МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра вычислительных технологий

Допустить к защите
Заведующий кафедрой
д-р физ.-мат. наук, профессор
_____ А.И. Миков
14 июня 2018 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ ВНУТРЕННЕГО ПОЗИЦИОНИРОВАНИЯ

Работу выполнил _____ В.К. Михолапов

Направление подготовки 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль) Вычислительные технологии

Научный руководитель
канд. физ.-мат. наук, доцент _____ Е.В. Кособуцкая

Нормоконтролёр
канд. физ.-мат. наук, доцент _____ Е.В. Кособуцкая

Краснодар
2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день навигация — актуальная тема. Наш мир меняется очень быстро: быстро возводятся новые постройки, быстро меняют адреса организаций и быстро появляются новые улицы, люди всё чаще и чаще меняют своё место жительства, путешествуют. В связи с указанными выше причинами увеличилась потребность в современных, мобильных и удобных методах навигации.

Навигацию можно условно разделить на два типа: глобальная и локальная. Глобальная навигация не привязана к какому-либо участку местности и позволяет ориентироваться в любой точке мира, где есть доступ к спутникам, обеспечивающим этот тип навигации. Примерами систем глобальной навигации могут выступать GPS (Global Positioning System) и ГЛОНАСС (ГЛОбальная Навигационная Спутниковая Система).

Локальная навигация ограничена каким-то определённым участком местности. Она может осуществляться на основе взаимодействия устройств, с помощью которого осуществляется навигация, с другими устройствами, например, Bluetooth-маячками или Wi-Fi модулями.

В то время, как средства глобальной навигации позволяют людям ориентироваться, например, при поездке из одного региона в другой или при поиске определённой организации в конкретном городе, средства локальной навигации помогают ориентироваться в пределах одного большого помещения или на территории парка.

Средств, для использования систем глобальной навигации на рынке очень много. Есть множество приложений, предоставляющих карты, с помощью которых можно ориентироваться в городе или за городом. В то же время решения для внутреннего позиционирования не так уж и много, и они используются преимущественно крупными технологическими компаниями.

Например, в компании Amazon работа склада полностью автоматизирована и роботы-грузчики самостоятельно могут ориентироваться

по складским помещениям с помощью системы дорожной разметки, которая отмечает маршруты для роботов.

Ещё одним примером может послужить мероприятие GeekPicnic, прошедшее в Москве и Санкт-Петербурге в 2015 году. На нём активно использовалась сеть из Bluetooth-маячков и приложение, которое могло с ними взаимодействовать, для более удобной навигации посетителей между разными зонами мероприятия [1].

Целью данной квалификационной работы является построение системы внутреннего позиционирования основанной на технологии iBeacon и двух алгоритмах, а также сравнение этих алгоритмов. Указанная система состоит из веб-сервера, сети маячков и мобильного приложения, которое будет позволять с помощью разных алгоритмов определять местоположение устройства, на котором оно запущено.

Задачи:

– Построение системы внутреннего позиционирования, основанной на технологии iBeacon и трилатерации.

– Построение системы внутреннего позиционирования, основанной на технологии iBeacon и методе снятия «отпечатков».

– Сравнение эффективности полученных систем по таким параметрам, как сложность алгоритма, точность, задержки определения местоположения, возможность масштабирования и изменения топологии, побочные расходы, связанные с разработкой, установкой и эксплуатацией системы (включая денежную стоимость и человеческие ресурсы).

1 Общие сведения о методах внутреннего позиционирования

1.1 Описание и назначение

Система внутреннего позиционирования (indoor positioning system или IPS, СВП) – это система, позволяющая находить объекты или людей внутри строения, используя при этом радиоволны, магнитные поля, акустические сигналы или иные сигналы, доступные мобильному устройству.

Системы внутреннего позиционирования используют технологии измерения расстояния с помощью опорных точек (они имеют фиксированную известную позицию, например, модули Wi-Fi, Bluetooth-маячки). Устройства, играющие роль опорных точек, либо активно находят мобильные устройства или метки, либо предоставляют данные об их окружении или себе.

Дизайн таких систем должен учитывать тот факт, что необходимо как минимум три опорных точки, чтобы однозначно определить позицию объекта (трилатерация). Так как все физические методы получения расстояния до объекта подвержены стохастическим помехам, то СВП так же должны включать в себя надёжные методы их сглаживания. Для этого можно использовать математический аппарат, либо использовать данные от нескольких систем, либо расширять систему горизонтально, то есть добавляя к ней ещё больше опорных точек.

Обнаружение ориентации устройства может быть достигнуто, например, с помощью обнаружения ориентиров, анализа изображения, поступающего с устройства в реальном времени или с помощью использования трилатерации.

1.2 Технологии, используемые для внутреннего позиционирования

1.2.1 Использование Wi-Fi

Внутри помещений Wi-Fi является хорошей альтернативой GPS, сигналы которого недоступны. В большинстве случаев Wi-Fi система позиционирования является легкой в установке, так как Wi-Fi точки доступа уже доступны во многих помещениях. Преимущество заключается в том, что могут использоваться существующие системы кассовых аппаратов, общественные и точки доступа магазинов. Пользователю необязательно подключаться к точкам доступа, достаточно включить Wi-Fi. Для позиционирования используется так называемый метод «снятия отпечатков». Сила сигналов Wi-Fi (индикация уровня принятого сигнала, RSSI) и MAC-адрес (управление доступом к среде) являются важными. На смартфоне должно быть установлено соответствующее приложение, которое рассчитывает текущую позицию на основе этих данных.

Точность зависит от множества факторов, например, от количества доступных сетей, отражений в коридорах, и не в последнюю очередь от экранирования стен, потолков и собственного тела. Точность Wi-Fi, используемого для внутреннего позиционирования, варьируется от 5 до 15 метров – в зависимости от предварительных условий. Большим преимуществом по сравнению с GPS является то, что можно определить текущий уровень пола.

1.2.2 Геомагнитное позиционирование

Основано на ориентировании по магнитному полю Земли и базируется на геомагнитных аномалиях как критериях для геомагнитного позиционирования (аномалии возникают вследствие неоднородности геомагнитного поля). Заключается в фиксации геомагнитных аномалий и нанесении их на карту территории, на которой предполагается

ориентироваться. В дальнейшем навигация производится по составленной карте устройством, в которое встроен магнитометр. Практический пример реализации – система IndoorAtlas команды учёных из финского университета Оулу. Недостаток – высокая сложность реализации, невысокая точность. В помещениях очень много динамически меняющихся магнитных аномалий (проводка, поле в которой меняется в зависимости от подключённой нагрузки и сильно меняет конфигурацию магнитного поля вокруг себя, посетители со своими радиоэлектронными устройствами, стеллажи, тележки), сильно усложняющих навигацию, основанную на указанном способе ориентировании в пространстве.

1.2.3 Системы спутниковой навигации и инерциальные навигационные системы (ИНС)

Подобные системы применяются, когда периодически появляется сигнал систем спутниковой навигации – например, проезд по тоннелю – когда въезжаем в тоннель, нам ещё доступны актуальные координаты и направление движения с GPS/ГЛОНАСС спутников, далее при въезде в тоннель мы теряем сигнал, и используем уже инерциальную навигационную систем (ИНС, на базе акселерометра, гироскопа, магнитометра), которая использует в качестве начальных условий последние актуальные данные с GPS/ГЛОНАСС до потери связи со спутником и поддерживает их актуальность на получаемых с датчиков данных о текущей скорости/ускорении/направлении движения, до возобновления связи со спутниками. Стоит принимать во внимание, что в ИНС ошибки постоянно накапливаются, и со временем данные, полученные с ИНС, становятся всё более и более отличными от действительности [4].

1.2.4 Ориентирование по базовым станциям операторов сотовой связи

В зоне видимости сотового телефона / GSM-модема постоянно находятся как минимум одна базовая станция GSM, а обычно – несколько. Координаты расположения этих базовых станций – известны (благодаря многочисленным навигационным сервисам, например, Яндекс.Навигатор – приложение получает информацию о видимых вашим телефоном базовых станций и текущем вашем положении по GSM/Глонасс, и отправляет эти сведения в Яндекс, где на основе этих данных строится база соответствий «Базовая станция координаты», к которой имеется свободный доступ через предоставляемое API). Отправляем в модем команду AT+CREG=2, в результате чего начинаем получать сообщения +CREG: с информацией о текущей подключенной базовой станции – LAC и CELLID (соответственно код зоны и идентификатор базовой станции). Отправив эти данные на один из специальных сервисов (предоставляемый Яндекс, Google и другими компаниями), получаем координаты этой базовой станции. Многие модемы позволяют получить список видимых базовых станций (БС) с указанием их LAC и CELLID – остаётся только через базы данных с координатами БС получить их координаты и методом триангуляции определить свое примерное местоположение. Минусы – невысокая точность (БС может быть удалена на расстоянии в 35 км от пользователя и некоторые БС являются мобильными и постоянно меняют свою дислокацию).

1.2.5 Использование Bluetooth-маячков

В настоящее время применяется в торговых центрах, выставочных залах, помещениях для проведения форумов, музеях и прочих помещениях, обладающих большой площадью или сложной структурой. Технология основывается на использовании Bluetooth-маячков, установленных по периметру помещения, базы данных о них и мобильном приложении

заведения. Подобная система позволяет добиться точности позиционирования в 3-5 метров. Стоимость маячков в среднем варьируется от 18 до 35 долларов за единицу, а их форм-фактор упрощает процесс монтажа в помещении (есть возможность распределить маячки по периметру используя простейший двусторонний скотч). Подобная система также пригодна для повторного использования. Достаточно демонтировать маячки, переконфигурировать их и монтировать на новой площади. Конечному пользователю для использования данной системы достаточно иметь под рукой телефон с установленным приложением и включенным адаптером Bluetooth. Таким образом можно говорить о невысокой стоимости конечной системы, простой установке, достаточной точности позиционирования и удобстве использования конечным пользователем.

1.2.6 Навигация, основанная на синергетическом эффекте

Данный подход решает задачу определения текущего местоположения, используя все (или большинство) из перечисленных выше способов. Эффективность достигается за счёт того, что мы используем сразу несколько векторов определения координат, что способствует компенсации ошибок и повышению точности определения координат [13].

1.3 Технология iBeacon

Маячки, работающие по технологии iBeacon, являются подмножеством спецификаций BLE-маячков. Отношения множеств маячков, Bluetooth-устройств, Bluetooth-маячков и маячков, работающих по технологии iBeacon, показаны на рисунке 1.

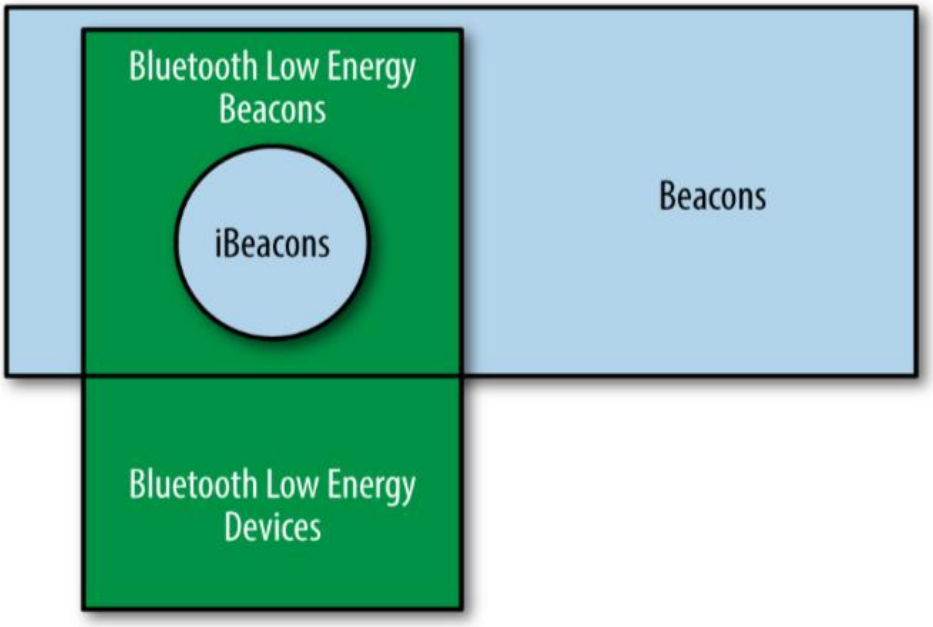

Рисунок 1 – Отношения между iBeacon, BLE-устройствами и маячками

Таким образом, все iBeacon-маячки являются обычными BLE-устройствами, которые широковещательно рассылают некоторую информацию.

Маячки, основанные на технологии iBeacon, могут только передавать данные. Они периодически широковещательно передают цифровые идентификаторы, которые устройства, способные и настроенные принимать эту передачу, интерпретируют и предпринимают некоторые действия.

