Урок 2

Background mode

Работа приложения в фоновом режиме

Базовые приёмы работы в фоне

В отличие от Android, в iOS нельзя заставить приложение работать после того, как оно было свёрнуто, и тем более после того, как оно было закрыто. С одной стороны, это ограничивает разработчика, с другой – даёт преимущества пользователям. Вы можете быть уверены, что приложение не потребляет ресурсы телефона, если им не пользуются. Но есть некоторые виды задач, которые не могут быть остановлены, – например, построение маршрута или работа телефонии. Для каждой из таких задач есть свои способы заставить приложение работать в фоне. К сожалению, мы не сможем рассмотреть их все на этом уроке. Мы познакомимся только с базовыми принципами, подходящими для любого приложения, а также с фоновым использованием геолокации.

BackgroundTask

Если свернуть приложение, не дождавшись завершения действия, задача прервётся. Большинство из них завершаются почти мгновенно. Но есть как минимум две часто встречающихся задачи, для завершения которых может не хватить времени. Первая – запуск таймера, вторая – отправка сетевого запроса.

Если вы запустите таймер, а затем свернёте приложение, он остановится, и вы никогда не дождётесь его срабатывания. Если же вы повторяете какое-либо действие по таймеру (например, проверяете появление необходимых данных), то ждать его повторения тоже не стоит.

Если вы запустили сетевую задачу, то потеряете ответ. Это может привести к большим проблемам, например, если результат должен быть сохранён в базе данных и использован в дальнейшем. Но даже если вы можете обойтись без этого ответа, вы получите ошибку загрузки, в результате её придётся обрабатывать дополнительно, что не всегда удобно.

Есть ли у нас выход из такой непростой ситуации? Разумеется. В iOS есть специальный метод, который может сообщить операционной системе, что нам нужно немного времени для завершения процессов – **BackgroundTask.**

Давайте проведём небольшой эксперимент: запустим таймер, который выводит сообщения в консоль, откроем первый же контроллер и добавим в него свойство **timer**, а также метод, его запускающий.

```
var timer: Timer?
 override func viewDidLoad() {
     super.viewDidLoad()

     configureTimer()
 }

 func configureTimer() {
     timer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true) { _ in
         print(Date())
     }
 }
```

Если запустить приложение, в консоли начнут появляться временные метки, но как только мы свернём приложение (это можно сделать с помощью сочетания клавиш **cmd+shift+H)**, таймер сразу остановится. Попросим у системы немного дополнительного времени:

```
var timer: Timer?
var beginBackgroundTask: UIBackgroundTaskIdentifier?
```

```
override func viewDidLoad() {
    super.viewDidLoad()

    configureTimer()
}

func configureTimer() {
    beginBackgroundTask = UIApplication.shared.beginBackgroundTask { [weak self]
in
        guard let strongSelf = self else { return }
        UIApplication.shared.endBackgroundTask(strongSelf.beginBackgroundTask!)
        strongSelf.beginBackgroundTask = UIBackgroundTaskInvalid
    }
    timer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true) { _ in
        print(Date())
    }
}
```

Мы создали свойство для хранения идентификатора нашей фоновой задачи – **beginBackgroundTask** и создали эту задачу перед запуском таймера. Обратите внимание, что у задачи есть замыкание, в котором мы вызываем метод **endBackgroundTask,** передавая ему идентификатор задачи, а также присваиваем свойству идентификатора константу **UIBackgroundTaskInvalid.** Это сделано, чтобы задача гарантированно завершилась. Код выглядит бессмысленным, но он обязателен.

Если свернуть приложение сейчас, таймер не остановится. Это довольно простой трюк, позволяющий выиграть немного времени. Это не значит, что приложение теперь будет работать вечно. Раньше запуск фоновой задачи гарантированно давал 10 минут дополнительного времени, сейчас же этот интервал не определён, но всё же не вечен. Рассчитывайте на 10 минут.

