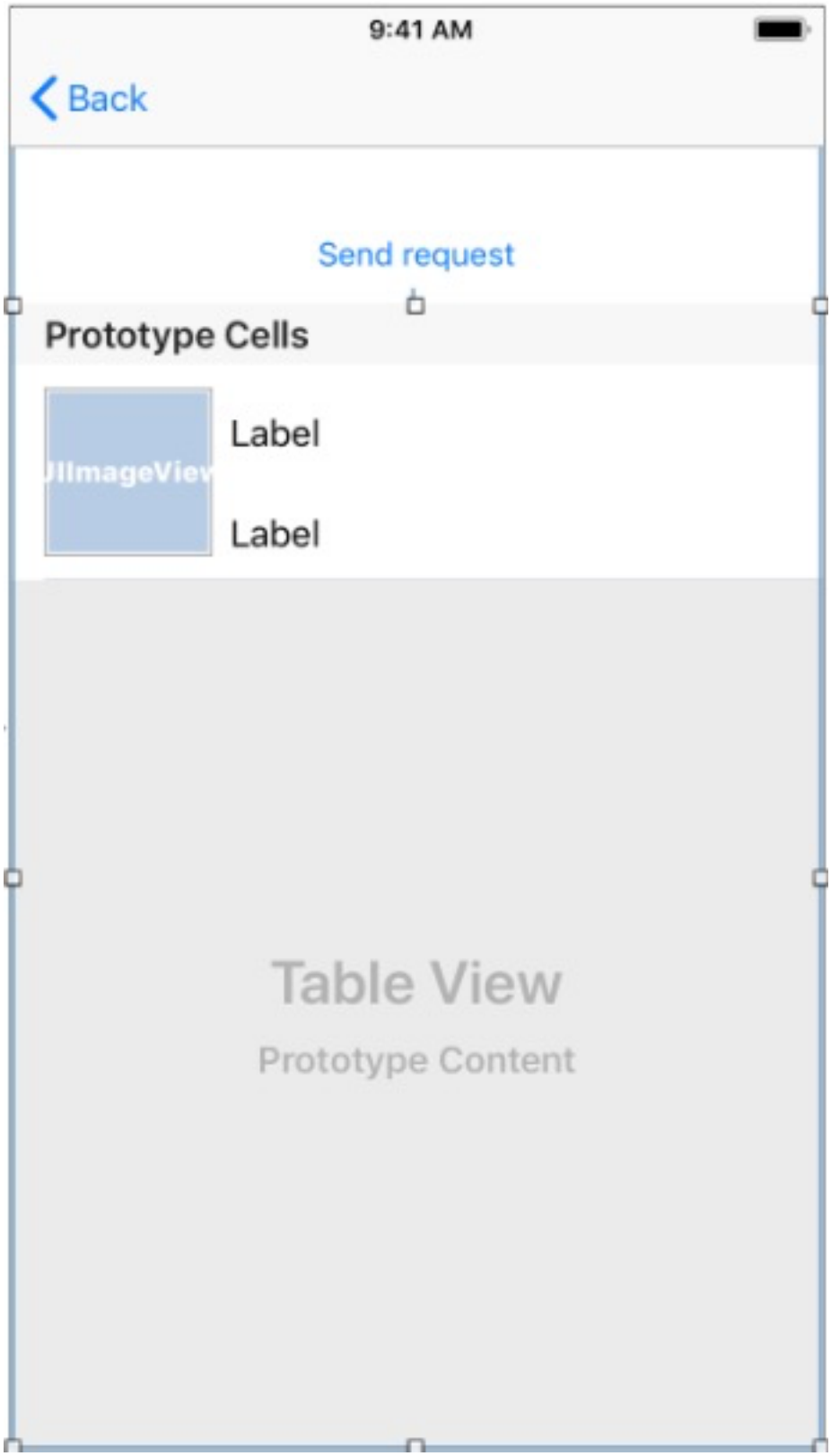
```
let courses = try JSONDecoder().decode([Course].self, from: data)
print(courses)
```

Запускаем симулятор, убеждаемся, что срабатывает ветвление catch

3. Делаем все свойства структуры Course опциональными

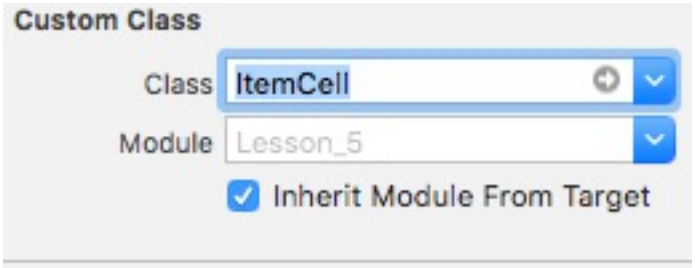
Парсинг джейсона

1. Из библиотеки объектов перетаскиваем таблицу на контроллер, размещаем её под кнопкой и задаем размеры и ограничения. Также создаем outlet для работы с этой таблицей.
2. Подписываемся под протоколы `UITableViewDelegate` и `UITableViewDataSource`
3. Из библиотеки объектов перетаскиваем ячейку в таблицу. Настраиваем высоту ячейки (я сделал 100), и размещаем на этой ячейке нужные элементы.