Согласно Bluetooth-терминологии, маячки называют бродкастерами (broadcaster). Бродкастеры периодически передают специальные широковещательные пакеры (согласно спецификации Bluetooth, их полное название – nonconnectable undirected advertising packets), который содержит информацию, используемую получателем. При этом получатель не должен отвечать на них. По сути, маячок просто постоянно передаёт на широковещательном канале данные о себе и ту полезную нагрузку, которая была записана в него.

Каждый iBeacon-маячок идентифицируется по трём идентификаторам:

– Универсальный уникальный идентификатор (UUID).

128-битный идентификатор компании, которой принадлежит маячок. Он используется для идентификации группы маячков на самом высоком уровне иерархии и может быть использован для идентификации всех маячков, которые принадлежат какой-либо организации.

– Мажорное число (Major number).

Спецификации Bluetooth и iBeacon не накладывают никаких ограничений на использование минорного и мажорного числа, но тем не менее в их применении всегда присутствует иерархия. Мажорное число используется для идентификации крупной группы маячков, которыми владеет некоторая сущность. На примере сети магазинов, в каждом из отдельных магазинов будет одно мажорное число для группы маячков.

– Минорное число (Minor number).

Минорное число призвано идентифицировать группу маячков на самом низком уровне иерархии внутри множества маячков. В примере с сетью магазинов, минорное число будет опознавать отдельный маячок. [8]

Схема работы системы внутреннего позиционирования на основе технологии iBeacon проста:

У нас есть установленные по всему периметру Bluetooth-маячки, координаты расположения которых мы знаем. Эти маячки с заданной периодичностью производят широковещательную рассылку, содержащую идентифицирующую их информацию в виде специальный пакетов (advertising packets, AP). Пользовательское приложение циклично получает эти данные, по базе данных определяет координаты маячков, и на основе силы сигнала высчитывает свои координаты [2].

Оценка расстояния до маячка производится с помощью измерения силы сигнала (received signal strength indication или RSSI) [8].

AP всегда имеют одну и ту же длину и состоят из последовательности полей, фиксированного размера. Содержимое такого пакета показано на рисунке 2:

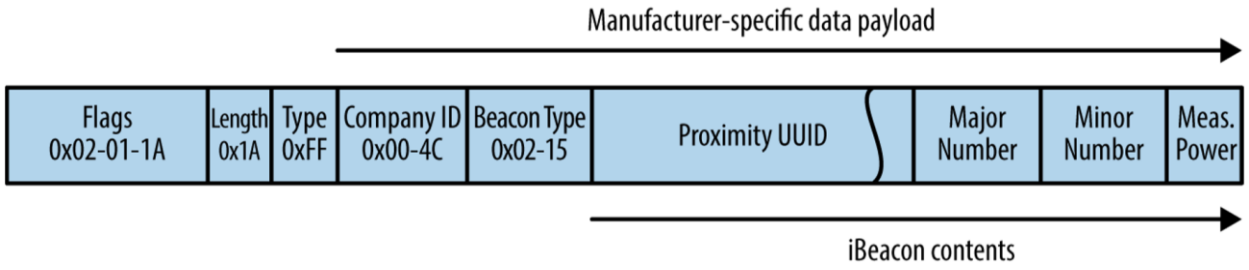

Рисунок 2 – Содержание широковещательного iBeacon-пакета

Последнее поле пакета содержит информацию, определяемую изготовителем. Оно представляет собой номинальную мощность сигнала.

Advertising packet содержит четыре поля, которые на практике имеют фиксированные значения для iBeacon и являются заголовками [8]:

1) Flags (3 байта, 0x02-01-1A).

Первый байт – это индикатор длины, который говорит о том, что данное поле имеет два дополнительных байта, помимо этого.

Второй байт описывает тип значения, содержащегося в следующем байте. В нём содержится 0x01, что говорит о том, что следующий байт является флагом.

Третий байт содержит сам флаг. Большинство iBeacon-устройств сообщают, что они способны работать в режиме общего распознавания, используя low-energy и non-low-energy режимы операции. Значение флага в таком случае будет 1A.

2) Length (1 байт, 0x1A, десятичное 26).

Данное поле хранит длину оставшейся части пакета. Все iBeacon-пакеты имеют длину 30 байт, но поле Length описывает только длину пакета после

первых трёх байт и одного байта для самого поля Length. Таким образом и получается значение 26.

3) Type (1 байт, 0xFF).

Согласно стандарту Bluetooth, значение данного поля предписывает, что последующие два байта хранят идентификатор компании, а оставшуюся часть пакета могут занимать данные в определённом изготовителем формате.

4) Company ID (2 байта, 0x004C).

Технически, с этого поля начинаются определяемые изготовителем данные. Спецификация Bluetooth требует, чтобы эти данные начинались с идентификатора компании. Для iBeacon-устройств это всегда идентификатор компании Apple.

Заголовки призваны представить то, какой будет полезная нагрузка которая будет следовать за ними [8]:

1) Beacon type (2 байта, 0x02-15).

Apple назначила значение для маячков, которое используется всеми iBeacon-устройствами. Первый байт содержит идентификатор протокола iBeacon (0x02), а второй байт – это длина оставшегося фрейма (0x15 или 21).

2) Proximity UUID (16 байт).

Поле содержит UUID маячка. Обычно данное поле будет содержать идентификатор организации, которая владеет маячком.

3) Major number (2 байта) и Minor number (2 байта).

Эти поля, каждое из которых по два байта, содержат минорное и мажорное числа, которые позволяют опознать маячок.

4) Measured power (1 байт).

Так как мощности передатчиков могут различаться, то каждый маячок калибруется с целью выявления среднего RSSI на расстоянии одного метра от устройства. Значение поля представляется в дополнительном коде.

2 Методы внутреннего позиционирования

2.1 Оценка расстояния до места

Показания уровня сигнала (RSSI) можно перевести в расстояние между маячками посредством теоретических или эмпирических моделей распространения радиоволн. Согласно [1], выбираем модель распространения One slope:

$$L(d) = L_{FS} + 10n\ lg\left(\frac{d}{d_0}\right),$$

где d_0 = 1 м,

L_{FS} – потери в свободном пространстве на расстоянии d_0,

n – коэффициент, зависящий от типа помещения, количества препятствий и их материала.

Параметр L_{FS} производители маячков обычно указывают самостоятельно. Параметр n определяется эмпирически. На практике, даже в условиях прямой видимости с маячком, параметр RSSI хаотично меняет своё значение, в результате чего без применения математического аппарата определить расстояние до маячка становится затруднительным. Это происходит из-за следующих факторов:

– положение и характеристики направленности излучения антенной маячка или приема пользовательского устройства;

– наличие экранирующих объектов на пути от маячка до принимающего устройства;

– нахождение в непосредственной близости объектов из материалов, хорошо отражающих радиосигнал, а также большое скопление маячков на одной территории, за счёт многолучевой интерференции с основным лучом.

В связи с этим перед дальнейшей обработкой необходимо усреднить значения RSSI с каждого из маячков. Настраиваем маячки на выдачу данных с максимальной частотой, накапливаем их в буфере, и с определенной периодичностью находим среднее значение RSSI для каждого из маячков. Этот процесс повторяется циклически. Следует учесть, что повышение частоты выдачи данных маячками снижает ресурс их работы от батареи. Повышение периода нахождения среднего RSSI с целью собрать больше данных, ведет к большей задержке перед выдачей информации пользователю. Следовательно, необходимо варьировать данные параметры ради получения низкого значения задержки и увеличения срока службы маячков без существенных потерь в точности.

Существует два основных подхода к оценке местоположения с использованием значений RSSI:

1) Метод «снятия отпечатков».

На интересующее нас пространство с заранее расставленными маячками накладывается сетка с заданным шагом, затем в каждой точке сетки определяются значения RSSI всех видимых маячков. Полученные от пользовательского устройства данные используются для определения местоположения, посредством нахождения наилучшего сопоставления.

2) Метод трилатерации.

Среди полученных значений RSSI выбираются три с лучшим значением, затем используя координаты этих маячков и информацию об удаленности от них, определяется текущее местоположение.

2.2 Трилатерация

В геометрии трёхмерная проблема трилатерации представляет собой нахождение координат точки пересечения трёх сфер, которые определяются путём решения системы уравнений. Чтобы упростить вычисления, полагаем, что центры всех трех сфер лежат в плоскости $z = 0$, один из них совпадает с

началом координат, второй — лежит на оси x. Наложенные ограничения не уменьшают общности: к такому виду может быть приведена любая система соответствующих уравнений путём перехода к другой системе координат. Чтобы найти решение в исходной системе координат, к решению, найденному в этой (приведенной) системе координат, применяются преобразования, обратные к тем, которые позволили исходное множество из трех точек привести в соответствие с ограничениями. Составим систему уравнений для трех сфер:

$$\begin{cases} r_1^2 = x^2 + y^2 + z^2 \\ r_2^2 = (x - d)^2 + y^2 + z^2 \\ r_3^2 = (x - i)^2 + (y - j)^2 + z^2 \end{cases}$$

Необходимо найти точку (x, y, z) удовлетворяющую всем трём уравнениям. Вычтем второе уравнение из первого и найдем x:

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d}.$$

Считаем, что первые две сферы пересекаются более, чем в одной точке, то есть:

$$d - r_1 < r_2 < d + r_1.$$

В этом случае, подставляя выражение x в уравнение первой сферы, получаем уравнение окружности, которое является искомым пересечением первых двух сфер:

$$y^2 + z^2 = r_1^2 - \frac{(r_1^2 - r_2^2 + d^2)^2}{4d^2}.$$

Подставляем $y^2 + z^2 = r_1^2 - x^2$ в уравнение третьей сферы и находим y:

$$y = \frac{r_1^2 - r_3^2 - x^2 + (x - i)^2 + j}{2j} = \frac{r_1^2 - r_3^2 + i^2 + j^2}{2j} - \frac{i}{j}x.$$

Зная x и y можно легко получить z:

$$z = \sqrt{r_1^2 - x^2 - y^2}.$$

Так как z выражается как положительный или отрицательный квадратный корень, у данной задачи может быть ноль, одно или два решения.

После определения местоположения методом трилатерации, можно заметить, что оно всё равно «скачет», но тем не менее на практике таким образом удалось получить точность в 3 метра [13]. Если раз в секунду выбирать маячок с лучшим RSSI, то при 10 последовательных измерениях получаем результаты, показанные на рисунке 3 (одно деление сетки равно 4 метрам):

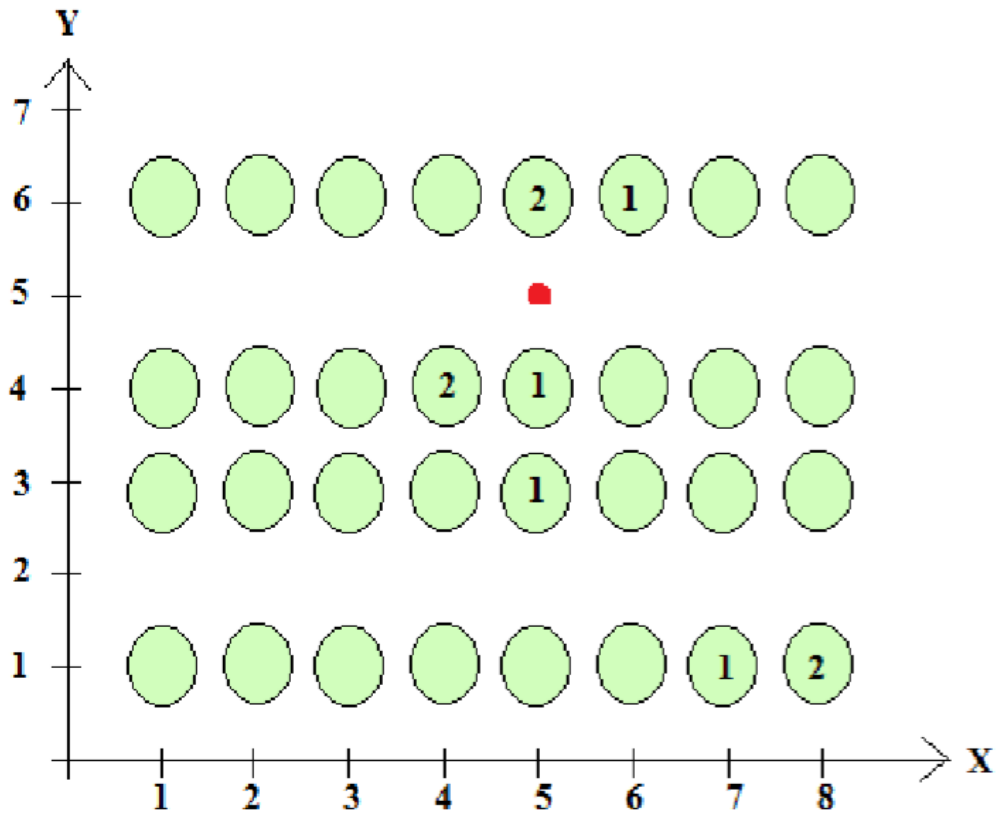

Рисунок 3 – Результат измерений положения устройства

Красной точкой отмечено местоположение устройства, координаты которого определяем. Зелёные круги представляют собой сеть Bluetooth-маячков. Числа внутри кругов показывают, сколько раз определённый маячок имел лучший RSSI за время проведения измерений [13].

