Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Компьютерные системы автоматизации производства»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:
«Система трекинга местоположений»

Студент РК9-11М _____ В. С. Смирнов
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ А. В. Урусов
(Подпись, дата) (И.О.Фамилия)

2017 г.

Оглавление

ВВЕДЕНИЕ

В современном мире популярность Интернета позволяет собирать большое количество данных о различных пользователях и действиях, которые они совершают. Один из типов статистики, собираемой с мобильных устройств, подключенных к интернету, - это данные о местоположении пользователя. Сбор таких данных позволяет строить хронологию местонахождений человека, а также делиться ей с другими людьми.

Типичное применение таких систем заключается в наблюдении за маленькими детьми или пожилыми родственниками. Устройство, передающее местоположение, или трекер находится у наблюдаемого. Оно работает в автоматическом режиме и не требует никаких действий с его стороны. Любые перемещения записываются и передаются на сервер в виде геокоординат. При этом пока человек находится в допустимой зоне, эти данные представляют собой просто статистику перемещений, но как только он покидает допустимую или безопасную зону, наблюдатель немедленно получает об этом сообщение.

Такая ситуация является одной из самых распространенных в практике наблюдений за детьми. Например, ребенку запрещено подходить к оживленному шоссе, которое находится в пределах километра от его школы. Однако дети могут попадать под влияние плохих компаний – ребенок идет туда против своей воли, «с друзьями». Задача родителей - вовремя обнаруживать и пресекать такие случаи. Известны прецеденты, когда ребенок не был вовремя остановлен родителями, после чего «безобидная вылазка» на шоссе, заканчивалась ДТП со смертельным исходом. Любой родитель расскажет десятки таких историй и покажет десятки мест (стройки, глухие парки, заброшенные здания и т.д.) где присутствие его ребенка крайне нежелательно.

Аналогичная ситуация может произойти с пожилыми родственниками. Внезапный приступ, ухудшение самочувствия, падение на обледенелом тротуаре – такие ситуации происходят с пожилыми людьми каждый день.

1. Предпроектное исследование

1.1 Технологии для получения местоположения

Основной технологией, обладающей достаточной точностью является система глобального позиционирования или GPS, основанная на работе навигационных спутников. GPS состоит из трёх основных сегментов: космического, управляющего и пользовательского. Спутники GPS транслируют сигнал из космоса, и все приёмники GPS используют этот сигнал для вычисления своего положения в пространстве по трём координатам в режиме реального времени [1]. Основной принцип использования системы — определение местоположения путём измерения моментов времени приёма синхронизированного сигнала от навигационных спутников антенной потребителя. Общим недостатком использования любой радионавигационной системы является то, что при определённых условиях сигнал может не доходить до приёмника, или приходить со значительными искажениями или задержками. Например, практически невозможно определить своё точное местонахождение в глубине квартиры внутри железобетонного здания, в подвале или в тоннеле даже профессиональными геодезическими приёмниками. Так как рабочая частота GPS лежит в дециметровом диапазоне радиоволн, уровень сигнала от спутников может серьёзно снизиться под плотной листвой деревьев или из-за очень большой облачности. Нормальному приёму сигналов GPS могут повредить помехи от многих наземных радиоисточников, а также, в редких случаях, от магнитных бурь, либо преднамеренно создаваемых «глушилок».

При использовании обычного GPS существуют следующие проблемы:

- Время первого определения координат зависит от актуальности хранящихся в приемнике данных, которые передаются сигналом GPS, и от орбитальных данных. Чем дольше устройство не было активно, тем больше приёмнику нужно получить информации, прежде чем определение позиции будет возможным. В зависимости от того, насколько устарели данные, различают «холодный», «тёплый» и «горячий» старт GPS-приёмника.

- В условиях города видимость GPS-спутников часто сильно ограничена, а в закрытых помещениях и туннелях даже невозможна.
- Высокая потребляемая мощность GPS-приёмника.

Чтобы решить проблему холодного старта, была предложена технология, названная A-GPS [2]. Она позволяет значительно ускорить обновление устаревших данных. Для алгоритмов A-GPS необходим канал связи с удалённым сервером, который предоставляет информацию для приёмника. Для мобильных устройств этим каналом чаще всего является Интернет-соединение с помощью сотовой связи либо Wi-Fi. Обновления данных по сети позволяют GPS-приёмнику знать, на каких частотах ожидать сигналы спутников, прежде, чем он получит обновления из сигнала. Время до первого местоопределения уменьшается с 30 секунд до примерно 1 секунды, также повышается чувствительность A-GPS-приёмника, что позволяет определять местоположение при более слабом или нестабильном уровне сигналов.