4. Создаем новый файл с классом **ItemCell**

5. Связываем ячейку с классом



6. Создаем аутлеты для каждого элемента ячейки

```
@IBOutlet var courseImage: UIImageView!
@IBOutlet var courseNameLabel: UILabel!
@IBOutlet var numberOfLessons: UILabel!
@IBOutlet var numberOfTests: UILabel!
```

7. Переходим в класс `ViewController` и добавляем свойство для хранения элементов

```
fileprivate var courses = [Course]()
```

8. Создаем расширение для вьюконтроллера и добавляем в него обязательные методы протоколов **UITableViewDataSource** и **UITableViewDelegate**

```
extension SecondViewController: UITableViewDelegate,
UITableViewDataSource {

    func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return courses.count
    }
```

```
    func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"Cell") as! ItemCell
        return cell
    }
}
```

9. Создаем приватный метод для конфигурации ячейки:

```
private func configureCell(cell: TableViewCell, for indexPath:
IndexPath) {

    let course = courses[indexPath.row]
    cell.courseNameLabel.text = course.name

    if let numberOfLessons = course.numberOfLessons {
        cell.numberOfLessons.text = "Number of lessons: \
(numberOfLessons)"
    }

    if let numberOfTests = course.numberOfTests {
        cell.numberOfTests.text = "Number of tests: \
(numberOfTests)"
    }

    guard let imageUrl = URL(string: course.imageUrl!) else
{ return }
    guard let imageData = NSData(contentsOf: imageUrl as URL)
else { return }

    cell.courseImage.image = UIImage(data: imageData as Data)
}
```

10. Вызываем метод `configureCell` из `cellForRowAt indexPath`:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
```

```
    let cell = tableView.dequeueReusableCell(withIdentifier:
"Cell") as! ItemCell
    configureCell(cell: cell, for: indexPath)
    return cell
}
```

1. Меняем адрес джейсона на https://swiftbook.ru//wp-content/uploads/api/api_courses
2. Переходим в айби экшин и заполняем элементы ячейки данными.

```
do {
    self.courses = try JSONDecoder.decode([Course].self, from:
data)
}
```

3. Для того, что бы отобразить полученные данные, нужно запустить в основном потоке метод таблицы `reloadData()`.

```
do {
    self.courses = try JSONDecoder.decode([Course].self, from:
data)

    DispatchQueue.main.async {
        self.tableView.reloadData()
    }
}
```

From snake_case to camelCase

Переходим в файл `Course` и меняем стиль свойства для количества уроков и тестов на **Camel Case**:

```
let numberOfLessons: Int?
```

```
let numberOfTests: Int?
```

Переходим в класс `ViewController`:

```
do {
    let decoder = JSONDecoder()
    decoder.keyDecodingStrategy = .convertFromSnakeCase
    self.courses = try decoder.decode([Course].self, from: data)

    DispatchQueue.main.async {
        self.tableView.reloadData()
    }
}
```

POST Request

Для создания POST запроса мы будем использовать сервис JSON Placeholder:

```
@IBAction func postRequest(_ sender: Any) {

    guard let url = URL(string: "https://
jsonplaceholder.typicode.com/posts") else { return }
}
```

В отличие от GET запроса, где мы просто считываем данные, в POST запросе мы должны выполнить передачу данных, поэтому создадим словарь с данными, которые будем передавать на сервер:

```
@IBAction func postRequest(_ sender: Any) {

    guard let url = URL(string: "https://
jsonplaceholder.typicode.com/posts") else { return }

    let userData = ["Course": "Networking", "Lesson": "GET and
POST"]
}
```

И прежде чем создавать сессию, нужно указать метод запроса:

```
@IBAction func postRequest(_ sender: Any) {

    guard let url = URL(string: "https://
jsonplaceholder.typicode.com/posts") else { return }

    let userData = ["Course": "Networking", "Lesson": "GET and
POST"]

    var request = URLRequest(url: url)
    request.httpMethod = "POST"
}
```

Теперь передадим созданные выше параметры для отправки на сервер в тело запроса:

```
@IBAction func postRequest(_ sender: Any) {

    guard let url = URL(string: "https://
jsonplaceholder.typicode.com/posts") else { return }

    let userData = ["Course": "Networking", "Lesson": "GET and
POST"]

    var request = URLRequest(url: url)
    request.httpMethod = "POST"

    guard let httpBody = try?
JSONSerialization.data(withJSONObject: userData, options: [])
else { return }
    request.httpBody = httpBody
}
```

Далее создаем сессию:

```
let session = URLSession.shared
session.dataTask(with: request) { (data, response, error) in

    guard let response = response, let data = data else
{ return }

    print(response)
```

```
    do {
        let json = try JSONSerialization.jsonObject(with: data,
options: [])
        print(json)
    } catch {
        print(error)
    }
} .resume()
```

Запускаем проект и смотрим на консоль. Статус код 201 говорит о том, что данные добавлены успешно. Хоть данные и получилось добавить, но сделали мы это не корректно.

```
"{\"Lesson\":\"GET and POST\",\"Course\":\"Networking\"}" = ""
```

В качестве ключа мы передали все ключи и значения словаря, при этом само значение осталось пустым. Проблема заключается в том, что мы не создали правило добавления новых записей. Давайте это исправим, скопировав необходимые данные из консоли:

```
"Content-Type" = ("application/json; charset=utf-8");
```

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

Снова запускаем проект и убеждаемся, что данные добавляются корректно