Ещё раз взгляните на код. Само название этой техники заставляет думать, что вы запускаете фоновую задачу, на эту же мысль наводит наличие замыкания у метода **beginBackgroundTask.** Кажется, что именно в этом замыкании и надо совершать необходимые нам действия, но это не так. **beginBackgroundTask** запускается независимо от остального кода, физически она никак не связана с вашими реальными задачами. После её запуска, если приложение будет свернуто, вы получите время.

Правило работы с фоновыми задачами обязует вас не только запускать их, но и уведомлять операционную систему, что вы завершили свои действия и работать в фоновом режиме больше не нужно. Наш пример подходит для демонстрации того, как работает приложение, но на практике использовать **beginBackgroundTask** таким образом нельзя. Мы просто запустили таймер, который должен работать неограниченно долго, а, как мы помним, у нас есть всего 10 минут, плюс неопределённый промежуток времени. Значит, фоновая задача здесь не поможет. Давайте перепишем пример корректно:

```
// Таймер
    var timer: Timer?
// Время, когда таймер был запущен
    var startTime: Date?
// Интервал, в течение которого должен работать таймер, в секундах
    let timeInterval: TimeInterval = 180
// Идентификатор фоновой задачи
    var beginBackgroundTask: UIBackgroundTaskIdentifier?

    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        configureMap()
```

```
        configureLocationManager()
        configureTimer()
    }

    func configureTimer() {
// Создание фоновой задачи на случай, если во время работы таймера приложение
// будет свёрнуто
        beginBackgroundTask = UIApplication.shared.beginBackgroundTask { [weak
self] in
            guard let beginBackgroundTask = self?.beginBackgroundTask else {
return }
// Гарантированная очистка фоновой задачи при её прекращении
            UIApplication.shared.endBackgroundTask(beginBackgroundTask)
            self?.beginBackgroundTask = UIBackgroundTaskInvalid
        }
// Запоминаем время запуска таймера
        startTime = Date()
// Запускаем таймер
        timer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true) { [weak
self] _ in
// Используем таймер, для примера выводим сообщение в консоль
            print(Date())

            guard
                let startTime = self?.startTime,
                let timeInterval = self?.timeInterval,
                let beginBackgroundTask = self?.beginBackgroundTask
            else {
                return
            }
// На каждом витке проверяем, сколько прошло времени с момента запуска
            let leftSeconds = Date().timeIntervalSince1970 -
startTime.timeIntervalSince1970
// Если таймер работает дольше, чем необходимо, то
            if leftSeconds >= timeInterval {
// Останавливаем таймер
                self?.timer?.invalidate()
// Очищаем память
                self?.timer = nil
// Останавливаем фоновую задачу, тем самым сообщая операционной системе, что
мы закончили
                UIApplication.shared.endBackgroundTask(beginBackgroundTask)
// Безопасно очищаем указатель на фоновую задачу
                self?.beginBackgroundTask = UIBackgroundTaskInvalid
            }
        }
    }
```

Этот пример вполне корректен: нам необходимо выполнять некое действие, например, показывать пользователю обратный отсчёт в течение трёх минут, и мы запускаем таймер, а также фоновую задачу, чтобы его работа не прервалась.

Ещё одно правило работы с фоновыми задачами гласит, что на каждое действие, которое может потребовать работы в фоне, вы должны запускать отдельную фоновую задачу и останавливать её, как только это действие закончится. Ваше приложение будет работать, если хотя бы одна фоновая задача активна, и завершится, как только завершится последняя из них.

Таймер и жизненный цикл приложения

Невозможно, чтобы таймер работал всегда, но мы можем запускать его после того, как приложение снова будет развёрнуто. В большинстве случаев приложение не закрывают, а сворачивают, и если необходимо продолжить работу, разворачивают. Многие думают, что это можно отслеживать в методе **viewWillAppear**, но это не так. Он не срабатывает при выходе из фонового режима. Но у нас есть специальные уведомления, которые помогут отслеживать это событие. Сейчас мы вернёмся к самой первой версии нашего таймера и перепишем её так, чтобы при сворачивании в фон таймер корректно останавливался, а при возобновлении работы – корректно запускался заново.

