ФГБОУ ВО «ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАТИКИ И МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Отчет по дисциплине «Верификация программного обеспечения»
Тестирование мобильного приложения для системы лояльности в сфере туризма

Выполнила:
студентка 6-го курса
группы №22608
Шевцова Кристина Олеговна

Преподаватель:
к.ф-м.н., доцент К. А. Кулаков

Петрозаводск – 2018

Оглавление

Объект тестирования

В рамках курса «Верификация программного обеспечения» будет рассматриваться мобильный сервис для системы лояльности в сфере туризма. Целью сервиса является предоставление персонализированных предложений для пользователей, основываясь на их перемещении.

Отслеживание перемещений пользователя осуществляется с использованием следующих технологий:

1. Geofencing – технология, позволяющая установить некоторую область (заранее заданный радиус) вокруг точки на карте города (по её географическим координатам), во время входа или выхода из которой мобильное приложение будет получать сигнал на выполнение заранее предопределенных действий. Такие географические области будем называть geofence.
2. iBeacon – технология, уведомляющая мобильное приложение о попадании/выходе из радиуса действия Bluetooth-маяка (beacon). В отличие от geofencing чаще используется для навигации внутри помещений.
3. ImageTargets – изображения, которые можно отсканировать с помощью камеры смартфона.

Когда пользователь попадает в радиус действия geofence/beacon или сканирует ImageTarget, мобильное приложение получает сообщение о произошедшем событии и предоставляет пользователю предложение, соответствующее данному событию.

Сервис состоит из сервера и мобильного приложения. Объектом тестирования является мобильное приложение, разработанное для операционной системы iOS версии 11.0 и выше, написанное на языке Swift 4.0.

Основными экранами приложения являются:

1. Главный экран приложения.

Экран представляет собой список карточек, где каждая карточка – это некоторое персональное предложение, предоставленное пользователю. Карточки могут

быть разных типов. В зависимости от типа карточки определяется действие, происходящее по нажатию на карточку: открытие ссылки в браузере, открытие ссылки в WebView, воспроизведение видео.

С экрана возможен переход на экран выбора города, экран карты и экран сканирования изображений.

2. Экран выбора города.

При первом запуске приложения пользователь попадает на экран выбора города и выбирает интересующий его город. После этого выбранный город сохраняется в памяти устройства. При последующих запусках приложения экран выбора города автоматически пропускается, и пользователь сразу попадает на главный экран приложения.

В зависимости от выбранного города приложение загружает список всех geofences, beacons и ImageTargets для данного города, а также сценарии к ним.

С экрана возможен переход на главный экран.

3. Экран сканирования.

Экран-камера. При наведении камеры на ImageTarget, происходит распознавание изображения. Если есть совпадение с ImageTarget, доступных в выбранном городе, то в список карточек добавится новая карточка, соответствующая отсканированному ImageTarget.

С экрана возможен переход на главный экран.

4. Экран карты.

В приложении доступен экран карты с обозначением расположения всех geofences, beacons и ImageTargets, доступных в выбранном городе. На экране карты пользователь может построить маршрут от текущего местоположения до какого-либо маркера посредством навигации, предоставляемой сервисом Google Maps.

С экрана возможен переход на главный экран, приложение или сайт Google Maps.

Помимо основных экранов важными модулями приложения являются:

1. APIDataManager.

Класс, отвечающий за взаимодействие с сервером. Функции класса вызываются из контроллеров. Класс получает или отправляет данные на сервер. В случае получения данных с сервера, APIDataManager передаёт их в класс соответствующей модели для того, чтобы перевести полученные данные в модель. Итоговую модель данных APIDataManager возвращает в вызывающий контроллер.

2. LocationManager.

Класс, работающий с геопозицией пользователя. В классе устанавливаются geofences и beacons для мониторинга. Также класс работает в фоновом режиме и отвечает за дальнейшие действия после получения события о входе/выходе пользователя в зону geofence или beacon.

3. CoreDataManager.

Класс, осуществляющий основные операции (CRUD – Create, Read, Update, Delete) по взаимодействию с базой данных.

Архитектура приложения представлена на рисунке 1.

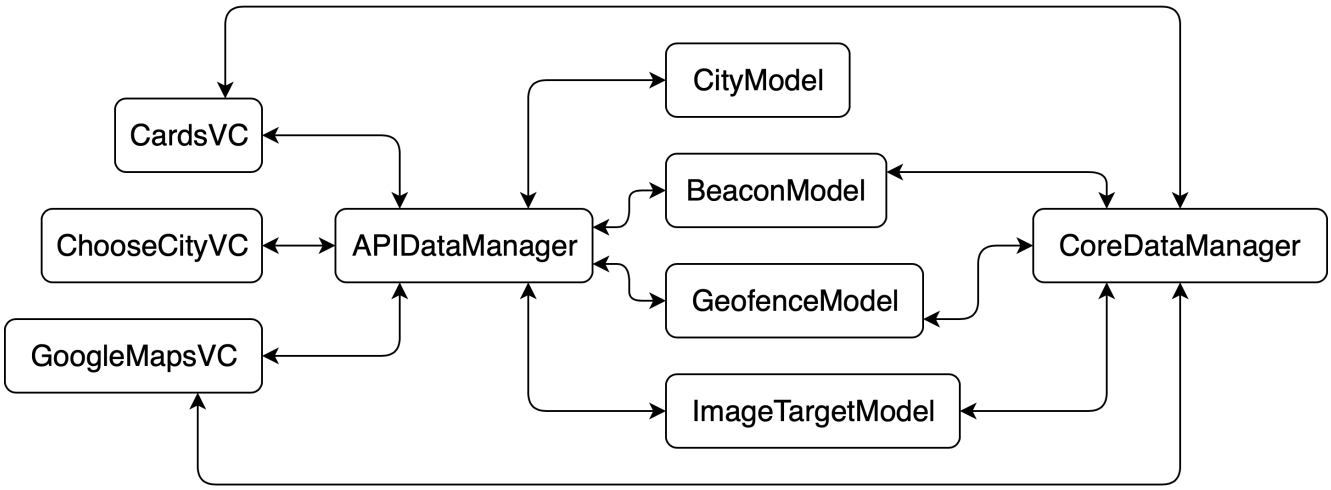

Рисунок 1 – Архитектура приложения

Функциональные требования к объекту тестирования

Тестируемое приложение должно реализовывать следующий набор функциональных требований:

1. Главный экран приложения:

    1.1. Отображение списка карточек, полученных из базы данных, для выбранного города.