Как альтернативные методы определения местоположения без GPS сигнала, используются технологии определения координат по базовым станциям радиотелефонной связи, а также по найденным Wi-Fi сетям.

Существует 3 основных способа определения местоположения по базовым станциям:

- TOA (Time of Arrival, Оценка времени прибытия сигнала) — основан на измерении и сравнении интервалов времени прохождения сигнала от мобильного телефона абонента до нескольких базовых станций. Требует модернизации оборудования сотовой сети. Точность может достигать 125 м. Базовые станции, принимающие сигнал мобильного телефона, должны быть оснащены LMU (Location Measurement Unit, блок определения местоположения). По разности времени поступления сигнала управляющим компьютером сети сотовой связи рассчитывается местоположение передатчика. Полученные координаты передаются соответствующему сетевому приложению (серверу услуги) или клиенту.

- OTD (Observed Time Difference, Наблюдаемая разность времени прибытия сигнала) — основан на измерении и сравнении интервалов времени прохождения сигналов от нескольких базовых станций до мобильного телефона абонента. Требует модернизации сетевого оборудования, однако такая модернизация значительно дешевле TOA. Управляющий контроллер мобильного телефона измеряет время прохождения сигнала от нескольких базовых станций, одна из которых оснащена блоком LMU. Для получения информации о своем местоположении абонент совершает звонок, при котором его телефон до установки речевого соединения посылает специальное сигнальное сообщение, MLC производит необходимые вычисления для расчета местоположения, после чего пакет данных с координатами местонахождения абонента пересылается на сотовый телефон.

Такой способ определения местоположения не обеспечивает большую точность, но зато не требует сигнала GPS, и менее энергозатратен [3].

Способ определения местоположения по Wi-Fi сетям основан на их названиях и мощности сигнала. Данные о сетях постепенно накапливаются и структурируются на серверах крупных поставщиков мобильного ПО, таких как Google и Apple. На основе полученной статистики мобильное устройство получает свое примерное местоположение.

1.2 Практическое применение технологий

В данном курсовом проекте предлагается реализовать трекер местоположения на основе современных технологий определения местоположения. В качестве устройства для определения геопозиции будет использоваться смартфон на ОС Android [4], обладающий всеми необходимыми датчиками. Данные со смартфона будут отправляться на сервер в Интернете, который будет предоставлять всю собранную статистику в Web-приложении, которое можно открыть через браузер.

2. Техническое задание

Разработанная система должна состоять из таких узлов как:

- Web-сервер;
- Android-приложение;
- Web-приложение.

Web-сервер является центральным узлом системы и позволяет:

- Зарегистрироваться или авторизоваться пользователю в системе;
- Отправить на сервер текущее местоположение;
- Просмотреть хронологию своих местоположений;
- Дать доступ другому пользователю к своей хронологии;
- Просматривать хронологию пользователей, которые дали доступ к просмотру;
- Добавлять разрешенные зоны местонахождения пользователя;
- Отправлять уведомления о выходе из разрешенной зоны, пользователям, имеющим соответствующий доступ.

Все вышеперечисленные требования должны быть выполнены как HTTP API. Необходимо соблюдать требования безопасности при разработке web-сервера, так как он обрабатывает чувствительные данные клиентов.

Android-приложение устанавливается на смартфон пользователя и позволяет:

- Авторизоваться пользователю в системе;
- Периодически отправлять местоположение пользователя на сервер;
- Включить или выключить периодическую отправку.
- Получить уведомление о том, что пользователь покинул разрешенную зону.

Важным моментом при разработке такого приложения является энергоэффективность такого приложения. Операция получения местоположения является чувствительной для аккумулятора устройства и должна быть оптимизирована.

Web-приложение представляет собой интерфейс в браузере, который позволяет:

- Зарегистрироваться или авторизоваться пользователю в системе;
- Просмотреть хронологию своих местоположений;
- Дать доступ другому пользователю к своей хронологии;
- Просматривать хронологию пользователей, которые дали доступ к просмотру;
- Добавлять разрешенные зоны местонахождения пользователя;

Интерфейс должен содержать визуальную карту для наглядного показа хронологии и разрешенных зон.

3. Концептуальный этап проектирования системы

3.1 Общая архитектура системы

Вся система разделена на 3 архитектурные части:

- Backend (web-сервер);
- Frontend (web-приложение);
- Android-приложение.