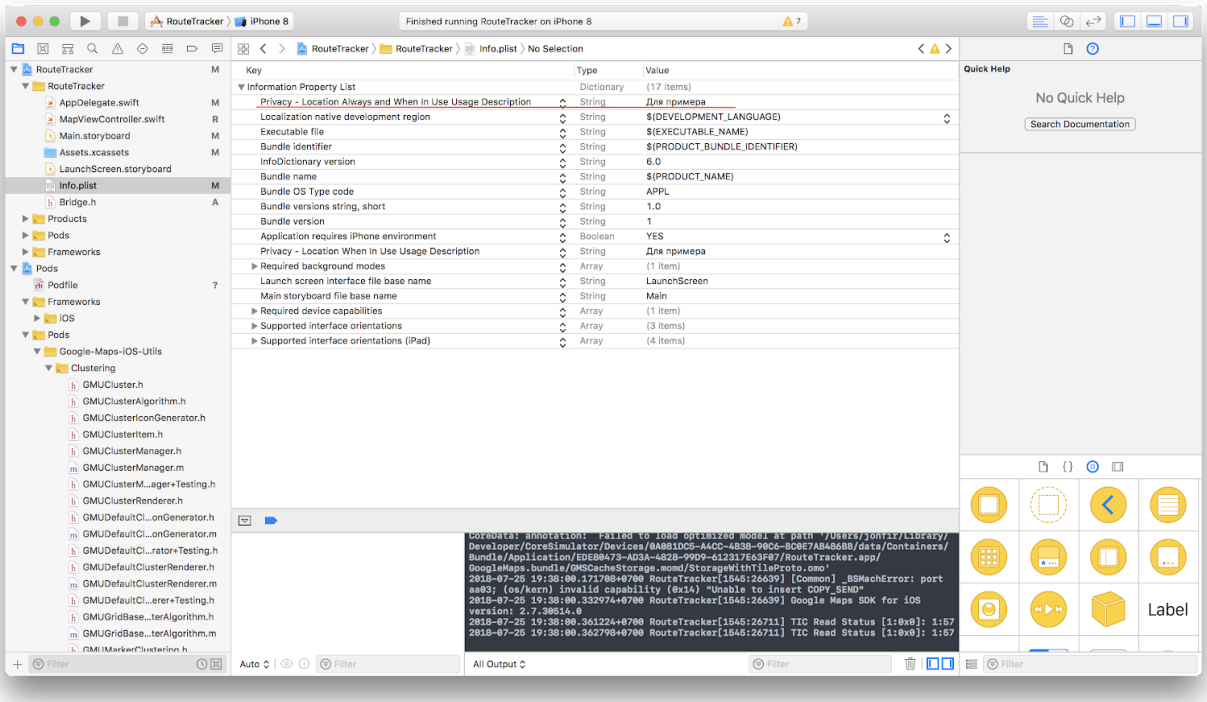
```
override func viewDidLoad() {
    super.viewDidLoad()

    configureTimer()
}

func configureTimer() {

    timer = Timer.scheduledTimer(withTimeInterval: 1, repeats: true) { _ in
        print(Date())
    }

    NotificationCenter.default.addObserver(
        forName: NSNotification.Name.UIApplicationDidEnterBackground,
        object: nil,
        queue: OperationQueue.main) { [weak self] _ in
            self?.timer?.invalidate()
            self?.timer = nil
    }

    NotificationCenter.default.addObserver(
        forName: NSNotification.Name.UIApplicationWillEnterForeground,
        object: nil, queue: OperationQueue.main) { [weak self] _ in

            self?.configureTimer()
    }
}
```

Кроме обычной настройки таймера, мы добавляем два обработчика для событий ухода в фон и выхода из фона.