    1.2. Удаление карточки из списка и базы данных по свайпу влево.

    1.3. По нажатию на карточку осуществление сценария для данной карточки (открытие ссылки в браузере/открытие ссылки в WebView/воспроизведение видео).

    1.4. Добавление новых карточек в начало списка в зависимости от получаемых сообщений от других модулей.

    1.5. Получение с сервера списка всех geofences, beacons и ImageTargets для выбранного города, его разбор, запись в базы данных и установление мониторинга geofences и beacons.

2. Экран выбора города:

    2.1. Получение с сервера списка городов, его разбор и отображение этого списка на экране.

3. Экран карты:

    3.1. Получение из базы данных всех geofences, beacons и ImageTargets и отображение их на карте.

    3.2. Отображение текущего местоположения пользователя.

    3.3. По нажатию на маркер конкретного объекта на карте построение маршрута от текущего местоположения пользователя до выбранного объекта в приложении (при наличии) или на сайте Google Maps.

4. Экран сканирования:

    4.1. Сканирование ImageTarget.

    4.2. Отправка сообщения о том, что конкретный ImageTarget был отсканирован пользователем.

5. APIDataManager:

    5.1. Получение команд от контроллеров.

    5.2. Отправка запросов на сервер.

    5.3. Передача JSON-файлов с сервера на разбор в модели.

    5.4. Возврат итоговых моделей обратно в контроллеры.

6. LocationManager:

    6.1. Установка мониторинга geofences и beacons.

    6.2. Отслеживание геопозиции пользователя, в том числе в фоновом режиме.

    6.3. Уведомление других модулей о том, что пользователь вошел/покинул радиус действия geofence или beacon.

7. CoreDataManager:

    7.1. Осуществление CRUD-операций с базой данных.

Описание модулей

Приложение разработано с использованием паттерна MVC (Model-View-Controller). Основными частями приложения являются:

1. CardsVC.

Назначение: Контроллер главного экрана приложения с выбором карточек.

Функции:

- `func getGeofencesBeaconsAndImageTargets()` – получает список geofences, beacons и ImageTargets, а также сценарии к ним из APIDataManager и обновляет данные в базе данных.
- `func showCards()` – получает список карточек из базы данных и подставляет данные во View.

2. ChooseCityVC.

Назначение: Контроллер экрана выбора города.

Функции:

- `func showCities()` – получает список городов из APIDataManager и подставляет данные во View.

3. GoogleMapsVC.

Назначение: Контроллер экрана с картой.

Функции:

- `func setUpMarkers()` – получает все доступные geofences, beacons и ImageTargets из базы данных и создает маркеры разных цветов для разных типов объектов.

4. APIDataManager.

Назначение: Модуль отвечает за взаимодействие с сервером.

Функции:

- `func initBeaconsAndGeofencesAndImageTargets(completionHandler: @escaping ErrorType? -> ())` – получает с сервера список

всех доступных geofences, beacons и ImageTargets. В случае успешного выполнения запроса инициализирует мониторинг beacons и geofences, иначе – возвращает ошибку.

- `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())` – получает список всех городов с сервера. Возвращает список городов или ошибку.

5. CityModel.

Назначение: Модель данных для города на экране выбора города.

Функции:

- `func getCities(from data: NSDictionary?) -> [CityModel]?` – получает список городов в формате JSON от APIDataManager, преобразует его в CityModel и возвращает в APIDataManager.

6. BeaconModel.

Назначение: Модель данных для beacon.

Функции:

- `func findTriggeringBeacon(with name: String)` – принимает название распознанного beacon и добавляет соответствующую карточку в базу данных.
- `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?` – получает список beacons в формате JSON от APIDataManager, преобразует его в BeaconModel и возвращает в APIDataManager.

7. GeofenceModel.

Назначение: Модель данных для geofence.

Функции:

- `func findTriggeringGeofence(with name: String)` – принимает название распознанного geofence и добавляет соответствующую карточку в базу данных.

- `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?` – получает список geofences в формате JSON от APIDataManager, преобразует его в GeofenceModel и возвращает в APIDataManager.

8. ImageTargetModel.

Назначение: Модель данных для ImageTarget.

Функции:

- `func findTriggeringImageTarget(with vuforiaId: String)` – принимает идентификатор распознанного ImageTarget и добавляет соответствующую карточку в базу данных.
- `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?` – получает список ImageTargets в формате JSON от APIDataManager, преобразует его в ImageTargetModel и возвращает в APIDataManager.

Объект и стратегия тестирования

Общая информация

Для тестирования будет использоваться стандартный инструментарий, встроенный в среду разработки Xcode версии 10.0. Тесты будут разрабатываться на языке Swift 4.0.

Стратегия блочного тестирования

Блочное тестирование будет проводиться для network-слоя приложения, а также для классов, отвечающих за разбор JSON-файлов, получаемых с сервера.

Следующие функции подлежат блочному тестированию:

1. `func initBeaconsAndGeofencesAndImageTargets(completionHandler: @escaping ErrorType? -> ())`
2. `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`
3. `func getCities(from data: NSDictionary?) -> [CityModel]?`
4. `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`
5. `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`
6. `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`

Функции, отвечающие за отображение данных на экране и взаимодействие с базой данных, не подлежат блочному тестированию (они не связаны с работой с сетью и с разбором JSON-файлов). Работоспособность этих функций будет проверена в рамках аттестационного тестирования.