Frontend представляет собой интерфейс взаимодействия между пользователем и основной программно-аппаратной частью или backend. Интерфейс представляет собой Web страницы, сверстанные и запрограммированные с помощью технологий и языков HTML, CSS и JavaScript.

Чтобы отдавать код интерфейса клиентам используется веб-сервер Nginx [5]. Данный сервер отличается высокой производительностью при отдаче статических данных, а также надежностью работы.

Все данные от клиентов проксируются с помощью Nginx на backend системы. Backend представляет собой веб-сервер, написанный на языке C++ на базе фреймворка Poco [6]. Данный фреймворк позволяет быстро проектировать и реализовывать веб-сервера, не задумываясь о более низкоуровневой реализации HTTP протокола. Веб-сервер предоставляет HTTP API для управления данными пользователя.

В качестве базы данных для хранения данных о пользователях используется MongoDB [7]. Данная СУБД удобна своим документно-ориентированным способом хранения данных и позволяет быстро ими оперировать.

Чтобы хранить историю местоположений пользователей, будем использовать СУБД Clickhouse [8]. Эта система удобна для хранения и обработки аналитических данных в реальном времени, какими как раз и являются данные о местоположении. Также она обладает мощным функционалом сжатия данных, который необходим для длительного хранения статистики.

Android-приложение также подключается к основному веб-серверу через Nginx для авторизации и отправки логов о текущем местоположении.

Общую схему архитектуры можно увидеть на рис.1.