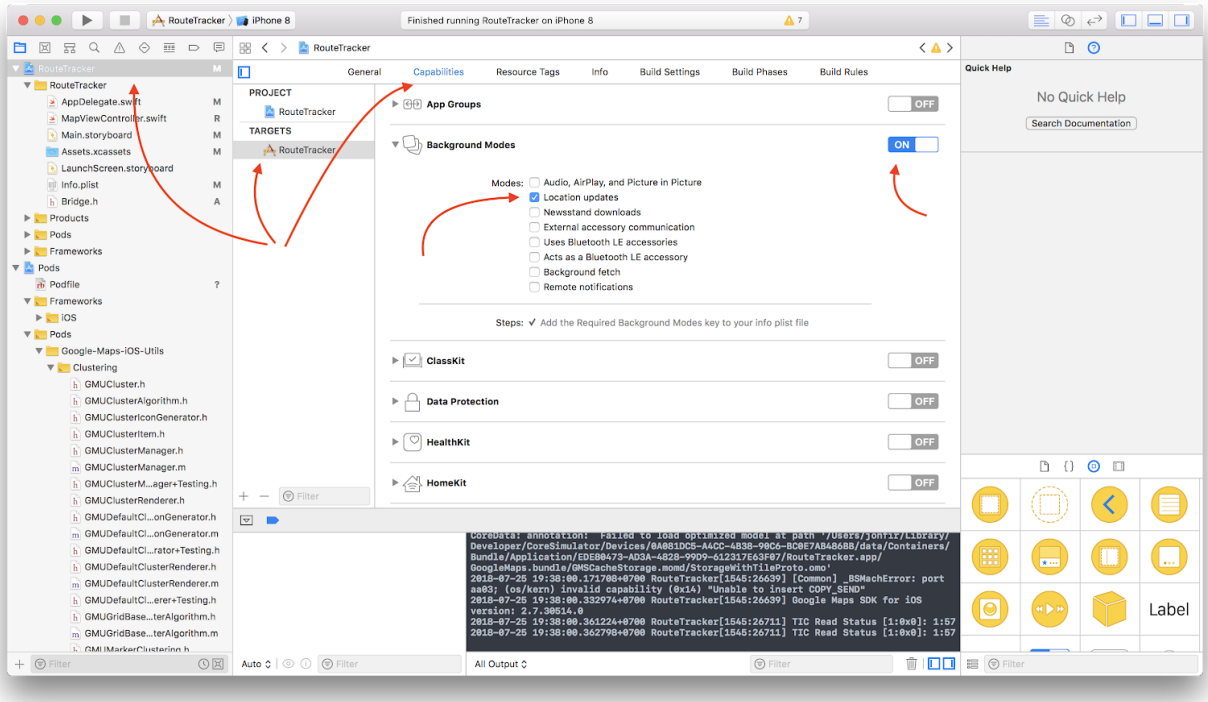
Важно понимать, что этот код не использует фоновые задачи, и в фоне приложение работать не будет. Но при этом оно будет корректно работать при выходе из фона. Код довольно простой и не требует специфических знаний и технологий, но тем не менее многие новички забывают про такие особенности и считают, что если что-то начало работать на контроллере, то оно будет работать и дальше. Более того, они не проверяют работу таких задач при разработке. В итоге пользователи сталкиваются с очень неприятными ошибками.

Фоновая работа с местоположением

Так как основная тема домашней работы – работа с картами, давайте посмотрим, что же необходимо для получения местоположения в фоне. Добавим описание в **Info.plist.**



Включим работу в фоне, выбрав отслеживание местоположения в настройках проекта.