Стратегия интеграционного тестирования

Цель интеграционного тестирования – удостовериться в корректности совместной работы элементов приложения (каждый модуль не только выполняет свои функции, но и взаимодействует с другими модулями без ошибок).

В данной работе выбран принцип восходящего тестирования. Сначала будет тестироваться самый нижний уровень системы. Затем постепенно к более низким уровням будут интегрироваться более высокоуровневые модули.

Этапы интеграции

1. Работа с geofences, beacons и ImageTargets.

Методы:

APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping  ErrorType? -> ())`

BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`

GeofenceModel: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`

ImageTargetModel: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`

CardsVC: `func getGeofencesBeaconsAndImageTargets()`

CardsVC: `func showCards()`

GoogleMapsVC: `func setUpMarkers()`

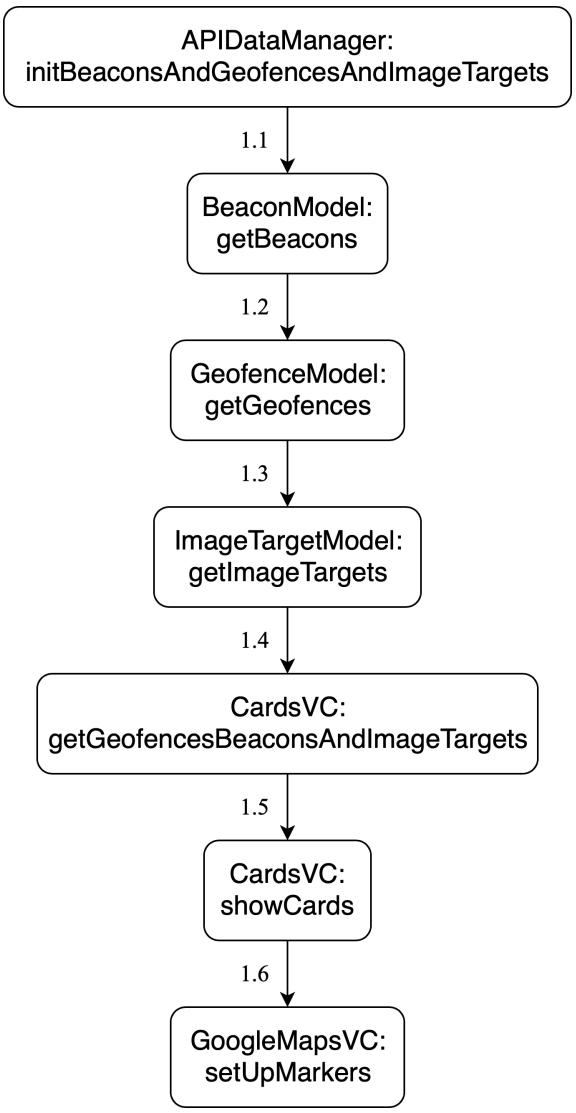

Рисунок 2 – Работа с geofences, beacons и ImageTargets

2. Работа с городами.

Методы:

APIDataManager: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

CityModel: `func getCities(from data: NSDictionary?) -> [CityModel]?`

ChooseCityVC: `func showCities()`

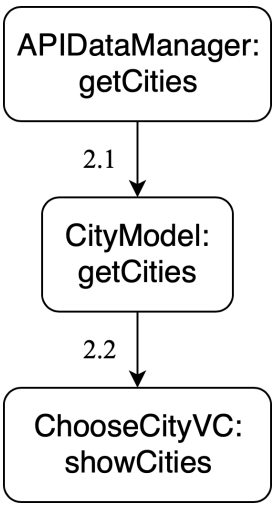

Рисунок 3 – Работа с городами

Стратегия конфигурационного тестирования

Приложение будет протестировано на устройствах, имеющих различную версию операционной системы и разрешение экрана.

Стратегия аттестационного тестирования

В ходе аттестационного тестирования будет протестирована работоспособность приложения и его возможность осуществлять заявленный функционал. Аттестационные тесты покрывают ранее перечисленные функциональные требования. Аттестационное тестирование происходит ручным способом.

Описание тестов

Корректность возвращаемых данных

Под корректной работой сервера подразумевается его доступность, а также отсутствие ошибок в логике работы сервера. Кроме того, сервер должен возвращать валидный JSON (см. ниже), коды ответа сервера должны быть в диапазоне 200-399.

Пример валидного JSON-файла со списком городов:

```
{
    "cities": [
        {
            "id": "1",
            "name": "Petrozavodsk",
            "country_name": "Russia",
            "latitude": 61.7782,
            "longitude": 34.364
        },
        {
            "id": "2",
            "name": "London",
            "country_name": "United Kingdom",
            "latitude": 51.50853,
            "longitude": -0.076132
        }
    ]
}
```

Валидность JSON-файла можно проверить при помощи сервиса-валидатора (например, jsonlint.com).

Описание блочных тестов

Тест 1.

Модуль: APIDataManager.

Метод: `func initBeaconsAndGeofencesAndImageTargets(completionHandler: @escaping ErrorType? -> ())`

Тип: Негативный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Нет подключения к сети Интернет.

Ожидаемый результат: Метод возвращает ошибку networkError.

Тест 2.

Модуль: APIDataManager.

Метод: `func initBeaconsAndGeofencesAndImageTargets(completionHandler: @escaping ErrorType? -> ())`

Тип: Негативный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Есть подключение к сети Интернет, сервер возвращает ошибку с кодом ответа в диапазоне от 400 до 599.

Ожидаемый результат: Метод возвращает ошибку serverError.

Тест 3.

Модуль: APIDataManager.

Метод: `func initBeaconsAndGeofencesAndImageTargets(completionHandler: @escaping ErrorType? -> ())`

Тип: Негативный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно, возвращает несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку modelError.

Тест 4.

Модуль: APIDataManager.