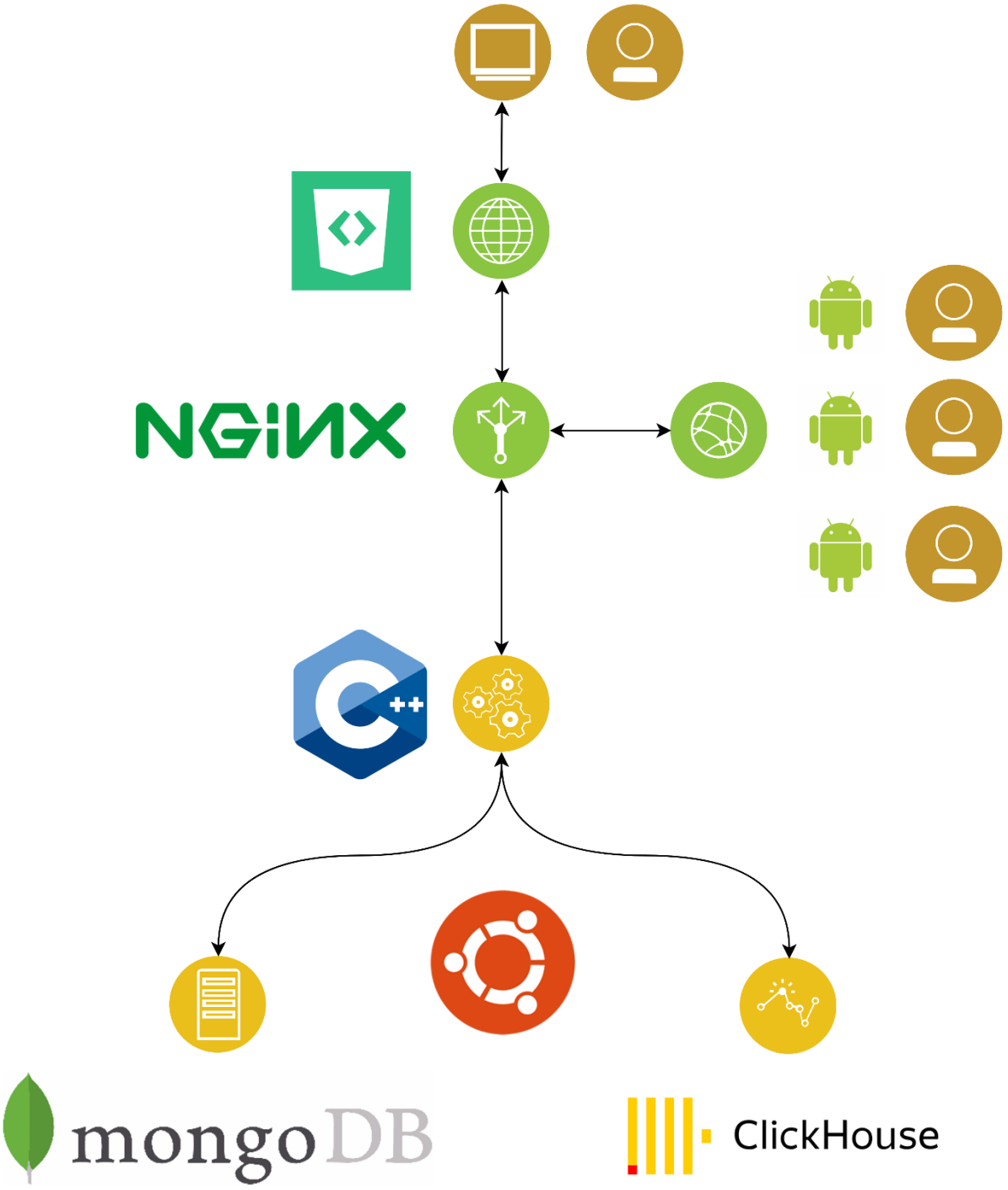


Рис. 1. Общая архитектура системы.

3.2 Web-сервер

Сервер должен позволять пользователю изменять данные в базе. Для этого необходимо предоставить API. Предлагается использовать архитектуру на основе REST [9]. RESTful API работает на основе HTTP запросов, которые в зависимости от параметров запроса несут ту или иную информацию.

Пользователь идентифицируется по уникальному email и может регистрироваться в системе с помощью него. Также при регистрации нужно указать пароль. После регистрации пользователь может зайти в систему с помощью ранее указанных email и пароля.

После авторизации пользователь может получить доступ к закрытому функционалу, такому как получение хронологии, добавление товарищей и разрешенных зон.

Получение хронологии происходит для указанного пользователя и даты. Пользователь может получить хронологию за любую дату, которая принадлежит ему или другим пользователям, которые добавили его в товарищи. В свою очередь, текущий пользователь может добавить других пользователей в товарищи, таким образом дав им доступ к своему местоположению.

Также пользователь может добавить зоны, в которых он может находиться. При покидании данной зоны, всем товарищам пользователя будет отправлено соответствующее оповещение.

При авторизации пользователя с Android-устройства, смартфон может отправлять на соответствующую конечную точку вызова информацию о текущем местоположении.

Далее будут представлены конечные точки вызова, которые позволяют выполнять требуемый функционал:

- /api/signup POST

```
{
    “email”: “lala@mail.ru”,
    “password”: “123”
}
```

Позволяет зарегистрироваться. Принимает JSON объект с полями email и password. Автоматически делает вход в систему в случае успеха.

- /api/session POST

```
{
     “email”: “lala@mail.ru”,
     “password”: “123”
}
```

Позволяет войти в систему. Принимает JSON объект с полями email и password.

- /api/session GET

Позволяет получить данные о текущем пользователе и его товарищах, совершившим вход в систему.

- /api/session DELETE

Позволяет выйти из системы.

- /api/log POST

```
{
     “latitude”: 38.4443,
     “longitude”: 55.4322,
     “accuracy”: 15.4,
     “speed”: 0,
     “time”: 1514010938
}
```

Позволяет записать новое местоположение в базу данных. Принимает JSON объект с геоданными.

- /api/track GET

```
/api/track?who=me&time=1514010938
/api/track?who=lala@mail.ru&time=1514010938
```

Позволяет получить хронологию местоположений пользователя. Принимает query параметры time и who, где time – это время для выбора дня показа, who – пользователь, данные которого нужно показать.

- /api/friends POST

```
{
     “email”: “lala@mail.ru”
}
```

Позволяет добавить товарища для текущего пользователя. Принимает JSON объект с полем email.

- /api/friends DELETE

```
{
     “email”: “lala@mail.ru”
}
```

Позволяет удалить пользователя из товарищей. Принимает JSON объект с полем email.

- /api/areas POST

```
{
     “name”: “Grove street”,
     “coordinates”: [
          {
               “latitude”: 38.432,
               “longitude”: 55.731
          },
          ...
     ]
}
```

Позволяет добавить разрешенную зону текущему пользователю. Принимает JSON объект с названием зоны и ее координатами в виде полигона.

- /api/areas DELETE

```
{
     “name”: “Grove street”
}
```

Позволяет удалить разрешенную зону у текущего пользователя. Принимает JSON объект с названием зоны.

3.3 Web-приложение

Приложение должно красиво и интуитивно понятно показывать информацию, полученную от web-сервера. Для этого необходимо спроектировать и реализовать интерфейс страниц приложения.

За основу предлагается взять платформу MaterializeCss [10], которая предоставляет интерфейс на базе Material Design от Google [11]. Данные гайдлайны тестируются уже долгое время и зарекомендовали себя как хороший стиль оформления.