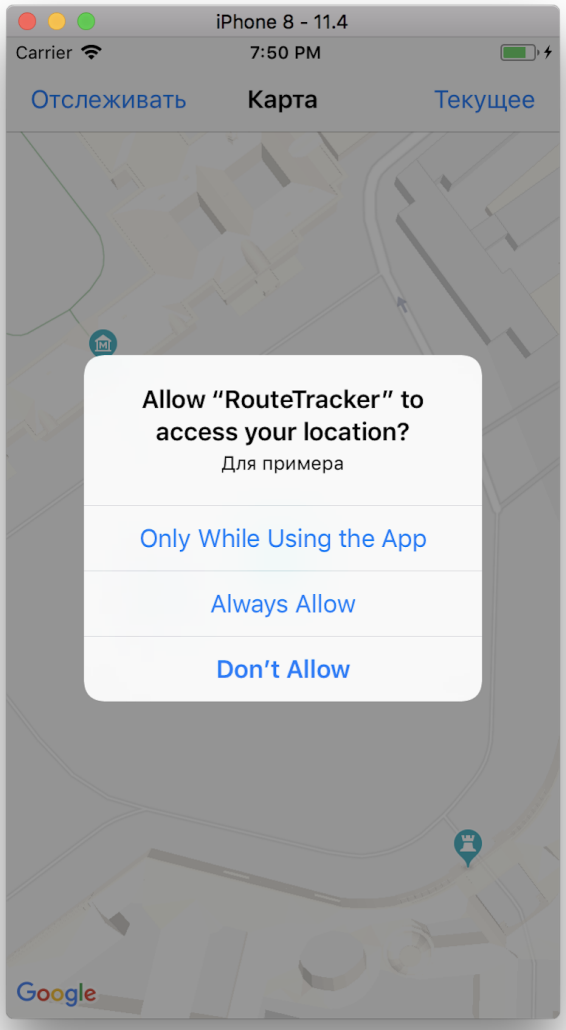


Запросим разрешения на работу геолокации в фоне у пользователя и установим флаг **allowsBackgroundLocationUpdates.**

```
func configureLocationManager() {
    locationManager = CLLocationManager()
    locationManager?.delegate = self
    locationManager?.allowsBackgroundLocationUpdates = true
    locationManager?.requestAlwaysAuthorization()
```

```
    }
```

Мы провели все подготовительные мероприятия и теперь можем запустить приложение.



Обратите внимание: у нас теперь три варианта. Пользователь может разрешить постоянное использование геолокации, запретить его и разрешить, только пока приложение используется, несмотря на то, что мы такой вариант не просили. Это особенности последних версий iOS. Мы выберем вариант **Always Allow.**

Если сейчас запустить отслеживание и свернуть приложение, мы всё ещё будем получать координаты. Кажется, что это просто, но на самом деле нюансов ещё много.

У **CLLocationManager** есть неочевидное поведение: он останавливает свою работу, если приложение находится в фоне и телефон некоторое время не меняет местоположения. Например, вы можете постоять на долгом светофоре – слежение прекратится и не будет возобновлено, пока вы не развернёте приложение. Чтобы этого не допустить, нужно установить флаг **pausesLocationUpdatesAutomatically** в **false.** Это тоже не гарантирует постоянную работу: пользователь может закрыть приложение, и самостоятельно оно уже не запустится. Но есть один трюк, который позволяет бороться и с этой проблемой. Вы можете попросить систему запускать приложение при определённых изменениях местоположения:

- **startMonitoringSignificantLocationChanges** – телефон переместился на значительное расстояние;
- **startMonitoringVisits** – прибытие в часто посещаеме место, например, домой;
- **startMonitoring** – вход или выход из определённой области.