Таким образом, требуется дальнейшая математическая обработка полученных результатов, чтобы нивелировать ошибку.

2.3 Метод снятия «отпечатков»

Отпечатком называют набор показателей RSSI маячков в некоторой точке. Создаётся координатная сетка с произвольным шагом деления (шаг деления зависит от требуемой точности определения местоположения). В каждой точке этой координатной сетки происходит снятие «отпечатков».

Такие «отпечатки» представляют собой вектор, элементами которого являются RSSI и соответствующие им маячки.

Во время самого процесса снятия «отпечатков» желательно произвести несколько измерений, меняя в течении процедуры угол поворота. Это необходимо для того, чтобы компенсировать флуктуации, которые возникают при повторных измерениях RSSI. Показания с одного и того же маячка могут значительно отличаться, если стоять к нему спиной или лицом.

Далее «отпечатки» усредняются и ассоциируются с определённой точкой координатной сетки. Впоследствии, при навигации, приложение один раз в определённый интервал снимает текущий «отпечаток», а также запрашивает все «отпечатки» из базы данных (они могут храниться локально, в облачном хранилище или специально разработанном сервере). Текущая позиция определяется путём поиска такого «отпечатка» из базы, который имеет с текущим наименьшую разницу. Проассоциированная с ним координата и будет считаться текущим местоположением устройства.

2.4 Фильтр Калмана

2.4.1 Процесс и оценки

Фильтр Калмана использует динамическую модель системы (например, физический закон движения), известные управляющие воздействия и множество последовательных измерений для формирования оптимальной оценки состояния. Фильтр убирает шумы измерения (случайные всплески) и выдаёт результат как с учетом результатов текущих измерений, так и с учётом предсказанных результатов на основе прошлых измерений. Алгоритм состоит из двух повторяющихся фаз: предсказание и корректировка. На первой рассчитывается предсказание состояния в следующий момент времени (с учетом неточности их измерения). На второй, новая информация с датчика

корректирует предсказанное значение (также с учётом неточности и зашумленности этой информации).

Фильтр Калмана рассматривает проблему оценки состояния процесса дискретного времени, которое может быть охарактеризовано линейным разностным уравнением:

$$x_k = Ax_{k-1} + Bu_k + w_k,$$

где A – матрица эволюции процесса/системы, которая воздействует на вектор x_{k-1} (вектор состояния в момент $k-1$);

B – матрица управления, которая прикладывается к вектору управляющих воздействий u_k;

w_k – нормальный случайный процесс с нулевым математическим ожиданием.

Для оценки также требуется измерение $z \in Rm$ следующим образом:

$$z_k = Hx_k + v_k.$$

Случайное значение w_k представляет шум процесса, ошибку между приближением и реальным процессом. Другая случайная величина, v_k, представляет шум измерений, ошибку между измерением Hx_k и действительным значением. Обе переменные считаются независимыми друг от друга, гауссовскими и белыми:

$$p(w) \sim N(0, Q),$$
$$p(v) \sim N(0, R).$$

В этом общем введении ковариация процессов шума Q и ковариация шума измерений R считаются постоянными по итерациям. Тем не менее, они

могут меняться на каждом шаге (на каждой итерации). Матрица A, размерности $n \times n$, представляет собой модель перехода состояния, которая связывает состояние на предыдущем временном шаге $k-1$ с состоянием на текущем этапе k. Матрица B, размерности $n \times l$, является управляющей моделью, которая связывает дополнительный управляющий вход $u \in Rl$ с состоянием. Матрица $m \times n$ H является моделью наблюдения, связывающей состояние с измерением z_k.

Теперь вводятся два определения. Первый, $\hat{x}_k^- \in R^n$, является априорной оценкой состояния на этапе k, данным знания процесса до шага k. Второй – $\hat{x}_k \in R^n$ и относится к оценке апостериорного состояния на шаге k, заданном измерением z_k.

Ошибки априорной и апостериорной оценки:

$$e_k^- \equiv x_k - \hat{x}_k^-,$$
$$e_k \equiv x_k - \hat{x}_k.$$

При этих ошибках ковариация априорной ошибки оценки и ковариация ошибок апостериорной оценки могут быть рассчитаны соответственно, как:

$$P_k^- = E[e_k^- e_k^{-T}],$$
$$P_k = E[e_k e_k^t].$$

При выводе уравнений для фильтра Калмана первый шаг – найти уравнения, которые вычисляют апостериорную оценку $\hat{x}_k$ как линейную комбинацию априорной оценки $\hat{x}_k^-$ – и взвешенную разность между фактическим измерением z_k и предсказанием измерения $H\hat{x}_k^-$, рассчитанной с использованием априорной оценки состояния.

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-).$$

Разность $(z_k - H\hat{x}_k^-)$ называется инновацией и измеряет несоответствие между прогнозируемым измерением и наблюдаемым измерением. Если нововведение равно нулю, это означает, что они находятся в полном согласии, поэтому предсказанное измерение является идеальным из-за априорной оценки состояния, также является совершенным.

Матрица K, размерности $n \times m$, является усилением Калмана, который минимизирует ковариацию апостериорной ошибки P_k. Общее выражение K в текущем временном шаге k равно:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}.$$

Рассматривая это выражение, можно извлечь некоторую информацию о поведении фильтра Калмана. Когда ковариация априорной ошибки $P_k -$ приближается к нулю, усиление Калмана также стремится к нулю, поэтому новшество не доверяется. Другой способ убедиться в том, что если P_k^- стремится к нулю, это потому, что оценка априорного состояния очень похожа на реальное значение состояния, а затем лучше доверять оценке, а не измерению. С другой стороны, если ковариация измерительного шума R приближается к нулю, это означает, что измерения имеют небольшую ошибку, поэтому лучше доверять им больше, чем оценке априорного состояния. Это также происходит, когда P_k^- очень велико, что означает, что измерения имеют небольшую ошибку по сравнению с предсказанием. В этом случае усиление Кальмана K_k стремится к H^{-1}.

2.4.2 Алгоритм

Фильтр Калмана оценивает процесс с использованием управления обратной связью. Во-первых, фильтр оценивает состояние процесса в определенное время и затем получает обратную связь в виде измерений

шумов. Учитывая это, уравнения можно разделить на две группы: уравнения корректировки и уравнения предсказаний. Первые из них определяют следующие по времени оценки состояния и ошибки ковариации, чтобы получить априорные оценки для следующего шага времени. После этого вторая группа уравнений вводит обратную связь. Конкретно они используют новое наблюдаемое измерение для коррекции априорной оценки и получения апостериорной оценки. Алгоритм и уравнения представлены на рисунок 4 [10].

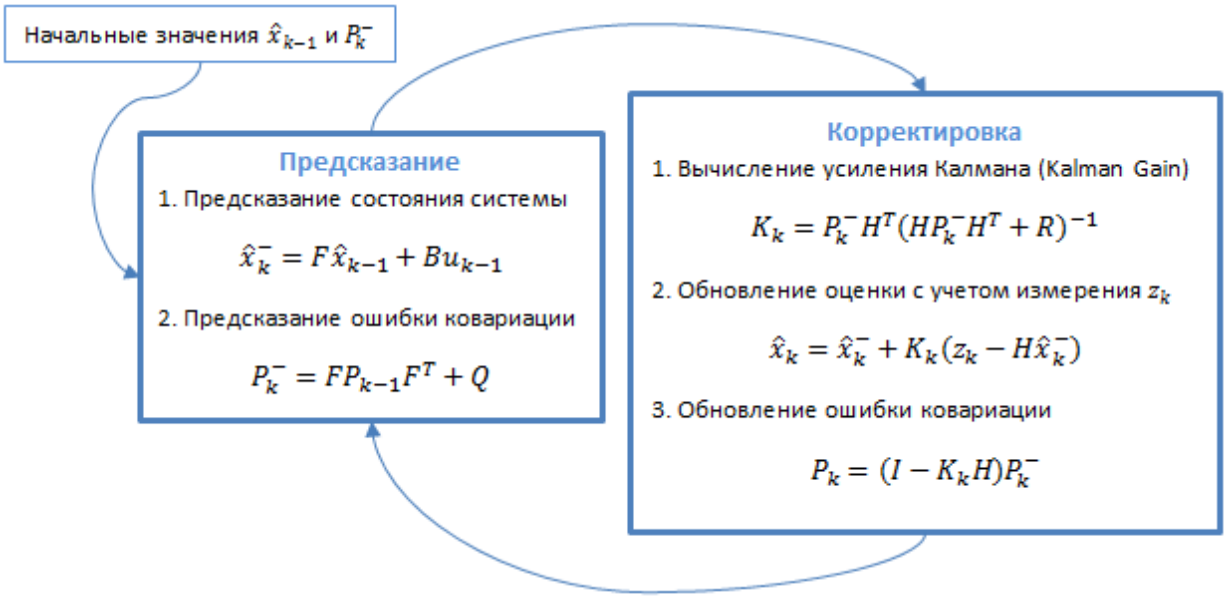

Рисунок 4 – Фильтр Калмана

где $\hat{x}_k^-$ – предсказание состояния системы в текущий момент времени;

$\hat{x}_{k-1}$ – состояние системы в прошлый момент времени;

F – матрица перехода между состояниями (динамическая модель системы);

B – матрица применения управляющего воздействия;

u_{k-1} – управляющее воздействие в прошлый момент времени;

P_k – предсказание ошибки;

P_{k-1} – ошибка в прошлый момент времени;

Q – ковариация шума процесса;

K_k – усиление Калмана;

H – матрица измерений, отображающая отношение измерений и состояний;

R – ковариация шума измерения;

z_k – измерение в текущий момент времени;

I – матрица идентичности.

Первое уравнение в уравнениях обновления времени – это то, которое отвечает за проецирование состояния на следующий шаг с использованием предыдущего состояния оценки и, опционального ввода. Второе уравнение - это то, которое проектирует ковариацию ошибок на шаг вперед. На этапе корректировки первое, что нужно сделать, – вычислить усиление Калмана. Тогда необходимо измерить процесс, чтобы получить z_k. При измерении z_k и оценке измерений $H\hat{x}_k^-$ – получаем обновление. Теперь, используя обновление и усиление Калмана, можно получить оценку апостериорного состояния, как показано во втором уравнении измерения. Третье и последнее уравнение порождают ковариационную оценку апостериорной ошибки. После обновления времени и измерения (или фазы предсказания и коррекции) процедура повторяется, проецируя предыдущие апостериорные оценки для получения новых априорных оценок. В фильтре имеется возможность учитывать управляющее воздействие. Например, таким управляющим воздействием может быть информация с акселерометра, что значительно улучшает результат (в таком варианте погрешность составляет уже не 3 метра, а 1-1,5 – тоже не мало, но связано отчасти с тем, что фильтр Калмана использует систему с заданным уравнением движения, а мы имеем дело с хаотичным движением).

3 Описание системы внутреннего позиционирования, основанной на технологии iBeacon

3.1 Аппаратные и программные требования

Приложение должно работать под управлением операционной системы iOS версии 10 и выше. Маячки и устройство, на котором будет работать приложение, должны быть оснащены Bluetooth LE.

В качестве маячков может использоваться любое устройство, которое имеет Bluetooth указанной версии и совместимое программное обеспечение, которое позволяет ему имитировать маячок, работающий по технологии iBeacon.

Для корректной работы приложению требуется доступ к Интернету, так как данные о маячках (идентификаторы, координаты) хранятся в облачном хранилище Firebase, предоставляемое компанией Google.

Перед применением приложения следует произвести настройку окружения.

Для использования функции навигации с помощью технологии iBeacon, необходимо смонтировать сеть маячков по сетке для наилучшего покрытия и наивысшей точности определения местоположения. После этого необходимо сконфигурировать приложение и добавить идентификаторы маячков, их калибровочное значение и координаты в хранилище, чтобы они были доступны для получения через Интернет. Для использования метода снятия «отпечатков» также необходимо составить координатную сетку и произвести само снятие «отпечатков» в каждой точке этой сетки.

Также следует убедиться, что каждый маячок имеет уникальную комбинацию трёх полей: proximity UUID, major, minor. В противном случае, поведение приложения не определено.