Метод: `func initBeaconsAndGeofencesAndImageTargets(completionHandler: @escaping ErrorType? -> ())`

Тип: Позитивный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод не возвращает ошибку.

Тест 5.

Модуль: APIDataManager.

Метод: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Нет подключения к сети Интернет.

Ожидаемый результат: Метод возвращает ошибку networkError.

Тест 6.

Модуль: APIDataManager.

Метод: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Есть подключение к сети Интернет, сервер возвращает ошибку с кодом ответа в диапазоне от 400 до 599.

Ожидаемый результат: Метод возвращает ошибку serverError.

Тест 7.

Модуль: APIDataManager.

Метод: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка

сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно, возвращает несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку modelError.

Тест 8.

Модуль: APIDataManager.

Метод: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

Тип: Позитивный.

Описание: Метод делает GET-запрос на сервер. В случае успешного выполнения запроса метод не возвращает ошибку (в замыкании ErrorType? передаётся равным nil). В случае неудачного выполнения запроса возвращается одна из следующих ошибок: ошибка подключения к сети Интернет (networkError), ошибка сервера (serverError), ошибка разбора JSON-файла, полученного с сервера (modelError).

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод не возвращает ошибку.

Тест 9.

Модуль: CityModel.

Метод: `func getCities(from data: NSDictionary?) -> [CityModel]?`

Тип: Позитивный.

Описание: Метод получает список городов в формате JSON и преобразует его в CityModel. В случае успешного разбора JSON-файла возвращает массив [CityModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив [CityModel] с количеством элементов, равным количеству элементов в JSON-файле.

Тест 10.

Модуль: CityModel.

Метод: `func getCities(from data: NSDictionary?) -> [CityModel]?`

Тип: Негативный.

Описание: Метод получает список городов в формате JSON и преобразует его в CityModel. В случае успешного разбора JSON-файла возвращает массив [CityModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Тест 11.

Модуль: BeaconModel.

Метод: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`

Тип: Позитивный.

Описание: Метод получает список beacons в формате JSON и преобразует его в BeaconModel. В случае успешного разбора JSON-файла возвращает массив [BeaconModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив [BeaconModel] с количеством элементов, равным количеству элементов в JSON-файле.

Тест 12.

Модуль: BeaconModel.

Метод: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`

Тип: Негативный.

Описание: Метод получает список beacons в формате JSON и преобразует его в BeaconModel. В случае успешного разбора JSON-файла возвращает массив [BeaconModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Тест 13.

Модуль: GeofenceModel.

Метод: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`

Тип: Позитивный.

Описание: Метод получает список geofences в формате JSON и преобразует его в GeofenceModel. В случае успешного разбора JSON-файла возвращает массив [GeofenceModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив [GeofenceModel] с количеством элементов, равным количеству элементов в JSON-файле.

Тест 14.

Модуль: GeofenceModel.

Метод: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`

Тип: Негативный.

Описание: Метод получает список geofences в формате JSON и преобразует его в GeofenceModel. В случае успешного разбора JSON-файла возвращает массив [GeofenceModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Тест 15.

Модуль: ImageTargetModel.

Метод: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`

Тип: Позитивный.

Описание: Метод получает список ImageTargets в формате JSON и преобразует его в ImageTargetModel. В случае успешного разбора JSON-файла возвращает массив [ImageTargetModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив [ImageTargetModel] с количеством элементов, равным количеству элементов в JSON-файле.

Тест 16.

Модуль: ImageTargetModel.

Метод: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`

Тип: Негативный.

Описание: Метод получает список ImageTargets в формате JSON и преобразует его в ImageTargetModel. В случае успешного разбора JSON-файла возвращает массив [ImageTargetModel]. В случае неудачного разбора JSON-файла возвращает nil.

Исходные данные: Несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Описание интеграционных тестов

Тест 17.

Методы:

1. APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping ErrorType? -> ())`
2. BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`

Тип: Общий.

Этап интеграции: 1.1.

Описание:

- Получение с сервера списка всех доступных geofences, beacons и Image-Targets.
- Разбор массива beacons из JSON-файла в модель.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [BeaconModel].

Тест 18.

Методы:

1. APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping ErrorType? -> ())`
2. BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`
3. GeofenceModel: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`

Тип: Общий.

Этап интеграции: 1.2.

Описание:

- Получение с сервера списка всех доступных geofences, beacons и Image-Targets.
- Разбор массива beacons из JSON-файла в модель.
- Разбор массива geofences из JSON-файла в модель.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [BeaconModel].
- В APIDataManager вернулся массив [GeofenceModel].

Тест 19.

Методы:

1. APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping ErrorType? -> ())`
2. BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`
3. GeofenceModel: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`
4. ImageTargetModel: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`

Тип: Общий.

Этап интеграции: 1.3.

Описание:

- Получение с сервера списка всех доступных geofences, beacons и ImageTargets.
- Разбор массива beacons из JSON-файла в модель.
- Разбор массива geofences из JSON-файла в модель.
- Разбор массива imageTargets из JSON-файла в модель.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [BeaconModel].

- В APIDataManager вернулся массив [GeofenceModel].
- В APIDataManager вернулся массив [ImageTargetModel].

Тест 20.

Методы:

1. APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping ErrorType? -> ())`
2. BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`
3. GeofenceModel: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`
4. ImageTargetModel: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`
5. CardsVC: `func getGeofencesBeaconsAndImageTargets()`

Тип: Общий.

Этап интеграции: 1.4.

Описание:

- Получение с сервера списка всех доступных geofences, beacons и ImageTargets.
- Разбор массива beacons из JSON-файла в модель.
- Разбор массива geofences из JSON-файла в модель.
- Разбор массива imageTargets из JSON-файла в модель.
- Получение списка geofences, beacons и ImageTargets, а также сценариев к ним и обновление данных в базе данных.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.

