Урок 1

Карты и локация

Отображение карты, создание объектов на карте. Определение текущего местоположения. Определение адреса по координатам и наоборот

Карты

- Провайдеры карт
- Добавление Google Maps в приложение
- Отображение карты
- Добавление элементов на карту
- Взаимодействие с картой

CoreLocation

CLGeocoder

Практическое задание

Дополнительные материалы

Используемая литература

Карты

На этом уроке мы будем говорить о навигационных картах. Если не считать стандартных элементов управления, карты – пожалуй, самый распространённый элемент интерфейса. С одной стороны, карта – это обычный **UIView,** который достаточно добавить на экран. С другой – мы можем управлять десятками параметров, такими как зум, местоположение, внутренние элементы интерфейса и т. д.

Провайдеры карт

Сейчас на рынке достаточно много поставщиков карт. Рассмотрим основных.

Apple Maps – единственный фреймворк, предоставленный самим Apple. Из плюсов можно назвать простоту интеграции в приложение и доступность из коробки. На этом плюсы заканчиваются: эти карты имеют плохую детализацию и не пользуются популярностью даже в США. В России фреймворк использует данные партнеров, но даже это не помогает найти нужный адрес.

Google Maps – наверное, самые популярные карты в мире. У них приемлемый уровень детализации даже для России (у карт крупных городов), они достаточно просто интегрируются в приложение, хоть и уже сложнее, чем Apple Maps.

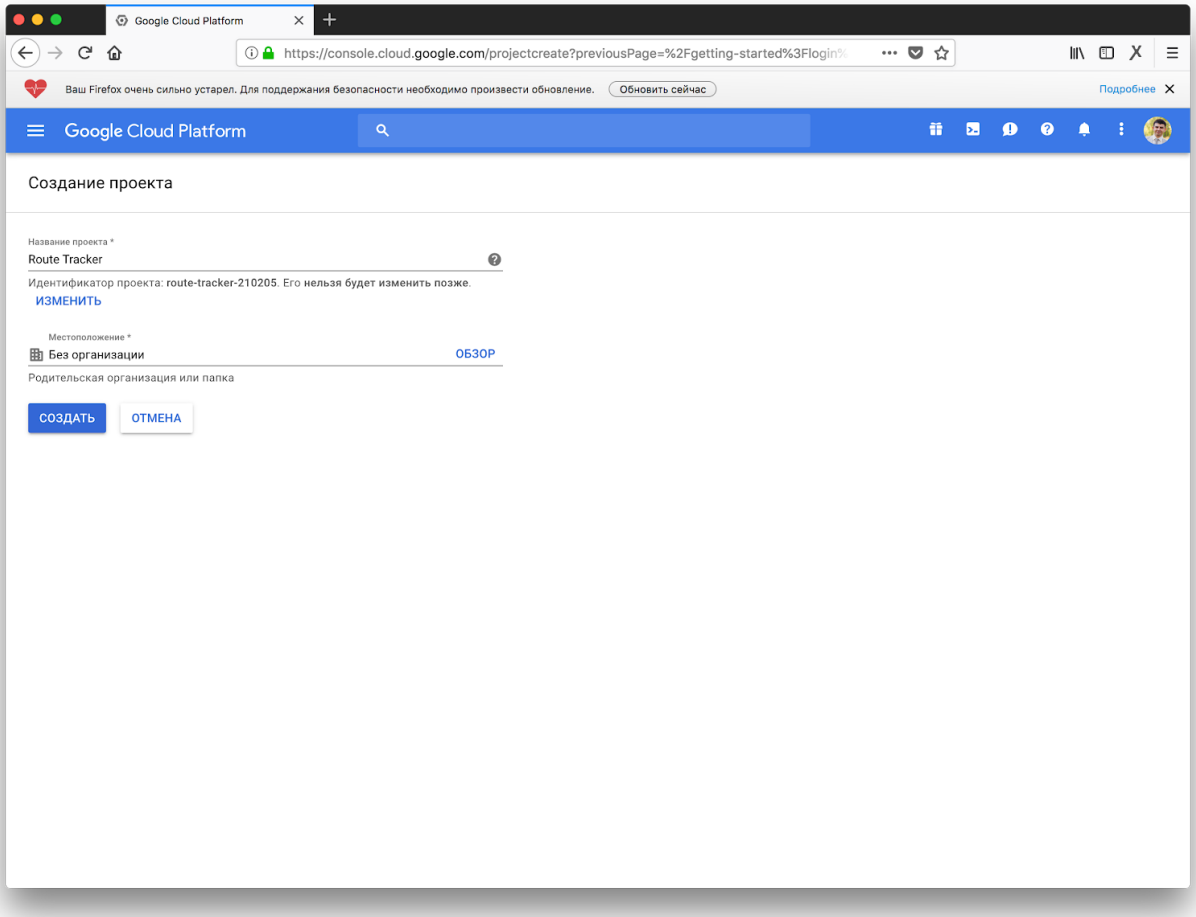
Yandex Maps – лучший вариант для стран СНГ: максимальная детализация, зачастую можно найти адреса еще не существующих объектов, отображаются пробки, отличная навигация. Карты от Яндекса были бы идеальным вариантом, но у них нет фреймворка для iOS – вернее, есть обёртка на html-странице с довольно скудным набором настраиваемых параметров, которых в большинстве случаев не хватает для нужд заказчика. Вот пример из практики: в приложении нужно было использовать именно этот инструмент, и после исследования всех возможностей и разговора с разработчиками из Яндекса выяснилось, что проще использовать **webView** и отображать веб-версию, а команды передавать в JavaScript и выполнять их, как будто это сайт.

Мы с вами будем изучать карты на примере Google Maps, так как у них лучшее соотношение простоты внедрения и возможностей.

Добавление Google Maps в приложение

Работа с данным фреймворком начинается со входа в Google Cloud Platform. Для этого нужно зарегистрировать аккаунт Google.

Теперь создадим новый новый проект. Для этого нажмём на кнопку «Выберите проект» и в открывшемся окне – «Создать проект».



Заполним название проекта и нажмём кнопку «Создать». После этого мы вернёмся на главную страницу консоли или, возможно, в настройки проекта (они показаны ниже).