3.2 Вспомогательное приложение

Из-за возможного отсутствия маячков было написано вспомогательное приложение, которое позволяет устройствам под управлением iOS версии 10 и выше и с наличием BLE функционировать в качестве маячка, работающего по технологии iBeacon, интерфейс которого изображён на рисунке 5.

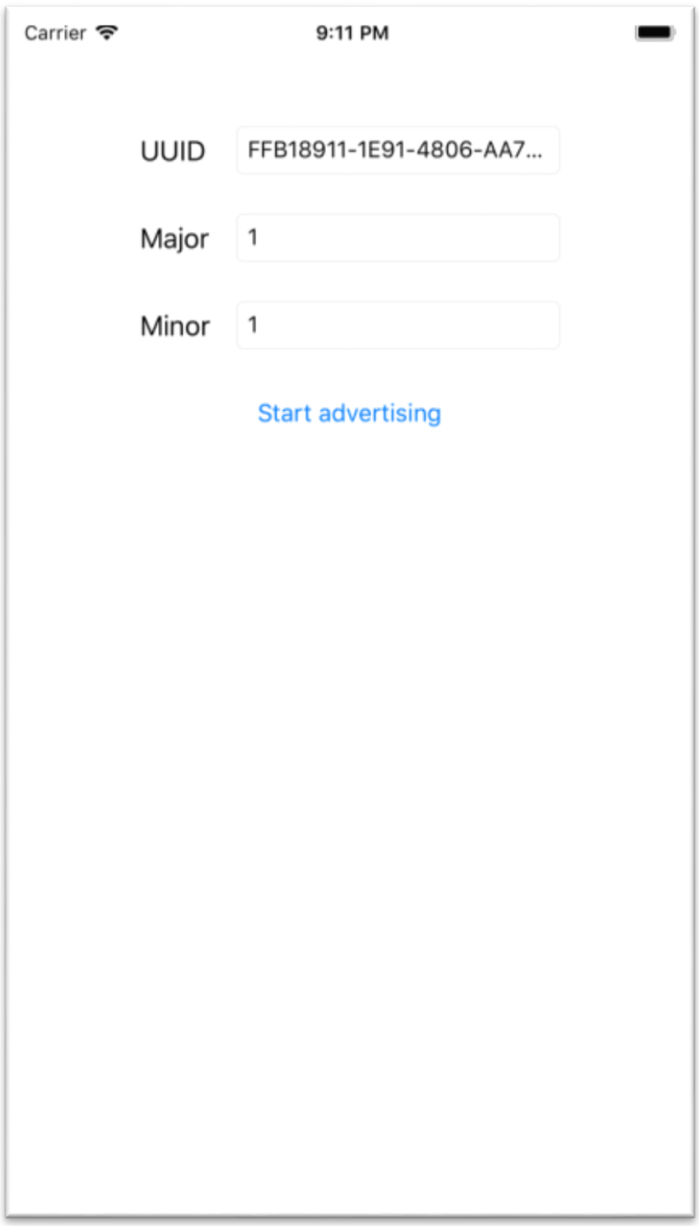

Рисунок 5 – Скриншот интерфейса приложения, позволяющего iPhone функционировать в качестве Bluetooth-маячка

В приложении доступно четыре поля: для ввода идентификаторов маячка (proximity UUID, major number, minor number) и калибровочного значения. Все эти значения приложение будет широковещательно рассылать, позволяя таким образом другим устройствам распознать его как маячок.

Ниже приведён фрагмент, отвечающий за формирование и широковещательную рассылку AP.

```
let region = CLBeaconRegion(proximityUUID: uuid, major: major, minor: minor,
identifier: "com.smedilink.beacon")

let payload = region.peripheralData(withMeasuredPower: nil)

peripheralManager.startAdvertising(payload as? [String : Any])
```

После того, как пользователь ввёл данные в поля ввода и нажал на кнопку «Start advertising», вызывается метод, который считывает эти значения и записывает в константы uuid, major, minor. Впоследствии эти константы и строковый литерал "com.smedilink.beacon" используются для конструирования объекта класса CLBeaconRegion. Это объект необходим, чтобы сформировать широковещательный пакет (advertising packet) с помощью метода peripheralData(withMeasuredPower:), в качестве параметра которому передаётся nil. В результате будет сформирован словарь NSMutableDictionary с полученными ранее значениями полей и калибровочным значением по умолчанию. Широковещательная рассылка пакета (advertising packet) начнётся после вызова метода startAdvertising(_:), в качестве параметра которому будут переданы данные, полученные на предыдущем шаге и конвертированные к типу Dictionary<String, Any>.

3.3 Общая архитектура

Написанное приложение позволяет использовать встроенные Bluetooth-модуль для навигации внутри помещения.

Мобильное приложение для локации написано на языке Swift. Данный язык является вторым официальным языком для iOS после Objective-C.

В процессе написания приложения были применены следующие шаблоны проектирования.

– Dependency injection или DI – недавно сформировавшийся подход в управлении зависимостями, который используется для увеличения модульности приложения и ещё большей изоляции отдельных компонентов друг от друга. Данная техника предписывает создавать все крупные компоненты в одном месте. В дальнейшем, созданные зависимости передаются предписанным объектам с помощью использования абстрактных фабрик и фабричных методов. Классы, которые являются фабриками или имеют фабричные методы, называют assembly (сборка) или assembler (сборщик). Таким образом, компоненты приложения не знают о том, как создаются их зависимости, а, значит, их легко можно поменять в случае необходимости, если интерфейс компонента остаётся прежним.

– Стратегия (Strategy) — поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путём определения соответствующего класса. Шаблон позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют [11].

– Одиночка (Singleton) – порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру [11].

– VIPER (View, Interactor, Presenter, Entity, Router) – архитектурный подход к разбиению компонентов приложения на модули. Оригинальная архитектура была предложена специалистами компании Mutual Mobile [5]. Её диаграмма изображена на рисунке 6.

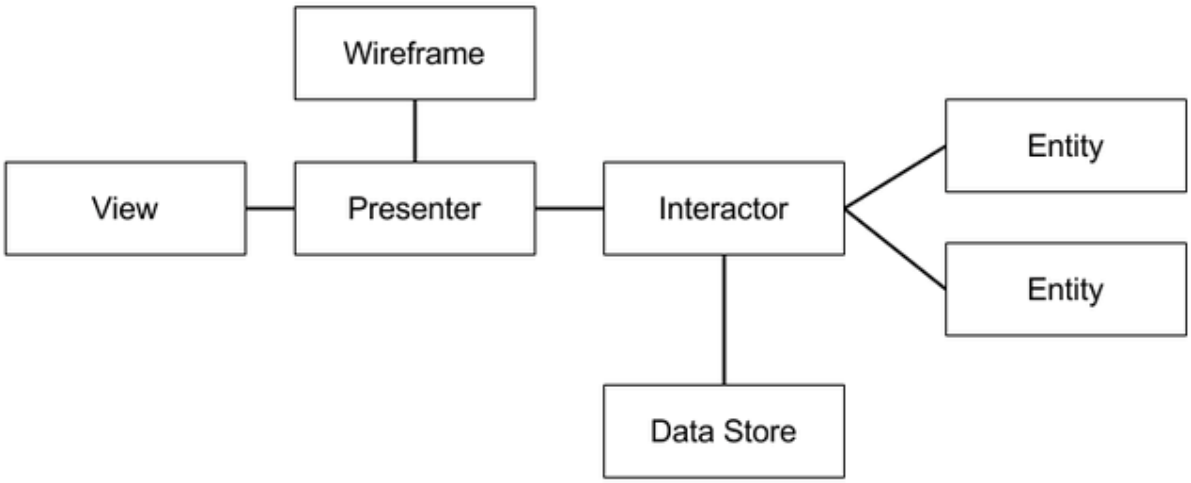

Рисунок 6 – Схема оригинальной архитектуры VIPER от Mutual Mobile

VIPER был предложен в качестве альтернативы MVC, основной и официальной архитектуры для iOS-приложений. Она была призвана решить проблему так называемых «Massive view controller», которая заключалась в том, что один из компонентов MVC, контроллер, был слишком универсален, поэтому разрастался до огромных размеров, и один единственный класс мог насчитывать несколько тысяч строк кода.

Схема, предложенная Mutual Mobile, также была не без недостатков. Например, компонент Wireframe имел сразу две зоны ответственности: он должен был собирать модуль и производить роутинг. Заметив эти недостатки, специалисты из Rambler предложили свой вариант архитектуры, в котором Wireframe разбивался на два компонента: Router, отвечающий за переход между модулями, и Assembly, отвечающий за сборку модуля и

осуществляющий инъекцию зависимостей. Её диаграмма изображена на рисунке 7.

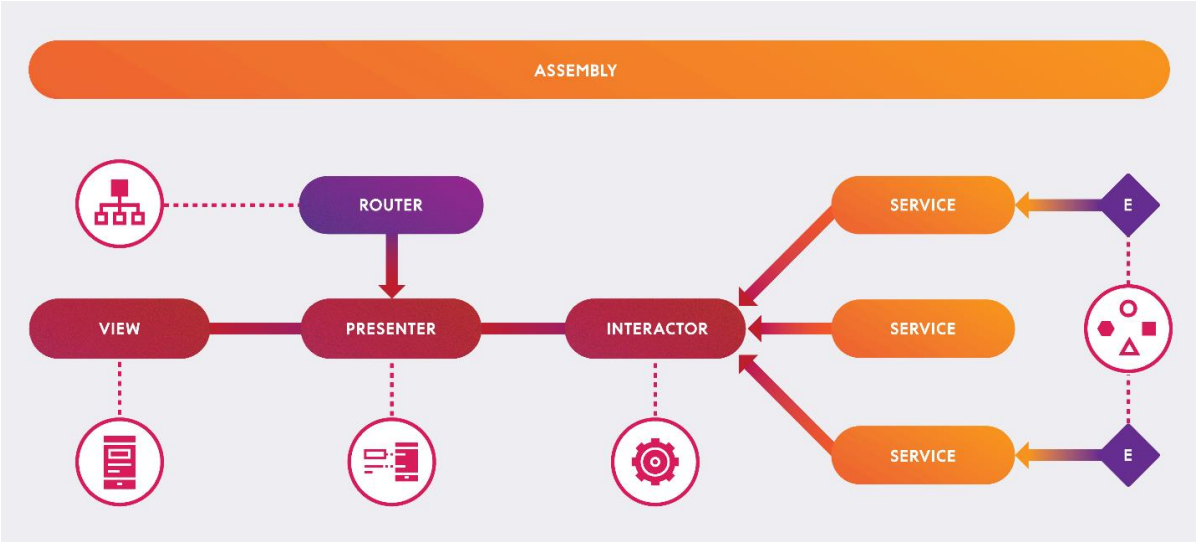

Рисунок 7 – Схема VIPER от Rambler

Впоследствии архитектура была принята в компании S Media Link в качестве основной и модернизирована. В подходе, используемом в SML, был удалён Interactor, а с сервисы были разделены на два подслоя, содержащие классы типа usecase и gateway. Её диаграмма изображена на рисунке 7.

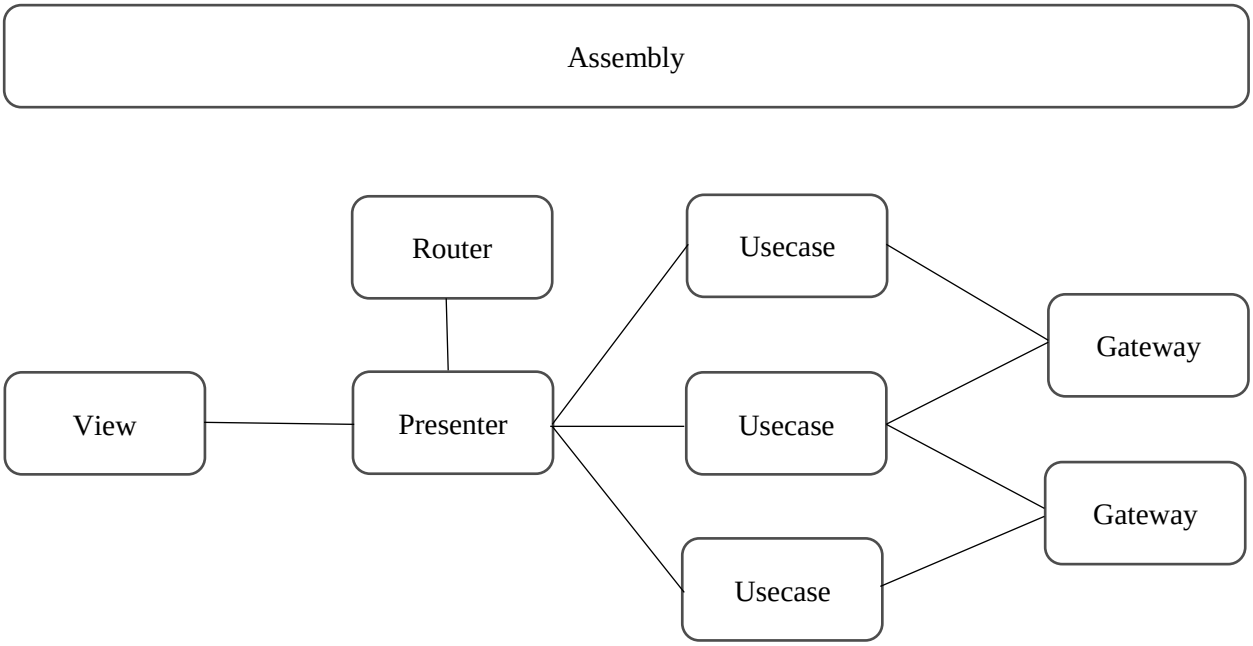

Рисунок 8 – Схема VIPER от S Media Link

– Сервис-ориентированная архитектура (Service-oriented architecture или SOA) – подход, при котором компоненты приложения распределяются по слабосвязанным слоям. Причём зависимости каждый слой знает только интерфейс зависимых слоёв, но не их реализацию. Таким образом, слои зависят только от интерфейсов нижележащих слоёв. Исключением является самый нижний слой, который может содержать внешние зависимости.