Также необходимо визуализировать хронологию местоположений и разрешенные зоны пользователя. Для этого необходимо отображать географическую карту и объекты на ней.

Предлагается использовать Yandex.Maps API для этой цели [12]. Это бесплатная библиотека для отображения карты мира, а также разных объектов и меток на ней с удобным API для программирования в своих целях.

Интерфейс также должен позволять выбрать дату для хронологии, добавлять товарищей и разрешенные зоны и смотреть хронологию товарищей. Все это может быть реализовано средствами MaterializeCss.

3.4 Android-приложение

Мобильное приложение должно выполнять функцию трекера. Оно должно позволять входить в систему под определенным email`ом. После входа, оно должно позволять включить, либо выключить сбор данных. Если сбор данных включен, то приложение должно в фоне периодически запрашивать данные о местоположении и отправлять их на сервер. В случае отсутствия подключения к Интернету приложение должно сохранять данные в энергонезависимой памяти, а при наладке связи отправлять их на сервер.

Также приложение должно уметь принимать Push-уведомления от сервера в случае обнаружения выхода за разрешенную зону товарища данного пользователя. Для этого предполагается использовать систему Google Cloud Messaging, которая позволяет передавать такие уведомления от сервера к приложению [13].

Последний важный аспект работы приложения – это его энергоэффективность. Работа датчиков местоположения, особенно GPS, тратит много энергии. Поэтому необходимо регулировать запросы к этим датчикам, чтобы свести энергозатраты к минимуму. Данный вопрос более подробно рассмотрен в исследовательской части курсового проекта.

4. Технический этап проектирования системы

4.1 Web-сервер

Точкой инициализации программы является класс ***Server***, унаследованный от класса ***Poco::Util::ServerApplication***. Данный базовый класс несет функционал превращения обычного приложения в службу в системе Windows, либо в демон в Unix-подобных системах. Здесь происходит вся начальная инициализация и настройка сервера. Происходит подключение к базам данных и запуск сервера.

Для определения обработчиков конечных точек вызова используются классы, дочерние к классу ***Poco::Net::HTTPRequestHandler***. Данный класс умеет принимать запрос от клиента, обрабатывать его и отправлять обратно ответ.

Чтобы определить какой путь к ресурсу соответствует определенному обработчику, используется класс ***Factory***, унаследованный от класса ***Poco::Net::HTTPRequestHandlerFactory***. Данный класс представляет собой фабрику обработчиков. Он умеет определять, какой нужен обработчик для текущего запроса в зависимости от пути к ресурсу, указанному в запросе.

Для связи с базой данных MongoDB используется библиотека ***mongocxx***, а для связи с базой данных Clickhouse используется библиотека ***clickhouse-cpp***. Обе библиотеки позволяют установить пул соединений с базой, создавать, изменять и удалять данные в ней, а также выполнять различные запросы к хранимым данным.

Далее будут представлены классы обработчиков конечных точек вызова:

- ***SignupHandler***

  Позволяет зарегистрировать пользователя. Принимает от него регистрационные данные и сохраняет их в базу данных MongoDB.

- ***SigninHandler***

  Позволяет войти в систему. Принимает от пользователя его авторизационные данные, проверяет их, и в случае совпадения отправляет пользователю случайно сгенерированное 512-битное число в виде cookie.

- ***MeHandler***

  Позволяет авторизованному пользователю получить данные о себе. Отправляет данные о пользователе из MongoDB.

- ***LogoutHandler***

  Позволяет авторизованному пользователю выйти из системы. Сбрасывает авторизационную cookie.

- ***LogHandler***

  Позволяет авторизованному пользователю отправить свое текущее местоположение в систему. Принимает от пользователя геоданные и сохраняет их в базу данных ClickHouse.

- ***TrackHandler***

  Позволяет авторизованному пользователю получить хронологию местоположений для определенного пользователя в определенный день. Принимает от пользователя параметры запроса и отправляет выборку из базы данных ClickHouse.

- ***AddFriendHandler***

  Позволяет авторизованному пользователю добавить товарища. Принимает данные о товарище и добавляет его в список. Если пользователя не существует, выдает ошибку.

- ***DeleteFriendHandler***

  Позволяет авторизованному пользователю удалить товарища. Принимает данные о товарище и удаляет его из списка. Если пользователя не существует, выдает ошибку.

- ***AddAreaHandler***

  Позволяет авторизованному пользователю добавить разрешенную зону. Принимает данные о зоне и добавляет ее в список.