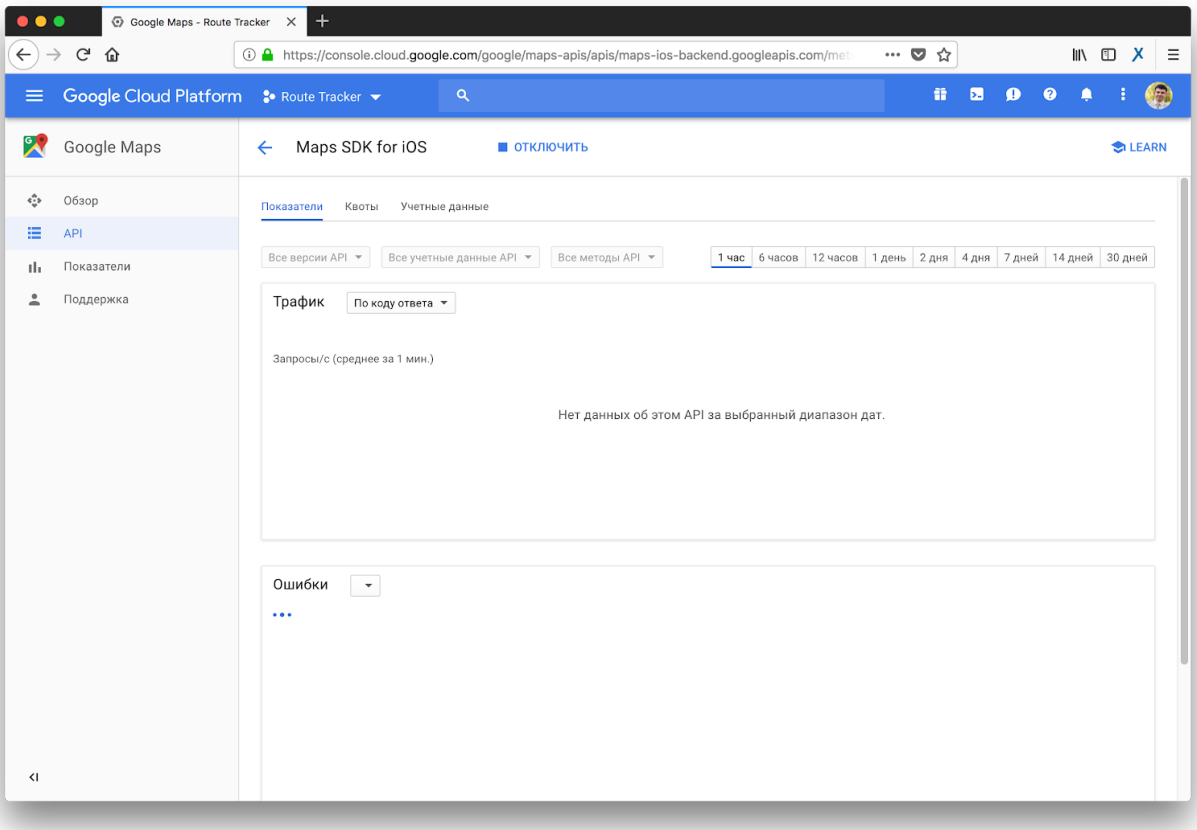
Чтобы попасть в настройки (если нас не направили туда изначально), нажмём на название платформы в заголовке.

Теперь нам надо перейти к списку доступных API. Для этого нажмём «Включить API и создать ключи». Далее мы попадаем в панель управления API.

Нажмём кнопку «Включить API и сервисы» для выбора необходимого фреймворка.

Здесь нам необходимо выбрать «Maps SDK for iOS». Если его не видно в списке, можно воспользоваться поиском или открыть категорию «Maps».

Внутри выбранного пункта нажмём «Включить». После короткой задержки мы попадём в панель управления API карт.



Осталось совсем немного до получения ключа. Перейдём на вкладку «Учётные данные».

Нажмём кнопку «Создать учётные данные» и выберем пункт «Ключ API». Откроется окно, откуда можно скопировать полученный ключ. Если вы забудете его скопировать, не волнуйтесь: он будет доступен в панели управления.

Прикреплять ограничения не нужно, просто закройте окно.

Теперь мы создадим проект, добавим в него поддержку cocoapods и установим **pod:** pod 'GoogleMaps'. Этот процесс рассмотрен в предыдущих курсах.

После запуска проекта с уже установленными зависимостями мы откроем **AppDelegate** и настроим карты, указав полученный API ключ.

```
import UIKit
import GoogleMaps // Импорт фреймворка

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


     func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

// Настройка ключа
        GMSServices.provideAPIKey("HGHgHJGhjGhjGhjGHJgHJGjhGjhGjGJgJg")

        return true
    }

}
```

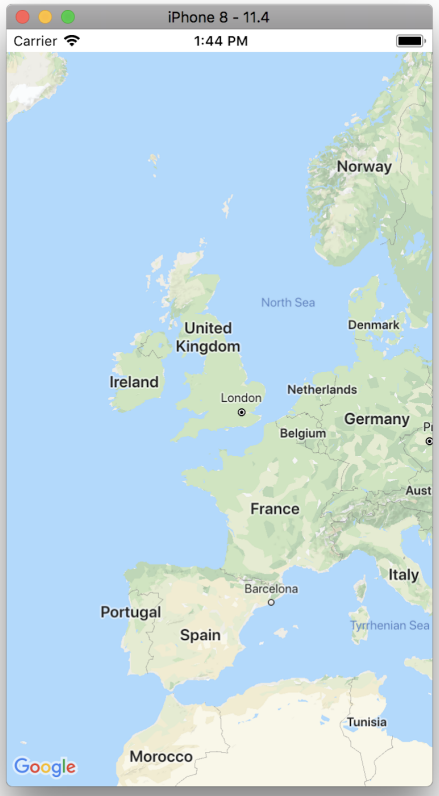
Теперь приложение готово для использования карт.

Отображение карты

Самое простое, что можно сделать с картой, – показать её на экране.

Мы создадим контроллер **MapViewController**, добавим на него **UIView**, с помощью констрейнтов растянем на весь экран и установим класс **GMSMapView**.