- В APIDataManager вернулся массив [BeaconModel].
- В APIDataManager вернулся массив [GeofenceModel].
- В APIDataManager вернулся массив [ImageTargetModel].
- Geofences, beacons и ImageTargets, а также сценарии к ним добавились в базу данных.

Тест 21.

Методы:

1. APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping ErrorType? -> ())`
2. BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`
3. GeofenceModel: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`
4. ImageTargetModel: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`
5. CardsVC: `func getGeofencesBeaconsAndImageTargets()`
6. CardsVC: `func showCards()`

Тип: Общий.

Этап интеграции: 1.5.

Описание:

- Получение с сервера списка всех доступных geofences, beacons и ImageTargets.
- Разбор массива beacons из JSON-файла в модель.
- Разбор массива geofences из JSON-файла в модель.
- Разбор массива imageTargets из JSON-файла в модель.
- Получение списка geofences, beacons и ImageTargets, а также сценариев к ним и обновление данных в базе данных.
- Отображение списка карточек с полученными geofences, beacons и ImageTargets на экране.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [BeaconModel].
- В APIDataManager вернулся массив [GeofenceModel].
- В APIDataManager вернулся массив [ImageTargetModel].
- Geofences, beacons и ImageTargets, а также сценарии к ним добавились в базу данных.
- Список карточек с полученными geofence, beacons и ImageTargets отображается на экране.

Тест 22.

Методы:

1. APIDataManager: `func initBeaconsAndGeofencesAndImageTargets (completionHandler: @escaping ErrorType? -> ())`
2. BeaconModel: `func getBeacons(from data: NSDictionary?) -> [BeaconModel]?`
3. GeofenceModel: `func getGeofences(from data: NSDictionary?) -> [GeofenceModel]?`
4. ImageTargetModel: `func getImageTargets(from data: NSDictionary?) -> [ImageTargetModel]?`
5. CardsVC: `func getGeofencesBeaconsAndImageTargets()`
6. CardsVC: `func showCards()`
7. GoogleMapsVC: `func setUpMarkers()`

Тип: Общий.

Этап интеграции: 1.6.

Описание:

- Получение с сервера списка всех доступных geofences, beacons и Image-Targets.
- Разбор массива beacons из JSON-файла в модель.
- Разбор массива geofences из JSON-файла в модель.
- Разбор массива imageTargets из JSON-файла в модель.
- Получение списка geofences, beacons и ImageTargets, а также сценариев к ним и обновление данных в базе данных.
- Отображение списка карточек с полученными geofences, beacons и Image-Targets на экране.
- Создание на карте маркеров для всех доступных geofences, beacons и ImageTargets.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [BeaconModel].
- В APIDataManager вернулся массив [GeofenceModel].
- В APIDataManager вернулся массив [ImageTargetModel].
- Geofences, beacons и ImageTargets, а также сценарии к ним добавились в базу данных.
- Список карточек с полученными geofence, beacons и ImageTargets отображается на экране.
- На карте установлены маркеры для всех доступных geofences, beacons и ImageTargets.

Тест 23.

Методы:

1. APIDataManager: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

2. CityModel: `func getCities(from data: NSDictionary?) -> [CityModel]?`

Тип: Общий.

Этап интеграции: 2.1.

Описание:

- Получение с сервера списка городов.
- Разбор массива городов из JSON-файла в модель.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [CityModel].

Тест 24.

Методы:

1. APIDataManager: `func getCities(completionHandler: @escaping ([CityModel]?, ErrorType?) -> ())`

2. CityModel: `func getCities(from data: NSDictionary?) -> [CityModel]?`

3. ChooseCityVC: `func showCities()`

Тип: Общий.

Этап интеграции: 2.2.

Описание:

- Получение с сервера списка городов.
- Разбор массива городов из JSON-файла в модель.
- Отображение списка городов на экране.

Исходные данные:

- Есть подключение к сети Интернет.
- Сервер работает корректно, возвращает валидный JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат:

- С сервера получен JSON-файл.
- В APIDataManager вернулся массив [CityModel].
- Список городов отображается на экране.

Описание аттестационных тестов

Тест 25.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка отображения списка карточек, полученных из базы данных, для выбранного города.

Алгоритм: Запустить приложение. Выбрать город.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Автоматический переход на главный экран приложения. На экране отображается список ранее полученных карточек.

Тест 26.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка удаления карточки из списка по свайпу влево.

Алгоритм: Запустить приложение. Выбрать город. Сделать свайп слева-направо по одной из карточек на главном экране.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Карточка удаляется (пропадает) из списка.

Тест 27.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка удаления карточки из базы данных по свайпу влево.

Алгоритм: Запустить приложение. Выбрать город. Сделать свайп слева-направо по одной из карточек на главном экране. Выгрузить приложение из памяти. Запустить приложение. Выбрать город.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Карточка удаляется (пропадает) из списка.

Тест 28.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка осуществления сценария для данной карточки по нажатию на карточку.

Алгоритм: Запустить приложение. Выбрать город. Получить карточку (например, отсканировать ImageTarget или другим способом) с вложенным сценарием «Открытие ссылки в браузере». Нажать на карточку.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Открытие ссылки в браузере.

Тест 29.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка осуществления сценария для данной карточки по нажатию на карточку.

Алгоритм: Запустить приложение. Выбрать город. Получить карточку (например, отсканировать ImageTarget или другим способом) с вложенным сценарием «Открытие ссылки в WebView». Нажать на карточку.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Открытие ссылки в WebView.

Тест 30.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка осуществления сценария для данной карточки по нажатию на карточку.

Алгоритм: Запустить приложение. Выбрать город. Получить карточку (например, отсканировать ImageTarget или другим способом) с вложенным сценарием «Воспроизведение видео». Нажать на карточку.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Открытие медиа-плеера и воспроизведение видео.

Тест 31.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка добавления новых карточек в начало списка в зависимости от получаемых сообщений от других модулей.