- ***DeleteAreaHandler***

Позволяет авторизованному пользователю удалить разрешенную зону. Принимает данные о зоне и удаляет ее из списка. Если зоны не существует, выдает ошибку.

Как только сервер получает очередное сообщение от пользователя с его текущим местоположением, он должен проанализировать, находится ли пользователь в пределах разрешенных зон. В случае обнаружения выхода за пределы зон, сервер должен отправить всем товарищам пользователя уведомление об этом.

Для этого используется технология Google Cloud Messaging, которая позволяет отправлять Push-уведомления мобильным приложениям. Для того чтобы отправить уведомление, нужно сделать POST-запрос на URL *https://gcm-http.googleapis.com/gcm/send*. Также в запросе обязательно нужно указать заголовок ***Authorization*** со значением ***key=AIzaSyA…***. Здесь key – это ключ авторизации к Google Cloud Messaging. Тело письма должно быть вида:

```
{
     “data”: { … }
     “to”:   { “token”: “0Y2_QravZwAE…”
}
```

Здесь token – это случайное значение, сгенерированное платформой Google Cloud Messaging, сопоставляемое с определенным приложением на определенном устройстве. Другими словами – это адрес, куда нужно доставить данные. Поле data – это JSON-объект, который полностью будет доставлен на мобильное приложение.

Таким образом, сервер должен иметь возможность делать HTTP запросы. Для этой цели удобно использовать популярную библиотеку libcurl для языка Си. Так как разработка ведется на языке C++, необходимо обеспечить удобный RAII интерфейс для работы с библиотекой. Для этого используется класс ***Curl***. Данный класс является статическим членом класса ***Server***, то есть создается в начале и уничтожается в конце работы программы. Этот класс умеет инициализировать ресурсы, необходимые для работы библиотеки, а также

освобождать их, когда время жизни объекта класса заканчивается. Также класс позволяет делать POST запрос на определенный URL с определенным телом запроса и необходимыми HTTP заголовками.

4.2 Web-приложение

Приложение в браузере представляет собой интерфейс, созданный с помощью языка гипертекстовой разметки HTML и каскадных таблиц стилей CSS, а также логику работы, запрограммированную на языке JavaScript.

В качестве готовых модулей CSS и JavaScript для создания интерфейса используется фреймворк MaterializeCss и Yandex.Maps API. Приложение состоит из 3 основных экранов:

- Экран входа
- Экран регистрации
- Основной экран

На экранах входа и регистрации, пользователь имеет возможность ввести свои данные и либо войти в систему, либо зарегистрироваться в ней.

На основном экране располагается карта, на которой пользователь может увидеть хронологию местоположений. Также на нем располагаются различные элементы управления, позволяющие, добавлять и удалять товарищей, добавлять и удалять разрешенные зоны, а также выбирать чью хронологию и когда нужно отобразить.

Бизнес-логика приложения содержит следующие функции:

- *login(signup = false)*

  Позволяет в зависимости от принятого аргумента либо зарегистрировать пользователя, либо попытаться войти в систему. В случае успеха переводит пользователя на основную страницу приложения.

- *logout()*

  Позволяет пользователю выйти из системы и переводит его на экран входа.

- *getTrack()*

  Позволяет запросить хронологию местоположений. В случае успеха отображает ее на карте.

- *getSession()*

  Позволяет запросить данные авторизованного пользователя. В случае успеха заполняет интерфейс данными, такими как email пользователя, его товарищи и разрешенные зоны.

- *addFriend()*

  Позволяет авторизованному пользователю добавить нового товарища.

- *deleteFriend(number)*

  Позволяет авторизованному пользователю удалить из товарищей пользователя под номером *number*.

- *addArea()*

  Позволяет пользователю нарисовать на карте новую разрешенную зону, ввести ее название и отправить на сервер.

- *deleteArea(number)*

  Позволяет пользователю удалить разрешенную зону под номером *number*.

- *changeTrackUser(user)*

  Позволяет выбрать, чью хронологию местоположений следует показывать: авторизованного пользователя или одного из его товарищей.

4.3 Android-приложение

Мобильное приложение состоит из двух основных экранов или активностей:

- Экран входа
- Основной экран

На экране входа пользователь может ввести свои данные и в случае успеха войти в нее. При этом приложение переходит на основной экран. На основном экране пользователь может включить или выключить отслеживание местоположения.

Также приложение содержит сервис для отправки текущего местоположения, работающий в фоне. Когда пользователь включает отслеживание, это означает запуск этого сервиса и, когда пользователь отключает отслеживание, сервис прекращает работу.