Все эти действия вам знакомы, хоть мы и никогда не назначали кастомные классы элементам (мы делали это только для контроллеров, но процессы ничем не отличаются). Класс **GMSMapView** доступен нам благодаря подключённому фреймворку Google. То, что мы не видим карту на самом storyboard, нормально: она отрисовывается в реальном времени при запуске устройства. Это мы сейчас и сделаем.



Карта открывается во весь экран. Но, во-первых, она нацелена на Лондон, во-вторых, захватывает слишком большой участок. Давайте это исправим.

Нам надо создать **IBOutlet** для карты (вы, конечно, помните, как это делать). Кроме того, в контроллере также стоит импортировать **GoogleMaps**, иначе мы получим ошибку.

```
import UIKit
import GoogleMaps

class MapViewController: UIViewController {
    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()

    }

}
```

Теперь мы можем взаимодействовать с картой. Напишем отдельный метод для настройки карты. За область отображения на карте отвечает **GMSCamera**. Если необходимо изменить то, что мы видим, нужно создать свою камеру. К счастью, это делается довольно просто. Для начала определимся с местом, например, выберем Красную площадь (широта 55.753215, долгота 37.622504).

```
func configureMap() {
// Центр Москвы
        let coordinate = CLLocationCoordinate2D(latitude: 55.753215, longitude:
37.622504)
```

```
// Создаём камеру с использованием координат и уровнем увеличения
        let camera = GMSCameraPosition.camera(withTarget: coordinate, zoom: 17)
// Устанавливаем камеру для карты
        mapView.camera = camera
    }
```

В первой строке мы получаем объект с координатами точки, в нашем случае – Красной площади. Делается это просто: вызываем конструктор у класса **CLLocationCoordinate2D,** затем создаём камеру для карты через вызов статического метода **camera** у класса **GMSCameraPosition,** при этом надо передать координаты и зум. Зум – это масштаб карты: чем больше зум, тем ближе мы к точке. Если установить зум 1, мы будем видеть всю Землю, при зуме 17 мы будем видеть небольшой кусочек вокруг искомой точки, но он будет отображаться довольно подробно. Устанавливаем полученную камеру в свойство **camera** у карты.

Установка камеры – не единственная возможность изменить область на карте. Давайте добавим кнопку на контроллер и только при нажатии на неё будем переходить к Красной площади. Проще всего добавить **UINavigationController** и уже в **UINavigationBar** добавить кнопку.

Не забудьте создать **IBAction** и перенести в него метод **configureMap**.

```
class MapViewController: UIViewController {
    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    func configureMap() {
// Центр Москвы
        let coordinate = CLLocationCoordinate2D(latitude: 55.753215, longitude:
37.622504)
// Создаём камеру с использованием координат и уровнем увеличения
        let camera = GMSCameraPosition.camera(withTarget: coordinate, zoom: 17)
// Устанавливаем камеру для карты
        mapView.camera = camera
    }

    @IBAction func goTo(_ sender: Any) {
        configureMap()
    }
}
```

После нажатия на кнопку карта мгновенно перемещается на новое место. Как правило, при перемещении на большое расстояние, такое как от Лондона до Москвы, это неизбежный эффект, но, если переместиться на небольшое расстояние, камера будет вести себя так же. Перейдём к Москве: сдвинем мышкой карту и нажмём кнопку. Карта рисуется заново.

Этот способ хорошо подходит, если надо задать начальную позицию, но перемещение фокуса в другое место выглядит не очень красиво. Если необходимо показать процесс изменения карты, есть другой метод: **animate(toLocation:).** Перепишем пример следующим образом.

```
class MapViewController: UIViewController {

// Центр Москвы
      let coordinate = CLLocationCoordinate2D(latitude: 59.939095, longitude:
30.315868)

    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()

        configureMap()
    }

    func configureMap() {
// Создаём камеру с использованием координат и уровнем увеличения
        let camera = GMSCameraPosition.camera(withTarget: coordinate, zoom: 17)
// Устанавливаем камеру для карты
        mapView.camera = camera
    }

    @IBAction func goTo(_ sender: Any) {
        mapView.animate(toLocation: coordinate)
    }
}
```

Что изменилось? Координаты вынесены из метода **configureMap** в свойство контроллера: теперь они будут использоваться в двух разных местах. Сам метод **configureMap** вернулся в **viewDidLoad,** то есть при запуске экрана мы сразу установим карту на Красную площадь. При нажатии на кнопку мы будем вызывать метод **animate(toLocation:),** передавая в него всё те же координаты.

Проверим, как это работает, проведя эксперимент. Немного отодвинем карту вручную с её позиции, а потом нажмём кнопку. В результате карта плавно переместится, а не нарисуется заново. Но при перемещении на большие расстояния этого эффекта, к сожалению, также не будет видно.

Кроме изменения местоположения, есть различные варианты метода **animate**. Мы можем анимировать зум, новую камеру или просто позицию камеры. При этом позиция камеры включает в себя координаты, зум, поворот камеры.

Теперь давайте посмотрим, как ещё мы можем настроить карту и какие параметры у неё есть:

- **isTrafficEnabled** – включить/выключить показ пробок;.
- **mapType** – тип карты (карта, спутник, гибрид, рельеф);
- **maxZoom** – ограничение на максимальный уровень зума;
- **minZoom** – ограничение на минимальный уровень зума.

На этом параметры у карты не заканчиваются. У нее есть свойство **settings** с такими параметрами:

- **scrollGestures** – включить/выключить прокрутку;
- **zoomGestures** – включить/выключить зум;
- **tiltGestures** – включить/выключить наклон;
- **rotateGestures** – включить/выключить поворот;
- **myLocationButton** – включить/выключить кнопку «Моя позиция»;
- **allowScrollGesturesDuringRotateOrZoom** – включить/выключить прокрутку в момент поворота или зума.

Как правило, они используются не очень часто.