Алгоритм: Запустить приложение. Выбрать город. Нажать на кнопку с изображением ImageTarget. Отсканировать ImageTarget.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»), пользователь разрешил доступ к камере, карточки для ImageTarget нет в списке карточек.

Ожидаемый результат: Приходит уведомление с названием ImageTarget. На главном экране приложения в начале списка появляется соответствующая карточка.

Тест 32.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка обновления списка карточек в зависимости от получаемых сообщений от других модулей.

Алгоритм: Запустить приложение. Выбрать город. Нажать на кнопку с изображением ImageTarget. Отсканировать ImageTarget.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»), пользователь разрешил доступ к камере, карточка для ImageTarget уже есть в списке карточек.

Ожидаемый результат: Приходит уведомление с названием ImageTarget. На главном экране приложения соответствующая карточка поднимается в начало списка.

Тест 33.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка установления мониторинга beacons, а также добавления новых карточек в начало списка в зависимости от получаемых сообщений от других модулей.

Алгоритм: Запустить приложение. Выбрать город. Войти в зону действия beacon.

Исходные данные: Bluetooth включен, есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»), карточки для beacon нет в списке карточек.

Ожидаемый результат: Приходит уведомление с названием beacon. На главном экране приложения в начале списка появляется соответствующая карточка.

Тест 34.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка установления мониторинга beacons, а также обновления списка карточек в зависимости от получаемых сообщений от других модулей.

Алгоритм: Запустить приложение. Выбрать город. Войти в зону действия beacon.

Исходные данные: Bluetooth включен, есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»), карточка для beacon уже есть в списке карточек.

Ожидаемый результат: Приходит уведомление с названием beacon. На главном экране приложения в начале списка появляется соответствующая карточка.

Тест 35.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка установления мониторинга geofences, а также добавления новых карточек в начало списка в зависимости от получаемых сообщений от других модулей.

Алгоритм: Запустить приложение. Выбрать город. Войти в зону действия geofence.

Исходные данные: Геолокация включена, есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»), карточки для geofence нет в списке карточек.

Ожидаемый результат: Приходит уведомление с названием geofence. На главном экране приложения в начале списка появляется соответствующая карточка.

Тест 36.

Экран: Главный экран приложения.

Тип: Позитивный.

Описание: Проверка установления мониторинга geofences, а также обновления списка карточек в зависимости от получаемых сообщений от других модулей.

Алгоритм: Запустить приложение. Выбрать город. Войти в зону действия geofence.

Исходные данные: Геолокация включена, есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»), карточка для geofence уже есть в списке карточек.

Ожидаемый результат: Приходит уведомление с названием geofence. На главном экране приложения в начале списка появляется соответствующая карточка.

Тест 37.

Экран: Экран выбора города.

Тип: Позитивный.

Описание: Проверка получения с сервера списка городов и отображение этого списка на экране.

Алгоритм: Запустить приложение.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Открытие экрана выбора города. На экране отображается список городов.

Тест 38.

Экран: Экран выбора города.

Тип: Позитивный.

Описание: Проверка автоматического перехода на главный экран при повторном входе в приложение.

Алгоритм: Запустить приложение. Выбрать город. Выгрузить приложение из памяти. Запустить приложение.

Исходные данные: Есть подключение к сети Интернет, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Открытие главного экрана.

Тест 39.

Экран: Экран выбора города.

Тип: Негативный.

Описание: Проверка получения ошибки об отсутствии подключения к сети Интернет.

Алгоритм: Выключить Wi-Fi и мобильные данные. Запустить приложение.

Исходные данные: Нет подключения к сети Интернет.

Ожидаемый результат: Открытие экрана выбора города. На экране отображается сообщение об отсутствии подключения к сети Интернет.

Тест 40.

Экран: Экран карты.

Тип: Позитивный.

Описание: Проверка получения из базы данных всех geofence, beacon и ImageTarget и отображения их на карте.

Алгоритм: Запустить приложение. Выбрать город. Нажать на кнопку с изображением карты.

Исходные данные: Нет.

Ожидаемый результат: Открытие карты. На карте расставлены маркеры для всех geofence, beacon и ImageTarget, доступных в выбранном городе.

Тест 41.

Экран: Экран карты.

Тип: Позитивный.

Описание: Проверка отображения текущего местоположения пользователя.

Алгоритм: Запустить приложение. Выбрать город. Нажать на кнопку с изображением карты.

Исходные данные: Геолокация включена.

Ожидаемый результат: На карте отображается текущее местоположение пользователя.

Тест 42.

Экран: Экран карты.

Тип: Позитивный.

Описание: Проверка построения маршрута от текущего местоположения пользователя до выбранного объекта на карте в приложении (при наличии) или на сайте Google Maps.

Алгоритм: Запустить приложение. Выбрать город. Перейти на экран карты. Нажать на одну из меток на карте. Нажать на кнопку построения маршрута.

Исходные данные: Геолокация включена, есть подключение к сети Интернет, на устройстве установлено приложение Google Maps.

Ожидаемый результат: Открытие приложения Google Maps с построенным маршрутом от текущего местоположения пользователя до выбранного объекта на карте.

Тест 43.

Экран: Экран карты.

Тип: Позитивный.

Описание: Проверка построения маршрута от текущего местоположения пользователя до выбранного объекта на карте в приложении (при наличии) или на сайте Google Maps.

Алгоритм: Запустить приложение. Выбрать город. Перейти на экран карты. Нажать на одну из меток на карте.

Исходные данные: Геолокация включена, есть подключение к сети Интернет, на устройстве не установлено приложение Google Maps.

Ожидаемый результат: Открытие браузера на странице Google Maps с построенным маршрутом от текущего местоположения пользователя до выбранного объекта на карте.

Тест 44.

Модуль: LocationManager.

Тип: Позитивный.

Описание: Проверка срабатывания geofence при открытом приложении.

Алгоритм: Запустить приложение. Выбрать город. Войти в зону действия geofence.

Исходные данные: Геолокация включена.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании geofence.

Тест 45.