Сервис представлен классом ***TrackingService***, наследующимся от класса ***android.app.Service***. Он выполняет периодический опрос датчиков местоположения в смартфоне. Местоположение может быть получено либо от GPS, либо от базовых станций радиотелефонной связи, либо по окружающим устройство точкам доступа Wi-Fi. После получения местоположения, оно отправляет на сервер. В случае отсутствия доступа в Интернет, местоположение сохраняется в энергонезависимой памяти и отправляется при появлении в сети.

Сервис использует вспомогательный класс ***TrackingLocationProvider*** для получения местоположения. Этот класс позволяет сделать запрос к датчикам устройства и вернуть результат программе. Также он регулирует частоту местоположений относительно друг друга по расстоянию. Если человек находится на одном месте, класс не возвращает новое местоположение, так как это излишняя информация. Еще одна важная функция этого класса – это выбор периода опроса датчиков таким образом, чтобы затраты энергии устройства на определение местоположения были минимальны. Данная функция подробнее рассмотрена в исследовательской части курсового проекта.

Также приложение должно уметь принимать push-уведомления от сервера с помощью Google Cloud Messaging. Для этого приложение содержит еще один сервис, выполняющий функцию приёма сообщений. Сервис представлен классом ***TrackingGcmListenerService***, который наследуется от класса ***GcmListenerService***, и содержит унаследованный метод

public void onMessageReceived(String from, Bundle data)

Данный метод получает данные от сервера в объекте data класса ***Bundle***, а затем выводит в виде уведомления пользователю.

5. Рабочий этап проектирования системы

5.1 Web-сервер

В результате разработки был получен HTTP сервер, позволяющий с помощью RESTful API производить необходимые изменения в базах данных и, таким образом, обеспечивать необходимые функции системы.

5.2 Web-приложение

В результате разработки приложения в браузере был получен интерфейс, представленный на рисунках 2, 3, 4.