Карту можно не только настроить, но и стилизовать, чтобы гармонично вписать её в интерфейс приложения. Настройка выполняется с помощью файла json. Давайте рассмотрим пример с официального сайта и перекрасим карту в тёмные цвета.

```
func configureMapStyle() {
        let style = "[" +
            "  {" +
            "    \"featureType\": \"all\"," +
            "    \"elementType\": \"geometry\"," +
            "    \"stylers\": [" +
            "      {" +
            "        \"color\": \"#242f3e\"" +
            "      }" +
            "    ]" +
            "  }," +
            "  {" +
            "    \"featureType\": \"all\"," +
            "    \"elementType\": \"labels.text.stroke\"," +
            "    \"stylers\": [" +
            "      {" +
            "        \"lightness\": -80" +
            "      }" +
            "    ]" +
            "  }," +
            "  {" +
            "    \"featureType\": \"administrative\"," +
            "    \"elementType\": \"labels.text.fill\"," +
            "    \"stylers\": [" +
            "      {" +
            "        \"color\": \"#746855\"" +
            "      }" +
            "    ]" +
            "  }," +
            "  {" +
            "    \"featureType\": \"administrative.locality\"," +
            "    \"elementType\": \"labels.text.fill\"," +
            "    \"stylers\": [" +
            "      {" +
            "        \"color\": \"#d59563\"" +
            "      }" +
            "    ]" +
            "  }," +
            "  {" +
            "    \"featureType\": \"poi\"," +
            "    \"elementType\": \"labels.text.fill\"," +
            "    \"stylers\": [" +
            "      {" +
            "        \"color\": \"#d59563\"" +
            "      }" +
            "    ]" +
            "  }," +
            "  {" +
            "    \"featureType\": \"poi.park\"," +
            "    \"elementType\": \"geometry\"," +
            "    \"stylers\": [" +
            "      {" +
            "        \"color\": \"#263c3f\"" +
            "      }" +
```

```
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"poi.park\"," +
"    \"elementType\": \"labels.text.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#6b9a76\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road\"," +
"    \"elementType\": \"geometry.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#2b3544\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road\"," +
"    \"elementType\": \"labels.text.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#9ca5b3\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.arterial\"," +
"    \"elementType\": \"geometry.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#38414e\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.arterial\"," +
"    \"elementType\": \"geometry.stroke\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#212a37\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.highway\"," +
"    \"elementType\": \"geometry.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#746855\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.highway\"," +
"    \"elementType\": \"geometry.stroke\"," +
```

```
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#1f2835\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.highway\"," +
"    \"elementType\": \"labels.text.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#f3d19c\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.local\"," +
"    \"elementType\": \"geometry.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#38414e\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"road.local\"," +
"    \"elementType\": \"geometry.stroke\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#212a37\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"transit\"," +
"    \"elementType\": \"geometry\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#2f3948\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"transit.station\"," +
"    \"elementType\": \"labels.text.fill\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#d59563\"" +
"      }" +
"    ]" +
"  }," +
"  {" +
"    \"featureType\": \"water\"," +
"    \"elementType\": \"geometry\"," +
"    \"stylers\": [" +
"      {" +
"        \"color\": \"#17263c\"" +
"      }" +
"    ]" +
```

```
            "  },\" +
            "  {\" +
            "    \"featureType\": \"water\"," +
            "    \"elementType\": \"labels.text.fill\"," +
            "    \"stylers\": [\" +
            "      {\" +
            "        \"color\": \"#515c6d\"\" +
            "      }\" +
            "    ]\" +
            "  },\" +
            "  {\" +
            "    \"featureType\": \"water\"," +
            "    \"elementType\": \"labels.text.stroke\"," +
            "    \"stylers\": [\" +
            "      {\" +
            "        \"lightness\": -20\" +
            "      }\" +
            "    ]\" +
            "  }\" +
        "]"
        do {
            mapView.mapStyle = try GMSMapStyle(jsonString: style)
        } catch {
            print(error)
        }

    }
```

Перед вами невероятно длинный метод, но весь он состоит из json для стилизации карты, и только пара строк в конце – применение этого json.

Разберём, по каким правилам строится json. Он довольно прост, если понять основные принципы. Это массив объектов, состоящих из трёх свойств:

- **featureType** – объект на карте, который мы стилизуем.
- **elementType** – часть объекта, которую стилизуем.
- **stylers** – стиль, который будет применён к части объекта.

Посмотрим на один из блоков json:

```
{
    "featureType": "water",
    "elementType": "labels.text.stroke",
        "stylers": [
            {
                "lightness": -20
            }
        ]
}
```

- **"featureType": "water"** – стилизуем изображение водоёмов;
- **"elementType": "labels.text.stroke"** – стилизуем обводку текста надписей у водоёмов;
- **"lightness": –20** – уменьшаем на 20 пунктов яркость стандартного стиля (он станет темнее).

Ниже перечислены все возможные объекты, которые поддаются стилизации.

- **all** – стиль применится ко всем объектам;
- **administrative** – все административные зоны;
- **administrative.country** – страна;

- **administrative.land_parcel** – земельный участок;
- **administrative.locality** – местность;
- **administrative.neighborhood** – район;
- **administrative.province** – область/провинция;
- **landscape** – все элементы ландшафта;
- **landscape.man_made** – рукотворные элементы ландшафта;
- **landscape.natural** – естественные элементы ландшафта;
- **landscape.natural.landcove**r – растительность;
- **landscape.natural.terrain** – рельеф (например, горы);
- **poi** – все объекты инфраструктуры;
- **poi.attraction** – достопримечательности;
- **poi.business** – бизнес-объекты;
- **poi.government** – правительственные;
- **poi.medical** – медицинские;
- **poi.park** – парки;
- **poi.place_of_worship** – религиозные;
- **poi.school** – образовательные;
- **poi.sports_complex** – спортивные;
- **road selects** – все дороги;
- **road.arterial** – магистрали;
- **road.highway** – шоссе;
- **road.highway.controlled_access** – платные шоссе;
- **road.local** – обычные дороги;
- **transit** – весь междугородный транспорт;
- **transit.line** – междугородные маршруты;
- **transit.station** – все станции отправления;
- **transit.station.airport** – аэропорты;
- **transit.station.bus** – автобусные остановки;
- **transit.station.rail** – железнодорожные вокзалы;
- **water** – водоёмы.

Каждый из объектов имеет одинаковый набор частей, которые можно использовать:

- **all** – стиль будет применен ко всем частям;
- **geometry** – вся фигура;
- **geometry.fill** – заливка фигуры;
- **geometry.stroke** – обводка фигуры;
- **labels** – вся текстовая надпись;
- **labels.icon** – иконка внутри надписи;
- **labels.text** – текст внутри надписи;
- **labels.text.fill** – заливка текста;
- **labels.text.stroke** – обводка текста.