3.4 Используемые сторонние библиотеки

Внешние библиотеки управляются с помощью двух менеджеров зависимостей: Carthage и CocoaPods. Это вызвано особенностями их работы и методами распространения библиотек.

Основные используемые сторонние библиотеки приведены в таблице 1.

Таблица 1 – Основные используемые сторонние библиотеки

Название библиотеки	Краткое описание	Для чего используется
RxSwift, RxCocoa, RxCoreLocation	Набор библиотек, позволяющий писать асинхронный код в декларативном стиле. RxSwift содержит базовые классы и протоколы для написания реактивного кода. RxCocoa и RxCoreLocation – обёртки над системными фреймворками, расширяющими системные классы и встраивающие Rx-парадигму в них.	Они призваны сделать код более читаемым и легкомодифицируемым.
ViperKit	Фреймворк, предоставляющий базовые классы и протоколы для построения VIPER-модулей.	Используется в слое Presentation для построения VIPER-модулей.
EasyDi	Быстрый и простой фреймворк, реализующий шаблон инъекции зависимостей.	Необходим для увеличения уровня абстракции и переиспользования.
SwiftyBeaver	Библиотека, позволяющая организовать форматированный вывод информации в консоль.	Используется для отладочных целей.
PKHUD	Библиотека с компонентами для отображения всплывающих информационных окон (окна ошибки, прогресса, успеха).	Необходима для пояснения состояния приложения для пользователя.
Firebase	Облачная платформа от компании Google, предоставляющая различные сервисы по хранению и обработке данных, хостингу.	Требуется, чтобы хранить данные о маячках.

Помимо указанных выше, используются ещё несколько библиотек, но они не влияют на возможности приложения.

3.5 Архитектура приложения.

Приложение состоит из трёх слоёв.

– Слой Core содержит внешние зависимости и внутренние обёртки над внешними сервисами. Через данный слой происходит всё взаимодействие с внешней средой.

– Слой BusinessLogic включает в себя бизнес-логику приложения. Отвечает за обработку данных с маячков и вычисление местоположения устройства. Состоит из двух подслоёв, верхний из которых состоит из сущностей типа usecase, а нижний – gateway.

Usecase является внешним интерфейсом слоя бизнес-логики и доступен слою представления через протоколы (аналог интерфейсов из языка Java). Каждый usecase представляет собой класс с небольшим количеством публичных методов, объект которого представляет собой один из вариантов использования приложения. Таким образом, каждый usecase реализует только одну функцию приложения.

Gateway представляет собой низкоуровневый объектов, содержащих публичные методы и поля, необходимые для выполнения примитивных операций в контексте приложения.

– Слой Presentation содержит компоненты, управляющие пользовательским интерфейсом. Представляет собой набор VIPER-модулей, объединённых в user story. User story являются сценариями поведения, то есть объединяют в себя все те компоненты уровня представления, которые необходимы для реализации определённой функциональности приложения.

3.5.1 Слой Core

Содержит вспомогательные классы и обёртки над системными классами. В этом слое реализуется надстройка над фреймворком EasyDi,

которая позволяет использовать его для реализации VIPER Assembly (рисунок 8), а также обёртка над SwiftyBeaver.

3.5.2 Слой BusinessLogic

Содержит в себе объекты типа usecase и gateway, а также соответствующие им протоколы, сущности уровня бизнес-логики (модели).

В приложении используются следующий структуры данных:

– BeaconDescriptor (описатель маячка);
– Point, Size, Rect (точка, размер, прямоугольник);
– Beacon (маячок);
– BeaconRegion (область маячков);
– Fingerprint (отпечаток).

BeaconDescriptor представляет собой набор полей uuid, major, minor, по которым можно опознать каждый конкретный маячок.

Структуры Point, Size и Rect представляют собой геометрические двумерные сущности, которыми манипулирует приложение. Point – это точка в пространстве, которая имеет поля x и y. Size представляет размеры некоторой области, представленные полями width и height. Структура Rect – это комбинация двух предыдущих структур. Она содержит поле origin и поле size, которые представляют координаты начала и размер прямоугольника, соответственно.

Beacon содержит всю информацию необходимую приложению об одном маячке. Содержит поля типов descriptor с описателем данного маячка, position с координатами установки и txPower, которое хранит калибровочное значение.

BeaconRegion инкапсулирует данные, необходимые для функционирования метода снятия «отпечатков». В эту структуру включены уникальный для системы идентификатор региона id (задаётся пользователем), область региона (rect), а также массив отпечатков fingerprints, элементами которого являются «отпечатки».

Fingerprint – это псевдоним (typealias) для словаря, ключами которого являются описатели маячков, а значениями – соответствующие RSSI.

Вычисление местоположения производятся с помощью объектов, классы которых спроектированы согласно шаблону Команда. В коде объявлен интерфейс для алгоритмов определения местоположения TrackingAlgorithm с единственным методом computePosition(with:_), параметром которого является словарь, содержащий в качестве ключей описатели маячков, а в качестве значений – RSSI. Данный метод возвращает структуру Point, которая является предполагаемым местоположением устройства; либо nil, если данных для определения местоположения недостаточно.

Интерфейс TrackingAlgorithm реализуют два класса: FingerprintTrackingAlgorithm и TrilaterationTrackingAlgorithm.

Реализация определения местоположения по методу «отпечатков» в классе FingerprintTrackingAlgorithm:

```
func computePosition(with measures: [BeaconDescriptor: Int]) -> Point? {
    if beaconRegions.isEmpty { return nil }

    let mostStrongSignal = measures.reduce(0) { max($0, Double(abs($1.value))) }
    let currentFingerprint = Dictionary(uniqueKeysWithValues: measures.map { ($0, Double($1) / mostStrongSignal) })

    var minDiff = Double.greatestFiniteMagnitude
    var minDiffInd = 0
    for (ind, beaconRegion) in beaconRegions.enumerated() {
       for fingerprint in beaconRegion.fingerprints {
          let diff = compareFingerprints(currentFingerprint, fingerprint)
          if diff < minDiff {
             minDiff = diff
             minDiffInd = ind
          }
       }
    }

    let rect = beaconRegions[minDiffInd].rect
```

```
        return Point(
            x: rect.x + rect.width / 2.0,
            y: rect.y + rect.height / 2.0
        )
    }
```

Реализация определения местоположения с помощью трилатерации в классе TrilaterationAlgorithm:

```
func computePosition(with measures: [BeaconDescriptor: Int]) -> Point? {
        let three = measures
            .filter { self.beacons[$0.key] != nil }
            .sorted { $0.value < $1.value }
            .prefix(3)
            .map { (self.beacons[$0], $1) }
            .map { (beacon: $0, distance: computeDistance(basedOn: $1, with:
$0.txPower)) }
            .sorted { $0.beacon.position.x < $1.beacon.position.x }

        guard three.count == 3 else { return nil }

        let (first, second, third) = translateCoords(
            three[0].beacon.position,
            three[1].beacon.position,
            three[2].beacon.position
        )

        let x = (pow(three[0].distance, 2) - pow(three[1].distance, 2) + pow(second.x,
2)) / (second.x * 2)
        let y = (pow(three[0].distance, 2) - pow(three[1].distance, 2) + pow(third.x, 2)
+ pow(third.y, 2)) / (second.x * 2) * x
            - third.x / third.y

        return Point(
            x: three[0].beacon.position.x + x,
            y: three[0].beacon.position.y + y
        )
    }
```

Внешний интерфейс слоя бизнес-логики позволяет добавлять данные о маячках во внешнее хранилище, производить их калибровку (вычислять

measured value); снимать отпечатки и сохранять их в удалённое хранилище, определять местоположение устройства с помощью методов трилатерации и снятия отпечатков.

Нижний подслой реализует примитивные функции, такие как получение/отправка данных с/на Firebase, управление локальным кешем, управление Bluetooth-адаптером через API операционной системы.

3.5.3 Слой Presentation

Содержит в себе user story, разбитые на VIPER-модули, модели для них, элементы пользовательского интерфейса. Список user story:

- BeaconConfig;
- Drawer;
- FingerprintConfig;
- Main.

BeaconConfig включает в себя модуль по добавлению маячка, а также его калибровки. Внешний вид экрана данного модуля можно увидеть на рисунке 9.

Рисунок 9 – Интерфейс экрана добавления и калибровки маячка

Каждое поле представляет собой часть структуры Beacon, которая инкапсулирует всю необходимую приложению информацию для работы с маячком. Поля «x» и «y» служат для обозначения позиции маячка. В поля «Proximity ID», «Major», «Minor» вносятся соответствующие значения, которые будут рассылаться маячком.

При нажатии на кнопку «Submit» выполняется валидация данных с формы и отправка их на сервер.

Кнопка «Calibrate» служит для вычисления калибровочного значения маячка. Если же не выполнять калибровку, а сразу отправить данные в Firebase, то в структуру данных маячка будет записано значение по умолчанию – -59.

В userstory Drawer реализовано боковое меню, которое позволяет переключаться между экранами. С его помощью можно будет получить доступ к трём экранам: экран добавления и конфигурации маячка, экран добавления и конфигурации региона и к основному экрану приложения, на котором реализована функция внутреннего позиционирования.

Оно имеет вид, как показано на рисунке 10:

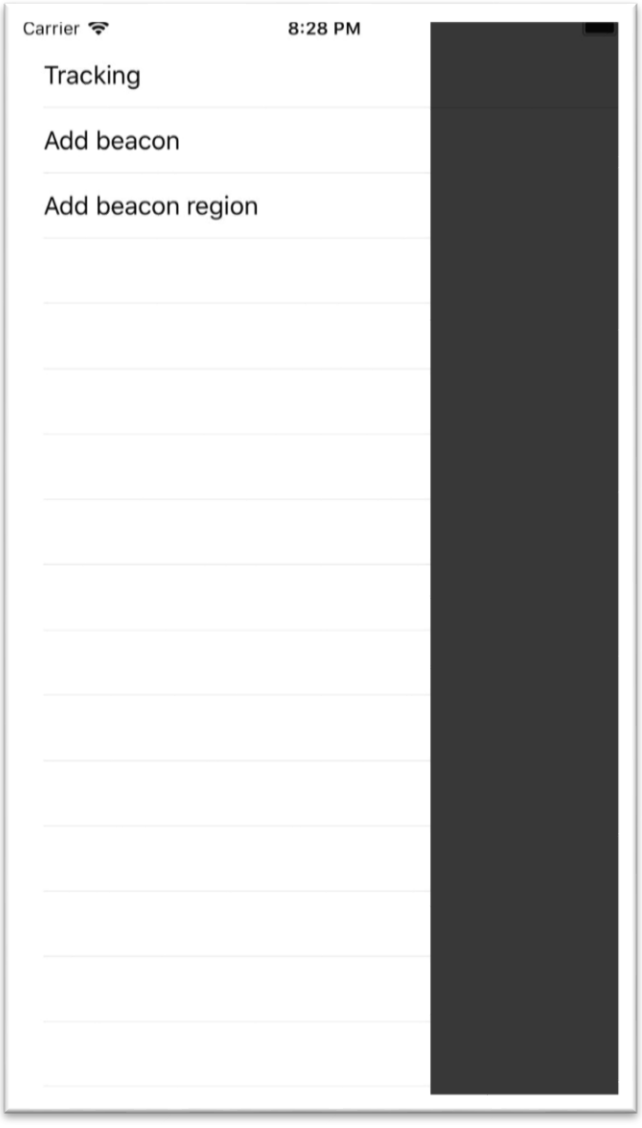

Рисунок 10 – Внешний вид бокового меню

FingerprintConfig выглядит аналогично BeaconConfig и позволяет добавлять во внешнее хранилище «отпечатки». Его интерфейс изображён на рисунке 11.