Очевидно, что **startMonitoringVisits** нам совсем не подходит. **startMonitoring** – тоже не очень удобное решение, так как надо постоянно определять регион, да и площадь достаточно велика. А вот **startMonitoringSignificantLocationChanges** – то что нужно. Как указано в документации, это уведомление приходит при перемещении устройства на 500 метров и более, но при определённых условиях может сработать и раньше.

```
func configureLocationManager() {
    locationManager = CLLocationManager()
    locationManager?.delegate = self
    locationManager?.allowsBackgroundLocationUpdates = true
    locationManager?.pausesLocationUpdatesAutomatically = false
    locationManager?.startMonitoringSignificantLocationChanges()
    locationManager?.requestAlwaysAuthorization()
}
```

Последнее, о чём нам надо задуматься – это энергопотребление. Чем больше точность геолокации, тем больше тратится заряд, и есть вероятность, что наше приложение будет удалено с устройства. Есть параметр **desiredAccuracy** – точность определения местоположения. Указывается в метрах. Как правило, вам не нужно отслеживать перемещение с точностью до метра, хватит и 100 метров, а может, и километра. Но если искать оптимальное значение не хочется, можно воспользоваться системными константами:

1. **kCLLocationAccuracyBestForNavigation.**
2. **kCLLocationAccuracyBest.**
3. **kCLLocationAccuracyNearestTenMeters.**
4. **kCLLocationAccuracyHundredMeters.**
5. **kCLLocationAccuracyKilometer.**
6. **kCLLocationAccuracyThreeKilometers.**

Это шесть часто используемых параметров. Мы используем **kCLLocationAccuracyNearestTenMeters.**

Теперь, когда мы умеем собирать точки в фоне, осталось придумать, что с ними делать. Нарисуем маршрут: для этого воспользуемся средствами рисования линий на карте – **GMSPolyline.**

Добавим два свойства для хранения объекта маршрута и объекта, представляющего его путь:

```
var route: GMSPolyline?
var routePath: GMSMutablePath?
```

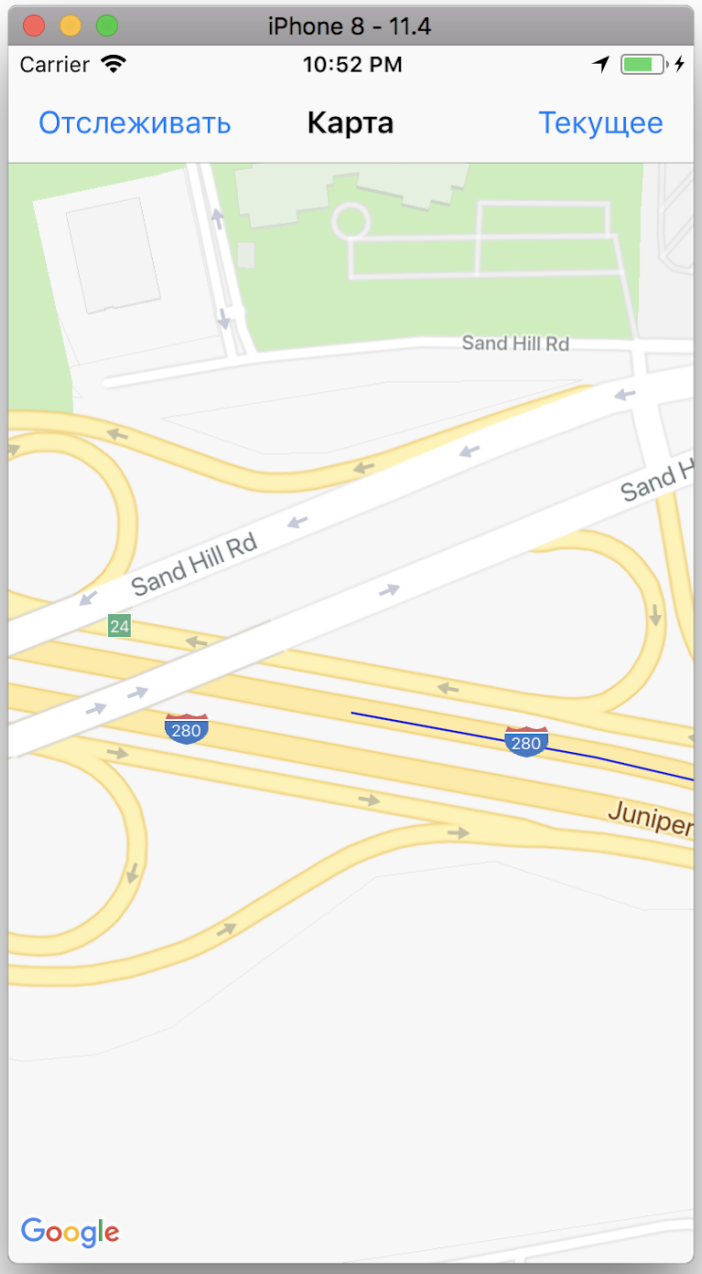
Изменим метод начала отслеживания так, чтобы он создавал или заменял уже имеющиеся объекты линии маршрута и пути (точек) маршрута:

```
    @IBAction func updateLocation(_ sender: Any) {
// Отвязываем от карты старую линию
        route?.map = nil
// Заменяем старую линию новой
        route = GMSPolyline()
// Заменяем старый путь новым, пока пустым (без точек)
        routePath = GMSMutablePath()
// Добавляем новую линию на карту
        route?.map = mapView
// Запускаем отслеживание или продолжаем, если оно уже запущено
        locationManager?.startUpdatingLocation()
    }
```