Кажется, что объектов и их составных частей достаточно, чтобы с картой можно было сделать всё, что требуется, но на самом деле этого мало. Например, если мы захотим стилизовать жилые дома, у нас не будет такой возможности: мы сможем выбрать только элемент **landscape.man_made,** а это целый квартал города, включая дворы и здания. Как именно будет применён стиль, можно узнать только экспериментальным путём. Кстати, начиная с определённого масштаба дома перейдут в режим 3D, и цвет их контуров нельзя будет задать отдельно: автоматически будет выбран оттенок установленного цвета, который достаточно сложно разглядеть.

Последнее – список параметров стиля.

- **hue** – цвет;
- **lightness** (–100...100) – яркость;
- **saturation** (–100...100) – насыщенность;
- **gamma** (0.01...10.0) – гамма;
- **invert_lightness** – изменяет значение яркости на противоположное;
- **visibility** (on, off, simplified) – отображение элемента (включено, выключено, упрощённое);
- **color** – цвет;

- **weight** – размер.

Каждый из четырёх параметров (hue, lightness, saturation, gamma) изменяет только одну настройку цвета, остальные остаются по умолчанию. Как правило, проще установить необходимый цвет, используя параметр **color:** это изменит сразу все 4 параметра.

Добавление элементов на карту

Давайте вернём карте обычный вид, удалив метод **configureMapStyle,** и разместим на ней простейший стандартный маркер. Для начала добавим ещё одну кнопку на панель навигации и создадим для неё **IBAction**.

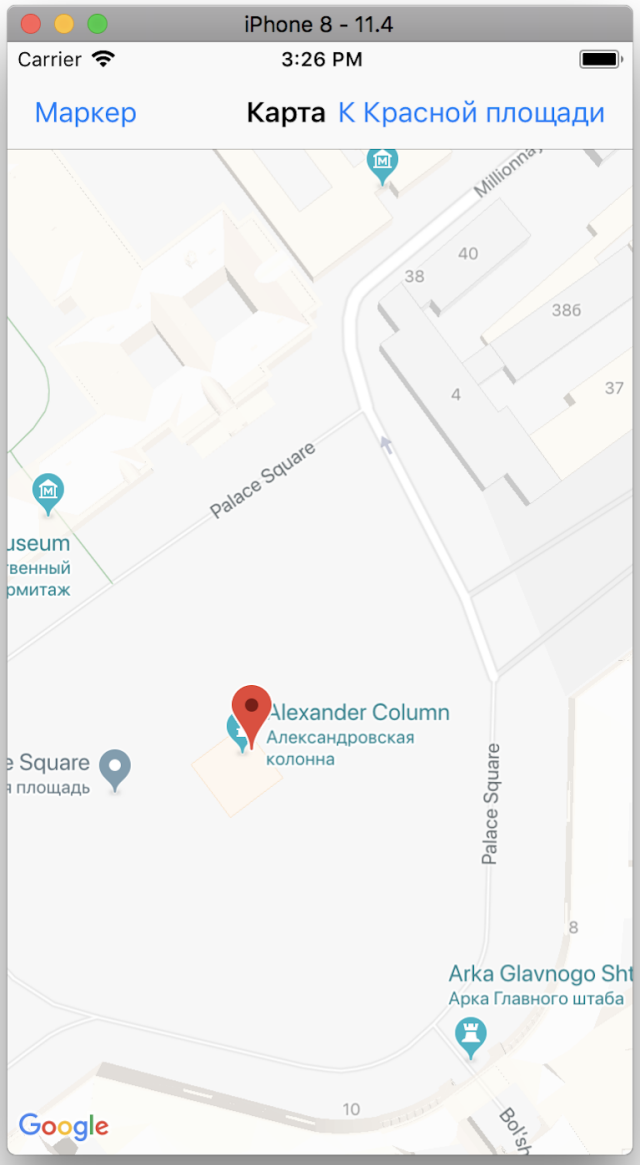
Напишем метод для добавления маркера и вызовем его при нажатии на кнопку.

```
@IBAction func toggleMarker(_ sender: Any) {
    addMarker()
}

func addMarker() {
    let marker = GMSMarker(position: coordinate)
    marker.map = mapView
}
```

Маркер представлен классом **GMSMarker.** При его создании достаточно указать координаты и карту, на которой он должен быть отрисован. Передавать карту маркеру, а не наоборот, не очень логично, но Google Maps работают так.

Результат будет таким:



Удаление маркера:

```
class MapViewController: UIViewController {

    <...>
    var marker: GMSMarker?

    <...>

    @IBAction func toggleMarker(_ sender: Any) {
        if marker == nil {
            addMarker()
        } else {
            removeMarker()
        }

    }
```

```
    func addMarker() {
        let marker = GMSMarker(position: coordinate)
        marker.map = mapView
        self.marker = marker
    }

    func removeMarker() {
        marker?.map = nil
        marker = nil
    }
}
```

Необходимо свойство для хранения маркера, назовем его **marker**. Добавим метод **removeMarker:** в нём мы удалим у маркера карту и сам маркер – тоже. В методе **addMarker** мы допишем установку маркера в одноимённое свойство. При нажатии на кнопку мы проверяем, имеется ли маркер, и если да – удаляем, если нет – добавляем.

К счастью, маркер кастомизируется намного проще, чем сама карта. Рассмотрим его параметры.