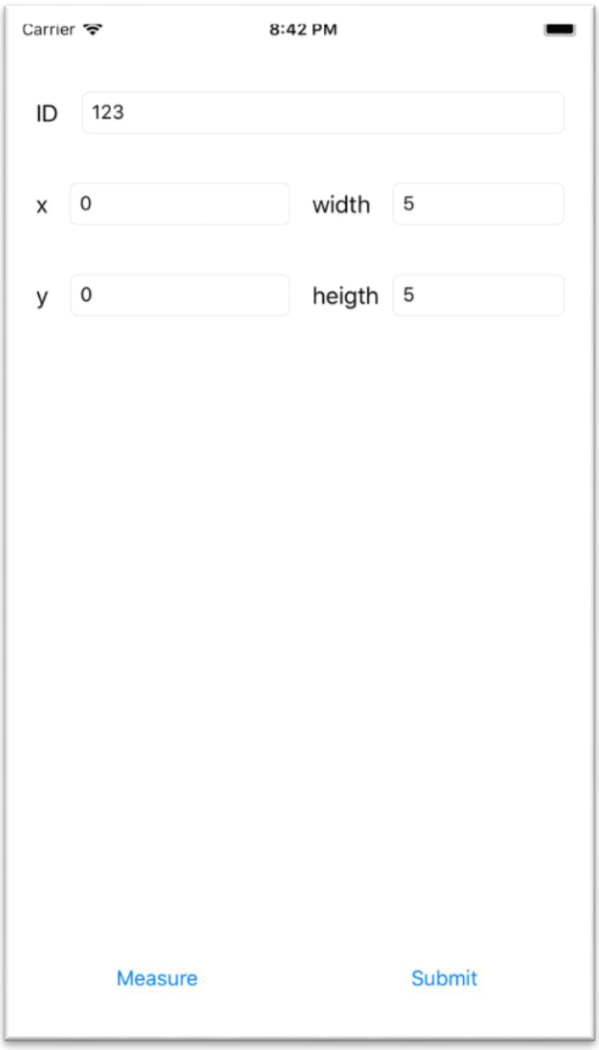

Рисунок 11 – Интерфейс экрана добавления и снятия «отпечатка»

Процесс работы приложения на этом этапе следующий:

1. Сначала вводятся данные об участке, с которого будет сниматься «отпечаток» (его размеры, точка начала участка, его имя в системе).

2. По нажатию на кнопку «Measure» происходит сам процесс снятия «отпечатка».

3. Полученные данные отправляются на внешнее хранилище, когда пользователь нажмёт кнопку «Submit». Если при это снятие отпечатка не было произведено, то будет отправлен «пустой» (нулевого размера) отпечаток.

Userstory Main является основной, и именно через неё пользователю предоставляется доступ к основной функции приложения – навигации внутри помещения. Её интерфейс представлен на рисунке 12.

Рисунок 12 – Интерфейс экрана навигации

Внутри данной userstory реализован единственный модуль, который имеет то же имя.

В целях удобства сравнительного анализа местоположение

предоставляется пользователю в виде пары координат, а не в виде схемы с точкой, обозначающей текущую позицию.

Координаты x и y показывают текущее местоположение, полученное с помощью одного из методов внутреннего позиционирования.

С помощью переключателя можно сменить алгоритм позиционирования.

4 Сравнительный анализ методов внутреннего позиционирования

Были реализованы следующие методы навигации в помещении:

– Навигация с помощью трилатерации на основе технологии iBeacon.

– Навигация с помощью «отпечатков» на основе технологии iBeacon.

Сравнительный анализ проводится по следующим критериям:

– точность;

– стоимость разработки, установки и поддержки системы (человеческие, временные и финансовые ресурсы);

– гибкость (простоту масштабирования, поддержки);

– сложность алгоритма.

Для проведения эксперимента по вычислению средней погрешности определения местоположения была выбрана решётчатая топология, так как она обеспечивает наиболее равномерное покрытие.

В качестве маячков выступают пять устройств iPhone и одно устройство iPad с установленным на них вспомогательным приложением, которое позволяет им функционировать в качестве iBeacon-маячков. Для краткости в дальнейшем в тексте они будут упоминаться как маячки.

На рисунке 13 представлена схема, изображающая расположение маячков. Начало координат находится в там же, где расположен верхний левый маячок. Цена деления – один метр. Ось ординат имеет направление сверху вниз, а ось абсцисс – слева направо.

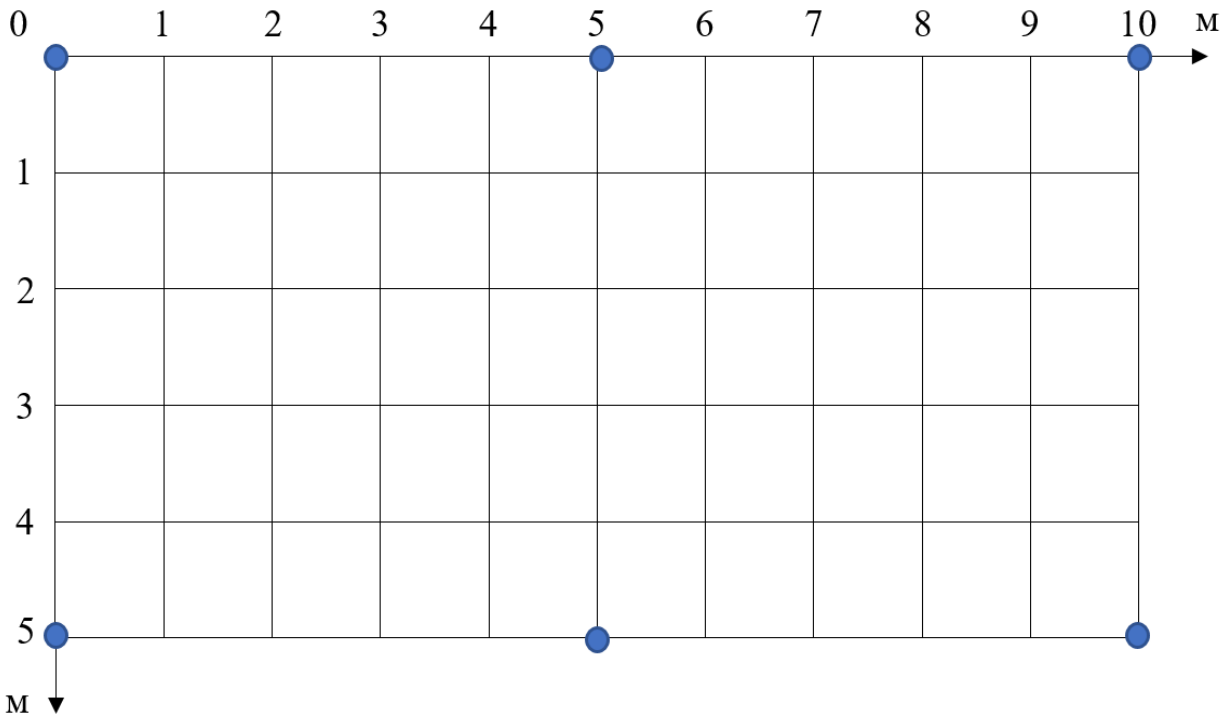

Рисунок 13 – Схема расположения маячков

Синими точками на рисунке изображены расположенные маячки. Расстояние между двумя соседними маячками равно ~5 метра.

Эксперимент проводился в офисном помещении, а это значит, что на пути радиосигнала находятся препятствия: столы, полки, перегородки. Приблизительная схема помещения, а также преграды в виде прямоугольников показаны на рисунке 14.

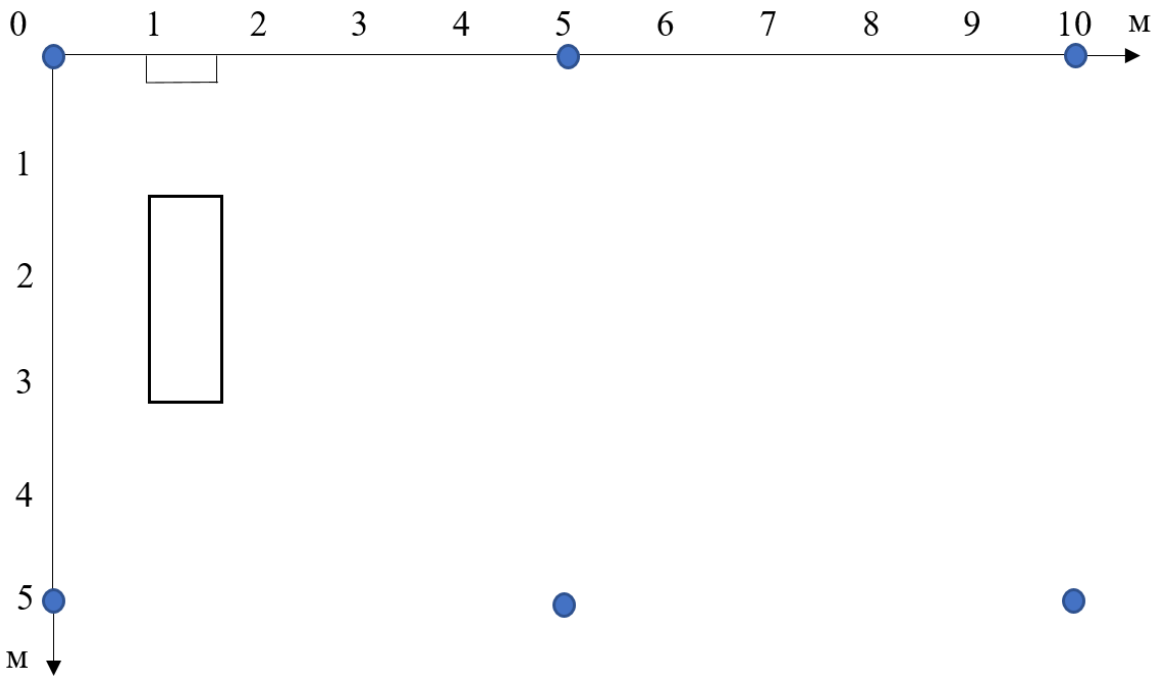

Рисунок 14 – Приблизительная схема помещения

Тестовый участок был поделён на 50 регионов размера 1×1 м^2, в каждом из которых было снято четыре отпечатка в четырёх разных ориентациях с разницей в $\sim 90^o$.

В рамках эксперимента было проведено 20 измерений обоими методами. Результаты показаны в таблице 2.

Таблица 2 – Результаты эксперимента по определению местоположения

Фактические координаты	Трилатерация	Погрешность	Снятие «отпечатков»	Погрешность
(0; 0)	(0,57; 0,42)	0,71	(2; 2)	2,83
(5; 0)	(6,63; 1,03)	1,93	(3; 0)	2
(10; 0)	(8,13; 0,19)	1,88	(9; 0)	1
(10; 5)	(8,88; 4,96)	1,12	(10; 3)	2
(5; 5)	(5,25; 6,09)	1,12	(9; 1)	2,24
(0; 5)	(0,71; 5,01)	0,71	(1; 5)	1
(1; 1)	(2.87; 1.44)	1,92	(3; 4)	2,24
(2; 2)	(3,23; 1,14)	1,5	(0; 2)	2
(3; 3)	(5,11; 2,22)	2,25	(3; 3)	0
(4; 4)	(5,4; 3,92)	1,4	(3; 2)	2,24
(5; 3)	(4,15; 1,76)	1,45	(3; 4)	2,24

Продолжение таблицы 2

(6; 1)	(4,69; 0,27)	1,5	(8; 2)	2,83
(7; 2)	(5,43; 1,03)	1,85	(9; 4)	2,83
(8; 3)	(7,06; 1,54)	1,12	(10; 5)	2,83
(9; 4)	(8,51; 2,99)	1,12	(10; 4)	2
(0; 3)	(1,7; 4,16)	2,06	(0; 3)	0
(3; 3)	(1,28; 3,02)	1,72	(0; 3)	0
(6; 3)	(6,81; 3,77)	1,12	(4; 3)	2
(10; 3)	(9,5; 3)	0,5	(10; 4)	1

По данным из таблицы вычисляется средняя погрешность для обоих алгоритмов: для трилатерации она равна ~1,4 м, а для снятия «отпечатков» – ~1,8 м.

Стоимость разработки, установки и поддержки систем, основанных на исследуемых методах почти идентична. Накладные расходы на содержание системы, основанной на методе трилатерации могут быть немного ниже ввиду того, что при выведении из строя одного из маячков, система будет продолжать функционировать без внесения серьёзных погрешностей в результаты определения местоположения, в то время как система, основанная на снятии «отпечатков», не сможет продолжить своё нормальное функционирование.