Модуль: LocationManager.

Тип: Позитивный.

Описание: Проверка срабатывания geofence в фоновом режиме.

Алгоритм: Запустить приложение. Выбрать город. Свернуть приложение. Войти в зону действия geofence.

Исходные данные: Геолокация включена.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании geofence.

Тест 46.

Модуль: LocationManager.

Тип: Позитивный.

Описание: Проверка срабатывания geofence, когда приложение выгружено из памяти.

Алгоритм: Запустить приложение. Выбрать город. Выгрузить приложение из памяти. Войти в зону действия geofence.

Исходные данные: Геолокация включена.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании geofence.

Тест 47.

Модуль: LocationManager.

Тип: Позитивный.

Описание: Проверка срабатывания beacon при открытом приложении.

Алгоритм: Запустить приложение. Выбрать город.

Исходные данные: Bluetooth включен.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании beacon.

Тест 48.

Модуль: LocationManager.

Тип: Позитивный.

Описание: Проверка срабатывания beacon в фоновом режиме.

Алгоритм: Запустить приложение. Выбрать город. Свернуть приложение. Войти в зону действия beacon.

Исходные данные: Bluetooth включен.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании beacon.

Тест 49.

Модуль: LocationManager.

Тип: Позитивный.

Описание: Проверка срабатывания beacon, когда приложение выгружено из памяти.

Алгоритм: Запустить приложение. Выбрать город. Выгрузить приложение из памяти. Войти в зону действия beacon.

Исходные данные: Bluetooth включен.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании beacon.

Описание конфигурационного тестирования

Все тесты, описанные выше, будут тестироваться на следующих устройствах:

1. Устройство: Apple iPhone

Модель: 5S

Версия ОС: iOS 11.4

Разрешение экрана: 1136 x 640 (326 ppi)

2. Устройство: Apple iPhone

Модель: 7

Версия ОС: iOS 12.1

Разрешение экрана: 1334 x 750 (326 ppi)

3. Устройство: Apple iPhone

Модель: X

Версия ОС: iOS 12.0

Разрешение экрана: 2436 x 1125 (458 ppi)

Целью конфигурационного тестирования является проверка работы заявленного выше функционала на различных устройствах, имеющих разные версии операционной системы и разные разрешения экранов. Также необходимо убедиться в том, что вёрстка приложения соответствует дизайну.

Примеры реализации тестов

Пример реализации теста получения списка городов с сервера:

```
func testGetCities() {
    //given
    var responseError: ErrorType?
    //when
        dataManager?.getCities(completionHandler: { responseData,
error in
        if error != nil {
            responseError = error
        }
    })
    //then
         XCTAssertNil(responseError, "error: failed when getting
cities»)
}
```

Пример реализации теста разбора JSON-файла со списком городов в модель:

```
func testCitiesJsonMappingCorrect() {
    //given
    let countOfItems = 10
    var itemCount: Int? = 0
    //when
      if let path = Bundle(for: type(of: self)).path(forResource:
"Cities", ofType: "json"),
        let data = NSData(contentsOfFile: path) {
        do {
                if let responseData = try? JSONSerialization.json-
Object(with: data as Data, options: []) as? NSDictionary {
                 let itemsArray = CityModel(from: responseData)
                 itemCount = itemsArray?.count
            }
        }
    }
```

```
        //then
         XCTAssertEqual(countOfItems, itemCount, "error: failed parsed
list of cities")
    }
```

Отчёт о проведении первоначального тестирования

№ Теста	Дата	Результат (iPhone 5S)	Результат (iPhone 7)	Результат (iPhone X)	Отчёт об ошибке
Блочное тестирование					
1	01.12.2018	Пройден	Пройден	Пройден	-
2	01.12.2018	Пройден	Пройден	Пройден	-
3	01.12.2018	Пройден	Пройден	Пройден	-
4	01.12.2018	Пройден	Пройден	Пройден	-
5	01.12.2018	Пройден	Пройден	Пройден	-
6	01.12.2018	Пройден	Пройден	Пройден	-
7	01.12.2018	Пройден	Пройден	Пройден	-
8	01.12.2018	Ошибка	Ошибка	Ошибка	Отчёт №1
9	01.12.2018	Пройден	Пройден	Пройден	-
10	01.12.2018	Пройден	Пройден	Пройден	-
11	01.12.2018	Пройден	Пройден	Пройден	-
12	01.12.2018	Пройден	Пройден	Пройден	-
13	01.12.2018	Пройден	Пройден	Пройден	-
14	01.12.2018	Пройден	Пройден	Пройден	-
15	01.12.2018	Пройден	Пройден	Пройден	-
16	01.12.2018	Пройден	Пройден	Пройден	-
Интеграционное тестирование					
17	01.12.2018	Пройден	Пройден	Пройден	-
18	01.12.2018	Пройден	Пройден	Пройден	-
19	01.12.2018	Пройден	Пройден	Пройден	-
20	01.12.2018	Пройден	Пройден	Пройден	-
21	01.12.2018	Пройден	Пройден	Пройден	-
22	01.12.2018	Пройден	Пройден	Пройден	-
23	01.12.2018	Пройден	Пройден	Пройден	-