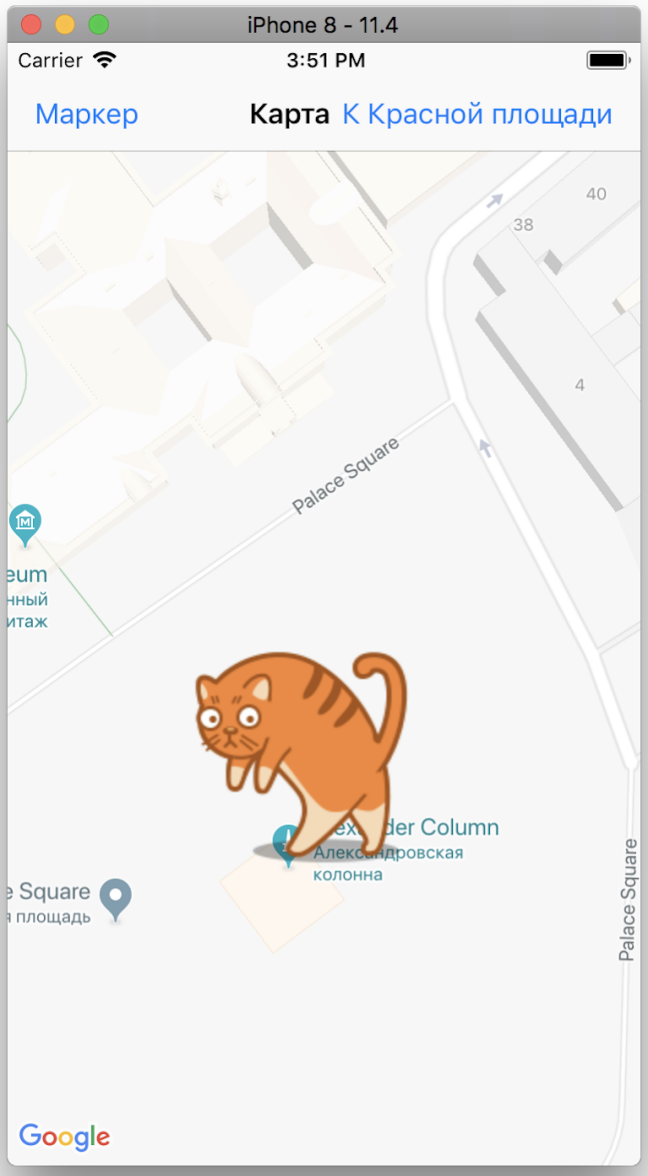
Во-первых, мы можем изменить его цвет.

```
    func addMarker() {
        let marker = GMSMarker(position: coordinate)
// Получаем стандартное изображение маркера, перекрасив его в зелёный
// Устанавливаем как изображение маркера
        marker.icon = GMSMarker.markerImage(with: .green)
        marker.map = mapView
        self.marker = marker
    }
```

Как видите, у класса **GMSMarker** есть статический метод **markerImage(with:),** который возвращает стандартную иконку, перекрашенную в выбранный цвет. Нам остаётся только установить это изображение в качестве иконки у маркера:

В качестве иконки можно установить любое изображение:

```
    func addMarker() {
        let marker = GMSMarker(position: coordinate)
        marker.icon = UIImage(named: "cat")
        marker.map = mapView
        self.marker = marker
    }
```



Более того, мы можем передать маркеру любой **UIView:**

```
    func addMarker() {
        let rect = CGRect(x: 0, y: 0, width: 20, height: 20)
        let view = UIView(frame: rect)
        view.backgroundColor = .red

        let marker = GMSMarker(position: coordinate)
        marker.iconView = view
        marker.map = mapView
        self.marker = marker
    }
```

Создадим обычный квадратный **UIView,** явно указав его размеры, сделаем красным и установим в свойство **iconView.**

При этом установленная view может быть довольно сложной: в ней может быть надпись, которую можно менять, или даже кнопка.

Любому маркеру можно добавить информационную надпись, которая отображается при клике на него. Для этого существуют два свойства: **title** – заголовок информационного окна, **snippet** – текст в окне. Эти свойства можно использовать как одновременно, так и по отдельности:

```
func addMarker() {
    let marker = GMSMarker(position: coordinate)
    marker.map = mapView
    marker.title = "Привет"
    marker.snippet = "Красная площадь"
    self.marker = marker
}
```

Последнее полезное свойство у маркера – **groundAnchor:** оно позволяет задать точку маркера, которая привязана к переданным координатам. По умолчанию это центр нижнего края. Обратите внимание: у стандартного маркера есть хвостик, который указывает как раз на точку переданных координат. Но если у маркера хвостик сверху, **groundAnchor** должен находиться посередине верхнего края. Используйте этот параметр для калибровки позиции маркера относительно координат.

```
marker.groundAnchor = CGPoint(x: 0.5, y: 0.5)
```

Взаимодействие с картой

Последнее, чему мы научимся, – отслеживать нажатия на карте. Для этого нам необходимо установить для карты делегат **GMSMapViewDelegate.**

```
    func configureMap() {
// Создаём камеру с использованием координат и уровнем увеличения
        let camera = GMSCameraPosition.camera(withTarget: coordinate, zoom: 17)
// Устанавливаем камеру для карты
        mapView.camera = camera
// Устанавливаем делегат
        mapView.delegate = self
    }
```

```
extension MapViewController: GMSMapViewDelegate {
            func mapView(_ mapView: GMSMapView, didTapAt coordinate:
CLLocationCoordinate2D) {
        print(coordinate)
    }
}
```

У делегата довольно много методов. Мы не будем рассматривать их все (есть подробная документация), но метод обработки нажатия рассмотрим. Добавив код, приведённый выше, при нажатии на карту вы будете видеть в консоли координаты нажатия. Это немного скучно. Давайте добавим возможность ставить маркер по клику на карте:

```
// Свойство для хранения маркера, установленного по клику на карте
var manualMarker: GMSMarker?
```

```
extension MapViewController: GMSMapViewDelegate {
            func mapView(_ mapView: GMSMapView, didTapAt coordinate:
CLLocationCoordinate2D) {
// Если маркер уже создан, меняем его позицию
        if let manualMarker = manualMarker {
            manualMarker.position = coordinate
        } else {
// Если маркера нет, то создаём его
            let marker = GMSMarker(position: coordinate)
            marker.map = mapView
            self.manualMarker = marker
        }
    }
}
```