Система, работающая с использованием метода трилатерации, оказалась более гибкой по сравнению с системой, работающей по методу снятия «отпечатков». Это обусловлено тем, что для использования трилатерации не требуется никаких дополнительных настроек при перестроении топологии маячков, замене выведенного из строя маячка или расширении сети. Приложение может самостоятельно получать данные об изменениях с сервера и синхронизировать локальный кеш маячков с серверным. В случае использования метода снятия «отпечатков» при любом изменении сети, качественном или количественном, требуется заново снимать «отпечатки», что можно занять продолжительный период времени в случае больших помещений или групп помещений.

Ниже приведена оценка сложности алгоритма трилатерации.

На самом первом шаге работы метода трилатерации из полученных измерений RSSI выбираются три, имеющих наибольшее значение. Чтобы это осуществить, сначала отсеиваются показания тех маячков, которые для нас неизвестны, что можно оценить как $O(n)$, где n – количество обнаруженных маячков. Затем полученные пары описатель-RSSI сортируются по убыванию с помощью быстрой сортировки, что в среднем занимает $O(n \log n)$, где n – уже количество маячков, оставшихся после фильтрации. В случае, если в помещении не установлено незарегистрированных маячков с тем же proximity UUID, будет равно n из предыдущей оценки. Далее берутся первые три значения (если они есть). Эта операция выполнятся за константное время. В целом сложность приведённых выше действий можно оценить как $O(n + n \log n)$. Приведение координат из оригинальной системы к упрощённой, расчёт местоположения, трансляция полученного результата в оригинальную систему координат выполняется за константное время.

Таким образом, финальная оценка сложности метода трилатерации:

$$O(n + n \log n),$$

где n – количество маячков, пакеты которых были получены на текущем этапе измерений.

Теперь будет приведена оценка сложности для метода снятия «отпечатков».

Сначала происходит снятие текущего «отпечатка». Этот процесс занимает $O(2n)$ времени, так как сенсорные показания для каждого маячка просматриваются два раза: первый – при нахождении наибольшей силы сигнала, второй – при вычислении относительной силы сигнала. В последующем полученный вектор сравнивается со всеми известными «отпечатками», что может иметь оценку $O(mn)$, где m – общее количество отпечатков по все регионам, n – количество маячков в одном «отпечатке»

(в среднем, при регулярной топологии и идентичности маячков, n будет одинаковым для всех отпечатков).

Финальной оценкой сложности метода снятия отпечатков можно положить:

$$O(2n + mn),$$

где n – это среднее количество маячков, присутствующих в «отпечатке»;

m – количество отпечатков.

При этом количество отпечатков m для корректного функционирования метода должно быть больше, чем количество маячков. Таким образом, получается, что сложность метода снятия «отпечатков» получается квадратичная от n и выше, в то время как сложность трилатерации логарифмическая. Следовательно, сложность метода «снятия» отпечатков выше, чем у трилатерации.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы были рассмотрены основные технологии определения местоположения в помещении. Для выполнения работы была выбрана технология iBeacon, потому что она была специально разработана для реализации систем внутреннего позиционирования. В качестве методов внутреннего позиционирования были рассмотрены метод снятия «отпечатков» и трилатерация.

В результате выполнения выпускной квалификационной работы была разработана система внутреннего позиционирования, основанная на технологии iBeacon, а также методе снятия «отпечатков» и трилатерации. Для этого были разработаны два приложения: первое – вспомогательное, которое позволяет iOS-устройствам функционировать в качестве маячков, работающих по технологии iBeacon; второе – основное, позволяющее заносить данные о маячках в облачное хранилище, снимать «отпечатки» и показывать координаты устройства, на котором оно запущено, с помощью указанных методов. В качестве веб-сервера выступает облачное хранилище Firebase от компании Google.

На основе данных, полученных от основного приложения, был проведён сравнительный анализ выбранных методов. По его итогам было выявлено, что

– система, в основе которой заложена трилатерация, является более гибкой, то есть она может продолжить функционировать без значительных искажений местоположения в случае выхода из строя малого количества маячков, а также не требует дополнительной настройки при изменении топологии и/или добавлении новых маячков в сеть;

– средняя точность метода трилатерации выше, чем у метода снятия «отпечатков».

Также было выявлено, что сложность алгоритма снятия «отпечатков» выше, чем у алгоритма трилатерации.

Учитывая всё вышеперечисленное, можно сделать следующий вывод: в проектах по разработке систем локальной навигации лучше использовать метод трилатерации, чем метод снятия «отпечатков».

На основе полученных результатов в компании ООО «ИНТ-Сервис» ведётся разработка внутреннего продукта для внутреннего позиционирования устройств компании.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1) iBeacon. Мифы и реальность [электронный ресурс], URL: https://habr.com/post/278689/ (дата обращения: 29.04.2018)

2) An adaptive indoor positioning system based on Bluetooth Low Energy RSSI [электронный ресурс], URL: http://www.academia.edu/7653732/An_adaptive_indoor_positioning_system_based_on_Bluetooth_low_energy_RSSI (дата обращения: 29.04.2018)

3) Желамский М.В. Электромагнитное позиционирование. Преимущества и области применения. – ЭЛЕКТРОНИКА: НТБ, 2007, №3.

4) Антонович К. М. Использование спутниковых радионавигационных систем в геодезии: монография: в 2 т.. — М.: Картгеоцентр, 2005. — Т. 1.

5) Meet VIPER: Mutual Mobile’s application of Clean Architecture for iOS apps [электронный ресурс], URL: https://mutualmobile.com/posts/meet-viper-fast-agile-non-lethal-ios-architecture-framework

6) Книга VIPER [электронный ресурс], URL: https://www.gitbook.com/download/pdf/book/etolstoy/the-book-of-viper

7) Gast M. S. Building Applications with iBeacon, O’Relly, 2014.

8) Bluetooth Technology Website [электронный ресурс], URL: https://www.bluetooth.com/specifications (дата обращения: 03.05.2018)

9) An Introduction to the Kalman Filter [электронный ресурс], URL: https://cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf (дата обращения: 03.05.2018)

10) Фильтр Калмана — Введение [электронный ресурс], URL: https://habr.com/post/140274/ (дата обращения: 06.05.2018)

11) Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес, Приёмы объектно-ориентированного проектирования. Паттерны проектирования, Спб: Питер, 2001. – 368 с.

12) Фильтр Калмана [электронный ресурс], URL: https://m.habr.com/post/166693/ (дата обращения: 06.05.2018)

13) Навигация в помещениях с iBeacon и ИНС [электронный ресурс], URL: https://habr.com/post/245325/ (дата обращения: 10.05.2018)

14) CoreBluetooth framework [электронный ресурс], URL: https://developer.apple.com/documentation/corebluetooth (дата обращения: 04.05.2018)

ПРИЛОЖЕНИЕ А

Основные классы приложения для навигации в помещении

Файл AppDelegate.swift:

```
import UIKit
import FirebaseCore

typealias LaunchOptions = [UIApplicationLaunchOptionsKey: Any]

class AppDelegate: UIResponder, UIApplicationDelegate {

   var mainLaunchRouter: LaunchRouter!

   func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: LaunchOptions?) -> Bool {
      FirebaseApp.configure()
      mainLaunchRouter.openInitialModule(animated: false)
      return true
   }

}
```

Файл AppDelegatesAssemblies.swift:

```
import EasyDi

final class AppDelegatesAssembly: Assembly, ResolvingAssembly {

   lazy private var systemInfrastructure: SystemInfrastructureAssembly =
context.assembly()
   lazy private var storyboards: StoryboardsAssembly = context.assembly()

   private var mainLaunchRouter: LaunchRouter {
      return define(
         init: MainLaunchRouter(
            window: self.systemInfrastructure.mainWindow,
            storyboard: self.storyboards.drawer
```

```
        )
      )
    }

    func inject(into delegate: AppDelegate) {
      defineInjection(into: delegate) {
        $0.mainLaunchRouter = self.mainLaunchRouter
        return $0
      }
    }
}

extension AppDelegate: StoryboardInstantiatable {
    typealias InstantiationAssembly = AppDelegatesAssembly
}
```

Файл main.swift:

```
import UIKit

UIApplicationMain(
    CommandLine.argc,
    UnsafeMutableRawPointer(CommandLine.unsafeArgv)
      .bindMemory(
        to: UnsafeMutablePointer<Int8>.self,
        capacity: Int(CommandLine.argc)),
    NSStringFromClass(Application<AppDelegate>.self),
    NSStringFromClass(AppDelegate.self)
)
```

Файл FingerprintTrackingAlgorithm.swift:

```
final class FingerprintTrackingAlgorithm: TrackingAlgorithm {

    let beaconRegions: [BeaconRegion]

    init(beaconRegions: [BeaconRegion]) {
      self.beaconRegions = beaconRegions
    }
```

```
    func computePosition(with measures: [BeaconDescriptor: Int]) -> Point? {
        if beaconRegions.isEmpty { return nil }

        let mostStrongSignal = measures.reduce(0) { max($0, Double(abs($1.value))) }
        let currentFingerprint = Dictionary(uniqueKeysWithValues: measures.map { ($0, Double($1) / mostStrongSignal) })

        var minDiff = Double.greatestFiniteMagnitude
        var minDiffInd = 0
        for (ind, beaconRegion) in beaconRegions.enumerated() {
            for fingerprint in beaconRegion.fingerprints {
                let diff = compareFingerprints(currentFingerprint, fingerprint)
                if diff < minDiff {
                    minDiff = diff
                    minDiffInd = ind
                }
            }
        }

        let rect = beaconRegions[minDiffInd].rect

        return Point(
            x: rect.x + rect.width / 2.0,
            y: rect.y + rect.height / 2.0
        )
    }

    private func compareFingerprints(_ lhv: [BeaconDescriptor: Double], _ rhv: [BeaconDescriptor: Double]) -> Double {
        var descriptors = Set(lhv.keys)
        descriptors.formUnion(rhv.keys)

        return descriptors.reduce(0.0) { (res, descriptor) in
            switch (lhv[descriptor], rhv[descriptor]) {
                case let (.some(lhv), .some(rhv)): return res + abs(lhv - rhv)
                case (.some(let val), .none), (.none, .some(let val)): return res + val
                default: return res
```

```
        }
      }
   }

}
```

Файл TrilatirationAlgorithm.swift:

```
import Foundation

final class TrilatirationAlgorithm: TrackingAlgorithm {

   let beacons: [BeaconDescriptor: Beacon]

   init(beacons: [Beacon]) {
      self.beacons = .init(uniqueKeysWithValues: beacons.map({ ($0.descriptor,
$0) }))
   }

   func computePosition(with measures: [BeaconDescriptor: Int]) -> Point? {
      let three = measures
         .filter { self.beacons[$0.key] != nil }
         .sorted { $0.value < $1.value }
         .prefix(3)
         .map { (self.beacons[$0], $1) }
         .map { (beacon: $0, distance: computeDistance(basedOn: $1, with:
$0.txPower)) }
         .sorted { $0.beacon.position.x < $1.beacon.position.x } // TODO:
Reconsider this part.

      guard three.count == 3 else { return nil }

      let (first, second, third) = translateCoords(
         three[0].beacon.position,
         three[1].beacon.position,
         three[2].beacon.position
      )
```

```
        let x = (pow(three[0].distance, 2) - pow(three[1].distance, 2) + pow(second.x,
2)) / (second.x * 2)
        let y = (pow(three[0].distance, 2) - pow(three[1].distance, 2) + pow(third.x, 2)
+ pow(third.y, 2)) / (second.x * 2) * x
            - third.x / third.y

        return Point(
            x: three[0].beacon.position.x + x,
            y: three[0].beacon.position.y + y
        )
    }

    // TODO: Reconsider this function.
    // swiftlint:disable:next large_tuple
    private func translateCoords(_ first: Point, _ second: Point, _ third: Point) ->
(Point, Point, Point) {
        let firstCoords = Point(x: 0.0, y: 0.0)
        let secondCoords = Point(x: first.distance(to: second), y: 0.0)
        let thirdCoords = Point(x: first.distance(to: third), y: -first.distance(to: third))

        return (firstCoords, secondCoords, thirdCoords)
    }

    private func computeDistance(basedOn rssi: Int, with txPower: Int) -> Double {
        let ratio = Double(rssi) / Double(txPower)

        // swiftlint:disable:next identifier_name
        let (a, b, c) = //ratio < 1.0 ?
//                  (1.0, 10.0, 0.0) :
                  (0.89976, 7.7095, 0.111)
        return a * pow(ratio, b) + c
    }

}
```