Осталось только добавить новые точки в маршрут. Для красоты и удобства наблюдения будем перемещать камеру к точке, только что добавленной в маршрут.

```
    func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
// Берём последнюю точку из полученного набора
        guard let location = locations.last else { return }
// Добавляем её в путь маршрута
        routePath?.add(location.coordinate)
// Обновляем путь у линии маршрута путём повторного присвоения
        route?.path = routePath

// Чтобы наблюдать за движением, установим камеру на только что добавленную
// точку
        let position = GMSCameraPosition.camera(withTarget: location.coordinate,
zoom: 17)
        mapView.animate(to: position)
    }
```

Теперь, если запустить приложение, мы увидим маршрут, по которому движемся. Он не будет прерываться, даже если мы свернём приложение.



Практическое задание

На основе задания предыдущего урока.

1. Настроить слежение за перемещением в фоне.
2. Добавить кнопки «Начать новый трек» и «Закончить трек».
3. При нажатии на «Начать новый трек»:
    a. Запускается слежение.
    b. Создаётся новая линия на карте или заменяется предыдущая.

4. При получении новой точки она добавляется в маршрут.
5. Добавить в приложение базу данных Realm.
6. При нажатии на «Закончить трек»:
    a. Завершается слежение.
    b. Все точки маршрута сохраняются в базу данных.
    c. Прежде чем сохранить точки из базы, необходимо удалить предыдущие точки.
7. Добавить кнопку «Отобразить предыдущий маршрут».
8. При нажатии на «Отобразить предыдущий маршрут».
    a. Если в данный момент происходит слежение, то появляется уведомление о том, что сначала необходимо остановить слежение. С кнопкой «ОК», при нажатии на которую останавливается слежение, как если бы пользователь нажал на «Закончить трек».
    b. Загружаются точки из базы.
    c. На основе загруженных точек строится маршрут.
    d. Фокус на карте устанавливается таким образом, чтобы был виден весь маршрут.

P.S. В документации к картам можно найти методы и свойства, которые помогут вам:

1. Остановить слежение.
2. Получить все точки из объекта **GMSMutablePath.**
3. Установить фокус карты точно так, чтобы она отображала маршрут, описанный объектом **GMSMutablePath.**

Чтение документации – важный навык, развивайте его!

Дополнительные материалы

1. Документация Google Maps.

Используемая литература

1. Документация Google Maps.