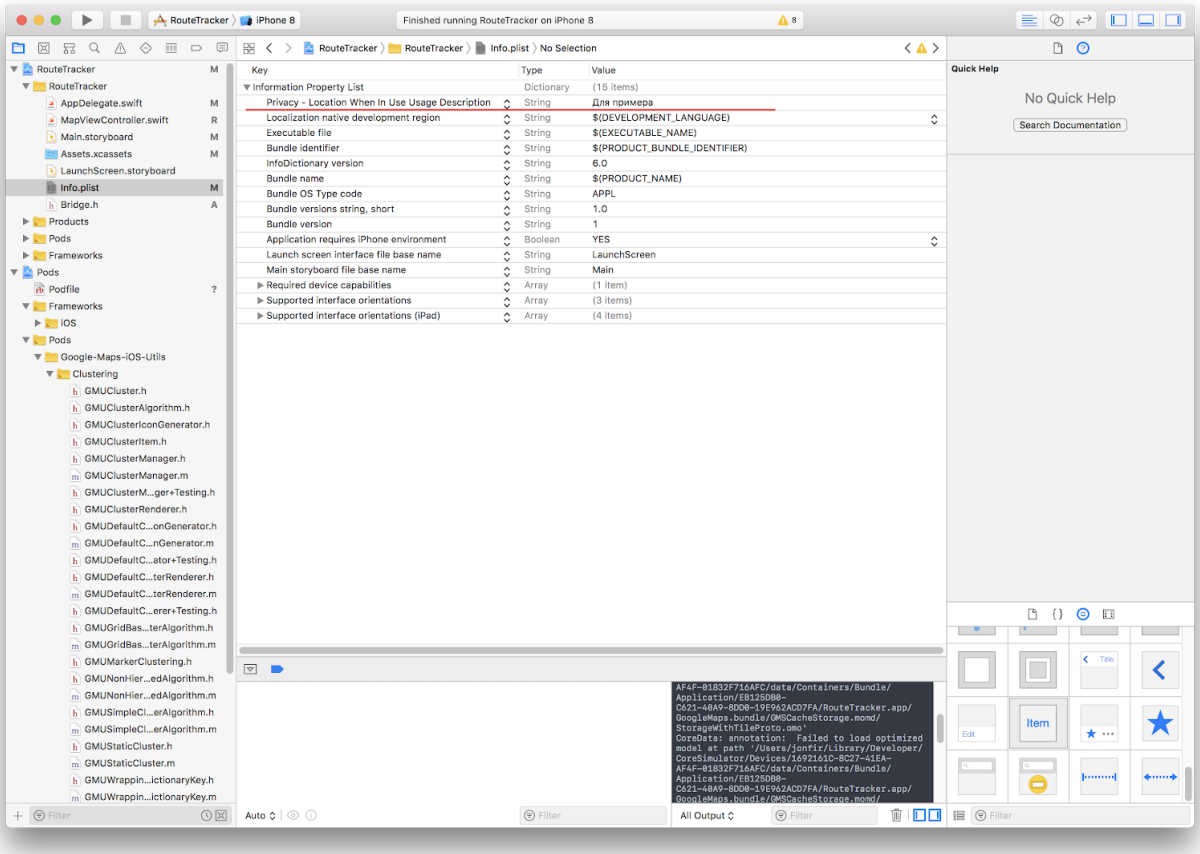
CoreLocation

Мы уже разобрались, как работать с картами, теперь поговорим, как отслеживать текущее местоположение. Новички часто полагают, что получение местоположения – одна из возможностей карты, но это не так. У нас есть специальный фреймворк, доступный «из коробки», – **CoreLocation**. Он отвечает за всё, что связано с определением местоположения устройства. Центральной фигурой

этого фреймворка является класс **CLLocationManager:** он может сообщать текущее местоположение, отслеживать его изменение, а также запрашивать разрешение на доступ к геоданным.

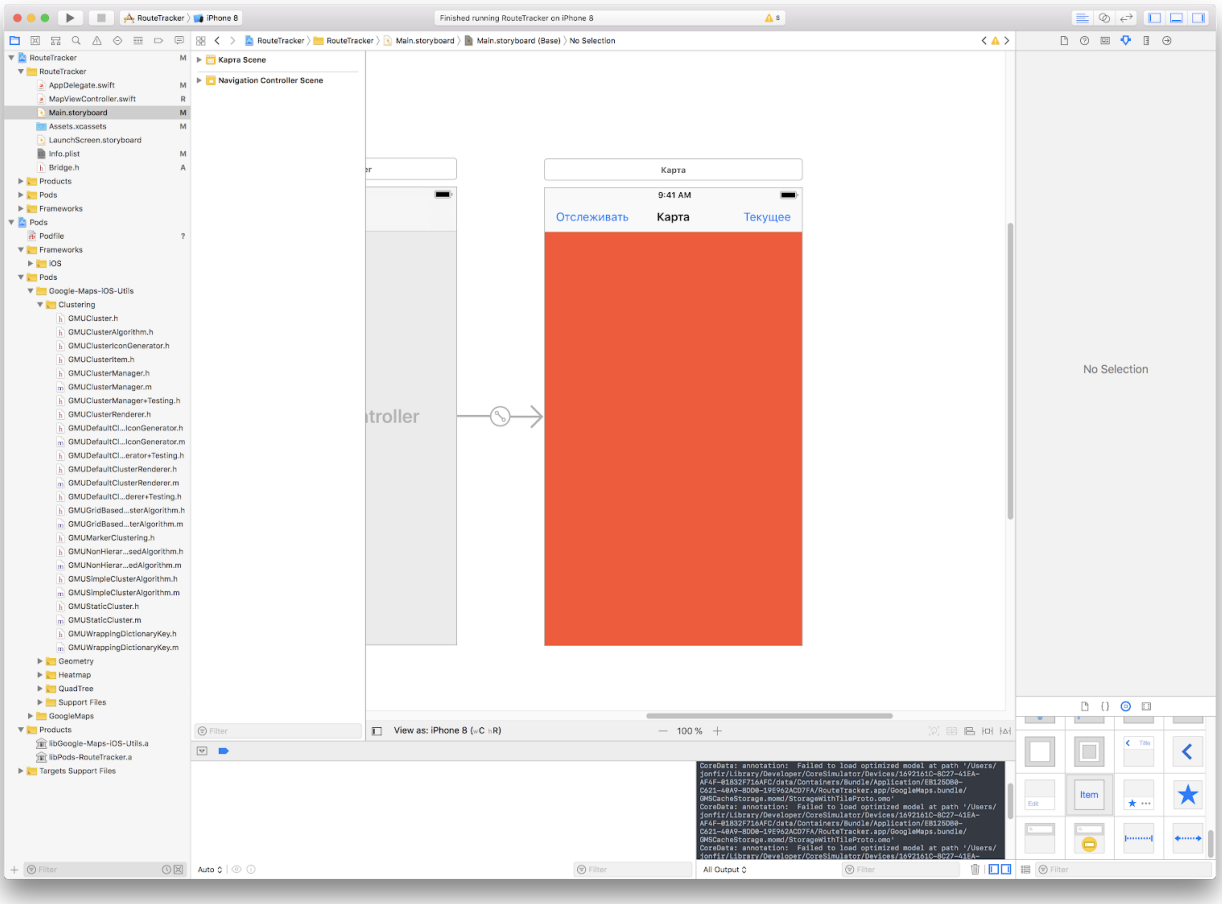
В современных iOS-устройствах огромное количество разных датчиков для определения текущего местоположения в пространстве: определяющие местоположение по спутникам GSM, GLONASS, магнитометры и прочие. Кроме того, местоположение может быть определено на основе источника Wi-Fi-сигнала и данных от сотовых вышек. Инструменты определения геопозиции отличаются и по скорости, и по точности работы. К счастью, нам не нужно выбирать, каким источником пользоваться: **CoreLocation** полностью инкапсулирует эту логику внутри себя, нам достаточно лишь выбрать желаемый уровень точности.