Файл FirebaseBeaconStorage.swift:

```
import FirebaseDatabase
import RxFirebase
```

```
import RxSwift

private class FirebaseMapper {

    private static let keySeparator = "," as Character

    static func map(_ descriptor: BeaconDescriptor) -> String {
        return
"\(descriptor.minor)\(keySeparator)\(descriptor.major)\(keySeparator)\(descriptor.u
uid.uuidString)"
    }

    static func map(_ str: String) -> BeaconDescriptor {
        let substrings = str.split(separator: keySeparator, maxSplits: 3)
        guard
            let major = BeaconMajorValue(substrings[0]),
            let minor = BeaconMinorValue(substrings[1]),
            let uuid = UUID(uuidString: String(substrings[2]))
            else { fatalError("Failed mapping BeaconDescriptor from Firebase.") }

        return BeaconDescriptor(uuid: uuid, major: major, minor: minor)
    }

    static func map(_ beaconInfo: (txPower: Int, position: Point)) -> Any? {
        return [
            "txPower": beaconInfo.txPower,
            "positionX": beaconInfo.position.x,
            "positionY": beaconInfo.position.y
        ]
    }

    static func map(_ beaconRegion: BeaconRegion) -> Any? {
        let rect = beaconRegion.rect
        let measures = beaconRegion
            .fingerprints
            .enumerated()
            .map { (ind, arr) in
                (ind, arr.map { (map($0), $1) })
            }
```

```
    return [
        "id": beaconRegion.id.uuidString,
        "x": rect.x,
        "y": rect.y,
        "h": rect.height,
        "w": rect.width,
        "measures": Dictionary(uniqueKeysWithValues: measures)
    ]
}

// swiftlint:disable identifier_name
static func map(_ value: Any?) -> [BeaconRegion] {
    guard let entities = value as? [AnyObject] else { return [] }

    var arr: [BeaconRegion] = []
    for entity in entities {
        if entity is NSNull { continue }

        guard
            let dict = entity as? [String: Any],
            let idString = dict["id"] as? String,
            let id = UUID(uuidString: idString),
            let x = dict["x"] as? Double,
            let y = dict["y"] as? Double,
            let h = dict["h"] as? Double,
            let w = dict["w"] as? Double,
            let measures = dict["measures"] as? [[String: Double]]
            else { fatalError("Failed mapping Fingerprint from Firebase.") }

        let fingerprints = measures.map { (arr) in
            Dictionary(uniqueKeysWithValues: arr.map { (map($0), $1) })
        }

        arr.append(
            BeaconRegion(
                id: id,
                rect: Rect(x: x, y: y, width: w, height: h),
                fingerprints: fingerprints
```

```
            )
        )
    }

    return arr
}
// swiftlint:enable identifier_name

static func map(_ value: Any?) -> [Beacon] {
    guard let entities = value as? [String: AnyObject] else { return [] }

    var arr: [Beacon] = []
    for (key, entity) in entities {
        let descriptor = parseDescriptor(from: key)

        guard
            let dict = entity as? [String: Any],
            let txPower = dict["txPower"] as? Int,
            let positionX = dict["positionX"] as? Double,
            let positionY = dict["positionY"] as? Double
            else { fatalError("Failed mapping Beacon from Firebase.") }

        arr.append(
            Beacon(
                descriptor: descriptor,
                txPower: txPower,
                position: Point(x: positionX, y: positionY)
            )
        )
    }

    return arr
}

private static func parseDescriptor(from string: String) -> BeaconDescriptor {
    let substrings = string.split(separator: keySeparator, maxSplits: 3)
    guard
        let major = BeaconMajorValue(substrings[0]),
        let minor = BeaconMinorValue(substrings[1]),
```

```
            let uuid = UUID(uuidString: String(substrings[2]))
            else { fatalError("Failed mapping Beacon from Firebase.") }

        return BeaconDescriptor(uuid: uuid, major: major, minor: minor)
    }

}

protocol RemoteBeaconStorageGateway: class {
    func save(_ beacon: Beacon) -> Single<Void>
    func save(_ beaconRegion: BeaconRegion, into position: Int) -> Single<Void>
    func fetchBeacons() -> Observable<[Beacon]>
    func fetchFingerprints() -> Single<[BeaconRegion]>
    func removeBeacon(for descriptor: BeaconDescriptor)
}

final class FirebaseBeaconStorage: RemoteBeaconStorageGateway {

    private let ref: DatabaseReference

    init(ref: DatabaseReference) {
        self.ref = ref
    }

    func save(_ beacon: Beacon) -> Single<Void> {
        let beaconDescriptor = FirebaseMapper.map(beacon.descriptor)
        let beaconInfo = FirebaseMapper.map((txPower: beacon.txPower, position:
beacon.position))

        return ref.child("beacons")
            .child(beaconDescriptor)
            .rx
            .setValue(beaconInfo)
            .map { (_) in return }
            .asSingle()
    }

    func save(_ beaconRegion: BeaconRegion, into position: Int) -> Single<Void> {
        let fingerprintInfo = FirebaseMapper.map(beaconRegion)
```

```
        return ref.child("fingerprints")
            .child("\(position)")
            .rx
            .setValue(fingerprintInfo)
            .map { (_) in return }
            .asSingle()
    }

    func fetchBeacons() -> Observable<[Beacon]> {
        return ref.child("beacons")
            .rx
            .observeSingleEvent(.value)
            .map { FirebaseMapper.map($0.value) }
    }

    func fetchFingerprints() -> Single<[BeaconRegion]> {
        return ref.child("fingerprint")
            .rx
            .observeSingleEvent(.value)
            .map { FirebaseMapper.map($0.value) }
            .asSingle()
    }

    func removeBeacon(for descriptor: BeaconDescriptor) {
        fatalError("Deleting is not implemented.")
    }

}
```

Файл LocalBeaconStorage.swift:

```
protocol LocalBeaconStorageGateway {
    var beacons: [Beacon] { get set }
    var beaconRegions: [BeaconRegion] { get set }
}

final class LocalBeaconStorage: LocalBeaconStorageGateway {
```

```
    var beacons: [Beacon] = []
    var beaconRegions: [BeaconRegion] = []

}
```

Файл ScannerGateway.swift:

```
import RxSwift
import RxCocoa
import RxCoreLocation
import CoreLocation

typealias RangeEvent = [(descriptor: BeaconDescriptor, rssi: Int)]

protocol ScannerGateway: class {
    func startScan() -> Observable<CLRegionEvent>
    func startMeasuring() -> Observable<CLBeaconsEvent>

    func startRanging() -> Observable<RangeEvent>
    func calibrate(beaconWith descriptor: BeaconDescriptor) -> Observable<Int>
}

class CLBeaconsEventMapper {
    static func map(_ event: CLBeaconsEvent) -> RangeEvent {
        return event.beacons.map { (beacon) in
            let descriptor = BeaconDescriptor(
                uuid: beacon.proximityUUID,
                major: beacon.major.uint16Value,
                minor: beacon.minor.uint16Value
            )

            return (descriptor: descriptor, rssi: beacon.rssi)
        }
    }
}

final class ScannerGatewayImp: ScannerGateway {

    private let locationManager: CLLocationManager
```

```
    private let beaconRegion = CLBeaconRegion(
        proximityUUID: UUID(uuidString: "FFB18911-1E91-4806-AA73-
9B7E8BDCA701")!, // swiftlint:disable:this force_unwrapping
        major: 1,
        identifier: "com.smedialink.beacon"
    )

    init(locationManager: CLLocationManager) {
        self.locationManager = locationManager
        locationManager.requestAlwaysAuthorization()
    }

    func startScan() -> Observable<CLRegionEvent> {
        locationManager.requestAlwaysAuthorization()
        locationManager.startMonitoring(for: beaconRegion)

        return locationManager.rx
            .didReceiveRegion
            .asObservable()
    }

    func startMeasuring() -> Observable<CLBeaconsEvent> {
        locationManager.requestAlwaysAuthorization()
        locationManager.startRangingBeacons(in: beaconRegion)

        return locationManager.rx
            .didRangeBeacons
            .asObservable()
    }

    func startRanging() -> Observable<RangeEvent> {
        return locationManager.rx
            .didRangeBeacons
            .map(CLBeaconsEventMapper.map)
    }

    func calibrate(beaconWith descriptor: BeaconDescriptor) -> Observable<Int> {
        return locationManager.rx
            .didRangeBeacons
```

```
            .map {
                guard
                    let beacon = $0.beacons.first,
                    $0.beacons.count != 1
                    else { fatalError("Failed to calibrate beacon \(descriptor).") }
                return beacon.rssi
            }
    }

}
```

Файл BeaconCalibrationUsecase.swift:

```
import RxSwift

protocol BeaconCalibrationUsecase: class {
    func calibrate(beaconWith descriptor: BeaconDescriptor) -> Single<Int>
}

final class BeaconCalibrationUsecaseImp: BeaconCalibrationUsecase {

    private let scanner: ScannerGateway

    init(scanner: ScannerGateway) {
        self.scanner = scanner
    }

    func calibrate(beaconWith descriptor: BeaconDescriptor) -> Single<Int> {
        return Single<Int>.create(subscribe: { (consume) -> Disposable in
            var count: Int = 0
            var sum: Int = 0

            let disposable = self.scanner
                .calibrate(beaconWith: descriptor)
                .take(30)
                .subscribe(onNext: { (rssi) in
                    count += 1
                    sum += rssi
                }, onError: { (error) in
```

```
                consume(.error(error))
            }, onCompleted: {
                consume(.success(sum / count))
            })
        return Disposables.create { disposable.dispose() }
    })
  }

}
```

Файл BeaconManagementUsecase.swift:

```
import RxSwift

protocol BeaconManagementUsecase: class {
  func initCache() -> Observable<Void>
  func add(beacon: Beacon) -> Single<Void>
  func add(beaconRegion: BeaconRegion) -> Single<Void>
}

final class BeaconManagementUsecaseImp: BeaconManagementUsecase {

  private let remoteBeaconStorage: RemoteBeaconStorageGateway
  private var localBeaconStorage: LocalBeaconStorageGateway

  init(remoteBeaconStorage: RemoteBeaconStorageGateway,
localBeaconStorage: LocalBeaconStorageGateway) {
    self.remoteBeaconStorage = remoteBeaconStorage
    self.localBeaconStorage = localBeaconStorage
  }

  func initCache() -> Observable<Void> {
    return remoteBeaconStorage
      .fetchBeacons()
      .debug()
      .map { self.localBeaconStorage.beacons = $0 }
  }

  func add(beacon: Beacon) -> Single<Void> {
```

```
        return remoteBeaconStorage
            .save(beacon)
    }

    func add(beaconRegion: BeaconRegion) -> Single<Void> {
        let position = localBeaconStorage.beaconRegions.count
        return remoteBeaconStorage
            .save(beaconRegion, into: position)
    }

}
```

Файл PositioningUsecase.swift:

```
import RxSwift

enum TrackEvent {
    case new(position: Point)
    case outOfRange
}

protocol PositioningUsecase: class {
    var algorithmType: TrackingAlgorithmType { get set }
    func trackPosition() -> Observable<TrackEvent>
}

final class PositioningUsecaseImp: PositioningUsecase {

    private let scannerGateway: ScannerGateway
    private let localStorage: LocalBeaconStorageGateway

    private var algorithm: TrackingAlgorithm

    init(scannerGateway: ScannerGateway, localStorage:
LocalBeaconStorageGateway) {
        self.scannerGateway = scannerGateway
        self.localStorage = localStorage

        algorithmType = .fingerprint
```

```
        algorithm = FingerprintTrackingAlgorithm(beaconRegions:
localStorage.beaconRegions)
    }

    var algorithmType: TrackingAlgorithmType {
        didSet {
            switch algorithmType {
                case .fingerprint:
                    algorithm = FingerprintTrackingAlgorithm(beaconRegions:
localStorage.beaconRegions)
                case .trilatiration:
                    algorithm = TrilatirationAlgorithm(beacons: localStorage.beacons)
            }
        }
    }

    func trackPosition() -> Observable<TrackEvent> {
        return scannerGateway
            .startRanging()
            .map { [algorithm] (measures) in
                algorithm.computePosition(with: .init(uniqueKeysWithValues:
measures))
            }
            .map { (position) in
                guard let position = position else { return .outOfRange }
                return .new(position: position)
            }
    }

}
```

Файл ScannerUsecase.swift:

```
import RxSwift
import RxCoreLocation

protocol ScannerUsecase: class {
    func startScanning() -> Observable<CLRegionEvent>
    func startMeasuring() -> Observable<CLBeaconsEvent>
```

```
}

final class ScannerUsecaseImp: ScannerUsecase {

    private let scannerGateway: ScannerGateway

    init(scannerGateway: ScannerGateway) {
        self.scannerGateway = scannerGateway
    }

    func startScanning() -> Observable<CLRegionEvent> {
        return scannerGateway.startScan()
    }

    func startMeasuring() -> Observable<CLBeaconsEvent> {
        return scannerGateway.startMeasuring()
    }

}
```