24	01.12.2018	Пройден	Пройден	Пройден	-
Аттестационное тестирование					
25	01.12.2018	Пройден	Пройден	Пройден	-
26	01.12.2018	Пройден	Пройден	Пройден	-
27	01.12.2018	Пройден	Пройден	Пройден	-
28	01.12.2018	Пройден	Пройден	Пройден	-
29	01.12.2018	Пройден	Пройден	Пройден	-
30	01.12.2018	Пройден	Пройден	Пройден	-
31	01.12.2018	Пройден	Пройден	Пройден	-
32	01.12.2018	Ошибка	Ошибка	Ошибка	Отчёт №2
33	01.12.2018	Пройден	Пройден	Пройден	-
34	01.12.2018	Пройден	Пройден	Пройден	-
35	01.12.2018	Пройден	Пройден	Пройден	-
36	01.12.2018	Пройден	Пройден	Пройден	-
37	01.12.2018	Пройден	Пройден	Пройден	-
38	01.12.2018	Пройден	Пройден	Пройден	-
39	01.12.2018	Пройден	Пройден	Пройден	-
40	01.12.2018	Пройден	Пройден	Пройден	-
41	01.12.2018	Пройден	Пройден	Пройден	-
42	01.12.2018	Пройден	Пройден	Пройден	-
43	01.12.2018	Пройден	Пройден	Пройден	-
44	01.12.2018	Пройден	Пройден	Пройден	-
45	01.12.2018	Ошибка	Ошибка	Ошибка	Отчёт №3
46	01.12.2018	Пройден	Пройден	Пройден	-
47	01.12.2018	Пройден	Пройден	Пройден	-
48	01.12.2018	Пройден	Пройден	Пройден	-
49	01.12.2018	Пройден	Пройден	Пройден	-

Отчеты об ошибках

Отчёт №1.

Краткое описание: Ошибка при выполнении теста 8. Проверка получения с сервера валидного JSON-файла со списком городов.

Ожидаемый результат: Метод не возвращает ошибку.

Фактический результат: Метод возвращает ошибку modelError.

Отчёт №2.

Краткое описание: Ошибка при выполнении теста 32. Проверка добавления перемещения уже имеющейся карточки в начало списка при повторном сканировании ImageTarget.

Ожидаемый результат: Карточка перемещается начало списка.

Фактический результат: Карточка добавляется повторно (дублируется).

Отчёт №3.

Краткое описание: Ошибка при выполнении теста 45. Проверка срабатывания geofence в фоновом режиме.

Ожидаемый результат: Пользователь получает уведомление об успешном срабатывании geofence.

Фактический результат: Пользователь не получает уведомление об успешном срабатывании geofence.

Отчёт о проведении повторного тестирования

№ Теста	Дата	Результат (iPhone 5S)	Результат (iPhone 7)	Результат (iPhone X)	Отчёт об ошибке
Блочное тестирование					
1	01.12.2018	Пройден	Пройден	Пройден	-
2	01.12.2018	Пройден	Пройден	Пройден	-
3	01.12.2018	Пройден	Пройден	Пройден	-
4	01.12.2018	Пройден	Пройден	Пройден	-
5	01.12.2018	Пройден	Пройден	Пройден	-
6	01.12.2018	Пройден	Пройден	Пройден	-
7	01.12.2018	Пройден	Пройден	Пройден	-
8	01.12.2018	Пройден	Пройден	Пройден	-
9	01.12.2018	Пройден	Пройден	Пройден	-
10	01.12.2018	Пройден	Пройден	Пройден	-
11	01.12.2018	Пройден	Пройден	Пройден	-
12	01.12.2018	Пройден	Пройден	Пройден	-
13	01.12.2018	Пройден	Пройден	Пройден	-
14	01.12.2018	Пройден	Пройден	Пройден	-
15	01.12.2018	Пройден	Пройден	Пройден	-
16	01.12.2018	Пройден	Пройден	Пройден	-
Интеграционное тестирование					
17	01.12.2018	Пройден	Пройден	Пройден	-
18	01.12.2018	Пройден	Пройден	Пройден	-
19	01.12.2018	Пройден	Пройден	Пройден	-
20	01.12.2018	Пройден	Пройден	Пройден	-
21	01.12.2018	Пройден	Пройден	Пройден	-
22	01.12.2018	Пройден	Пройден	Пройден	-
23	01.12.2018	Пройден	Пройден	Пройден	-
24	01.12.2018	Пройден	Пройден	Пройден	-
Аттестационное тестирование					

25	01.12.2018	Пройден	Пройден	Пройден	-
26	01.12.2018	Пройден	Пройден	Пройден	-
27	01.12.2018	Пройден	Пройден	Пройден	-
28	01.12.2018	Пройден	Пройден	Пройден	-
29	01.12.2018	Пройден	Пройден	Пройден	-
30	01.12.2018	Пройден	Пройден	Пройден	-
31	01.12.2018	Пройден	Пройден	Пройден	-
32	01.12.2018	Пройден	Пройден	Пройден	-
33	01.12.2018	Пройден	Пройден	Пройден	-
34	01.12.2018	Пройден	Пройден	Пройден	-
35	01.12.2018	Пройден	Пройден	Пройден	-
36	01.12.2018	Пройден	Пройден	Пройден	-
37	01.12.2018	Пройден	Пройден	Пройден	-
38	01.12.2018	Пройден	Пройден	Пройден	-
39	01.12.2018	Пройден	Пройден	Пройден	-
40	01.12.2018	Пройден	Пройден	Пройден	-
41	01.12.2018	Пройден	Пройден	Пройден	-
42	01.12.2018	Пройден	Пройден	Пройден	-
43	01.12.2018	Пройден	Пройден	Пройден	-
44	01.12.2018	Пройден	Пройден	Пройден	-
45	01.12.2018	Пройден	Пройден	Пройден	-
46	01.12.2018	Пройден	Пройден	Пройден	-
47	01.12.2018	Пройден	Пройден	Пройден	-
48	01.12.2018	Пройден	Пройден	Пройден	-
49	01.12.2018	Пройден	Пройден	Пройден	-

Заключение

Разработанное приложение удовлетворяет всем функциональным требованиям. Для тестирования приложения составлено 49 различных тестов. При тестировании приложения было найдено три ошибки: одна при блочном тестировании и две при аттестационном. Все найденные ошибки были исправлены после окончания тестирования. При повторном тестировании ошибок найдено не было.

Расчет тестового покрытия относительно исполняемого кода приложения проводится по формуле:

Covering = tested_length / code_length * 100%, где tested_length – это количество строк кода, покрытых тестами, а code_length – это общее количество строк кода в приложении.

Covering = 1343 / 1438 * 100% = 93%