Прежде чем начать отслеживание текущего местоположения, надо добавить описание в файл **info.plist**, зачем нам нужно его отслеживать.



В этом примере мы будем работать с геолокацией, только пока приложение запущено, так что выбираем пункт Privacy – Location When In Use Usage Description. Автодополнение подсказывает множество различных описаний для доступа к геолокации. Из их описания легко понять, в каком случае какой выбрать. Также можно вообще ничего не указывать – тогда мы получим ошибку в консоли с подсказкой, какой пункт не указан в настройках.

Теперь изменим контроллер следующим образом: удалим всё, что касается установки маркеров, но добавим настройку **CLLocationManager.** Начнём с интерфейса.



Мы переименовали кнопки. Теперь первая отслеживает местоположение, а вторая запрашивает текущее местоположение.

```
import UIKit
import GoogleMaps
import CoreLocation

class MapViewController: UIViewController {

// Центр Москвы
      let coordinate = CLLocationCoordinate2D(latitude: 59.939095, longitude:
30.315868)
    var locationManager: CLLocationManager?

    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        configureMap()
        configureLocationManager()
    }

    func configureMap() {
// Создаём камеру с использованием координат и уровнем увеличения
        let camera = GMSCameraPosition.camera(withTarget: coordinate, zoom: 17)
// Устанавливаем камеру для карты
        mapView.camera = camera
    }
```

```
    func configureLocationManager() {
        locationManager = CLLocationManager()
        locationManager?.requestWhenInUseAuthorization()
    }

    @IBAction func updateLocation(_ sender: Any) {
    }

    @IBAction func currentLocation(_ sender: Any) {
    }
}
```

Мы добавили **import CoreLocation,** то есть подключили фреймворк. **var locationManager: CLLocationManager?** – свойство для хранения самого менеджера. **configureLocationManager** отвечает за создание менеджера и запрос доступа к геопозиции. Если это первый запрос, пользователь увидит модальное окно с вопросом, предоставить доступ или отказать. Если это сообщение вызывать повторно, вопроса мы уже не увидим вне зависимости от того, было ли получено разрешение или отказ.

requestWhenInUseAuthorization() запрашивает доступ к геопозиции только в момент использования приложения. Есть и другой метод – **requestAlwaysAuthorization().** Он запрашивает постоянный доступ, даже когда приложение не свёрнуто и не используется. Его мы изучим на следующем занятии.

Если запустить приложение сейчас, оно выдаст модальное окно:

Нажмём «Разрешить», иначе предоставлять доступ придётся уже в настройках приложения.

Теперь давайте получим текущие координаты. Для этого нам придётся выполнить ряд шагов.

Назначим делегат для менеджера:

```
locationManager?.delegate = self
```

Реализуем два обязательных метода делегата:

```
extension MapViewController: CLLocationManagerDelegate {
        func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
        print(locations.first)
    }

     func locationManager(_ manager: CLLocationManager, didFailWithError error:
Error) {
        print(error)
    }
}
```

Добавим метод запроса текущей позиции:

```
  @IBAction func currentLocation(_ sender: Any) {
      locationManager?.requestLocation()
  }
```

Чтобы наше местоположение отображалось в симуляторе, надо включить его эмуляцию. При запущенном приложении нажмите на кнопку симуляции и выберите местоположение.

Теперь при нажатии на кнопку мы получим текущее местоположение в консоли.

Давайте также настроим отслеживание изменения местоположения устройства. Для этого есть метод **startUpdatingLocation().**

```
@IBAction func updateLocation(_ sender: Any) {
    locationManager?.startUpdatingLocation()
}
```

Если мы его вызовем сейчас, то получим текущие координаты только один раз, а можем и вообще ничего не получить. Дело в том, что, пока устройство не движется, обновлений нет. Нужно включить симуляцию движения. Для этого запустите приложение, выберите симулятор и в панели инструментов установите нужный тип симуляции.

Мы начнём получать сообщения о изменении текущих координат.

CLGeocoder

Еще один востребованный инструмент – геокодер. Он позволяет получать адрес по координатам и наоборот. Работать с ним довольно просто: нужно создать экземпляр класса **CLGeocoder** и вызвать у получившегося объекта метод **reverseGeocodeLocation(),** передав в качестве аргумента координаты.

```
        func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
        guard let location = locations.last else { return }
        geocoder.reverseGeocodeLocation(location) { places, error in
            print(places?.first)
        }
    }
```

В результате в консоли будет отображаться адрес.

Практическое задание

На этом курсе мы познакомимся с современными инструментами и технологиями, которые используют iOS-разработчики. Чтобы потренироваться и проверить, как вы усвоили новую информацию, вы разработаете простое приложение, отслеживающее передвижения пользователя.

1. Создать новое приложение.
2. Добавить в него фреймворк Google Maps.
3. Добавить на главный экран карту.
4. Добавить **CLLocationManager** в контроллер.
5. Добавить отслеживание изменения текущего местоположения.
6. При изменении местоположения сдвигать карту так, чтобы ваше местоположение было в её центре.
7. При изменении местоположения добавлять стандартный маркер на карту.
8. Удалять маркеры не нужно: в результате у вас получится дорожка из маркеров, показывающая ваш путь.

Дополнительные материалы

1. Документация Google Maps.

Используемая литература

1. Документация Google Maps.