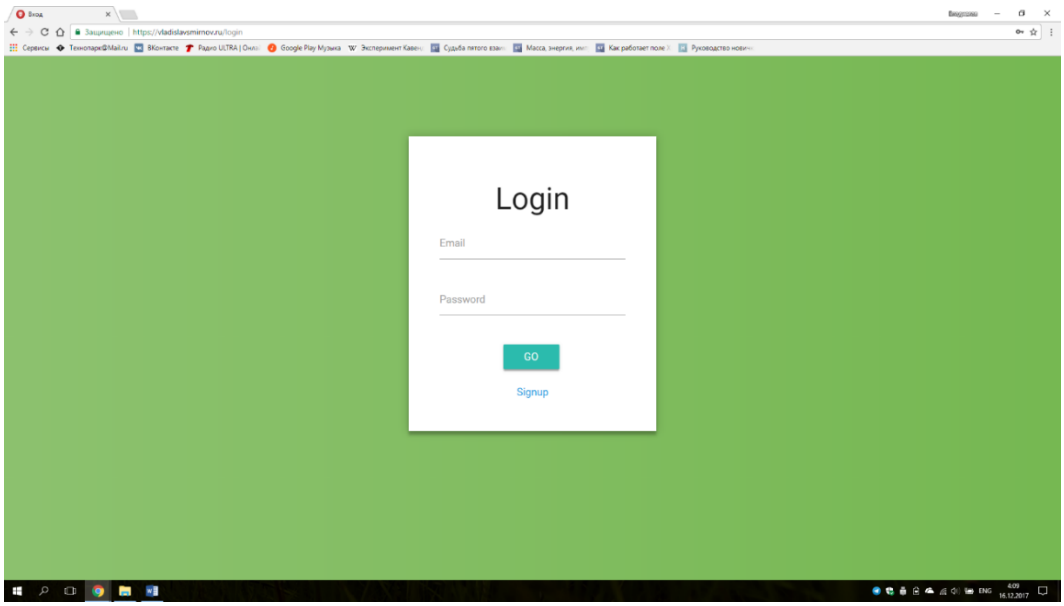


Рис. 2. Экран входа

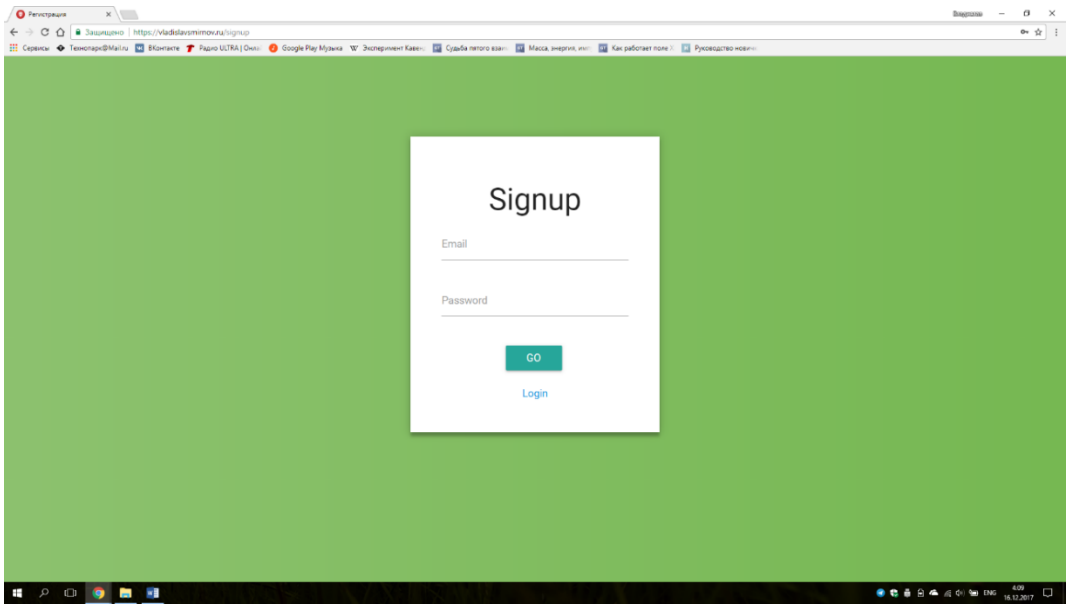


Рис. 3. Экран регистрации

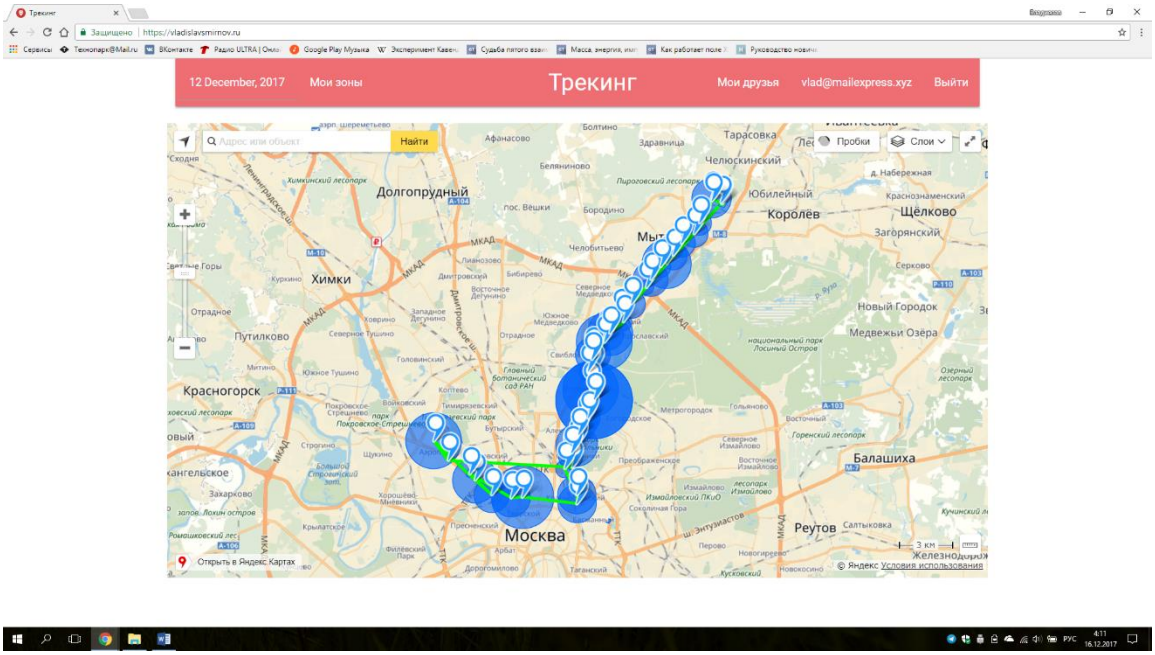


Рис. 3. Основной экран

5.3 Android-приложение

В результате разработки мобильного приложения, был реализован сервис получения местоположения и интерфейс, представленный на рисунках 4 и 5:

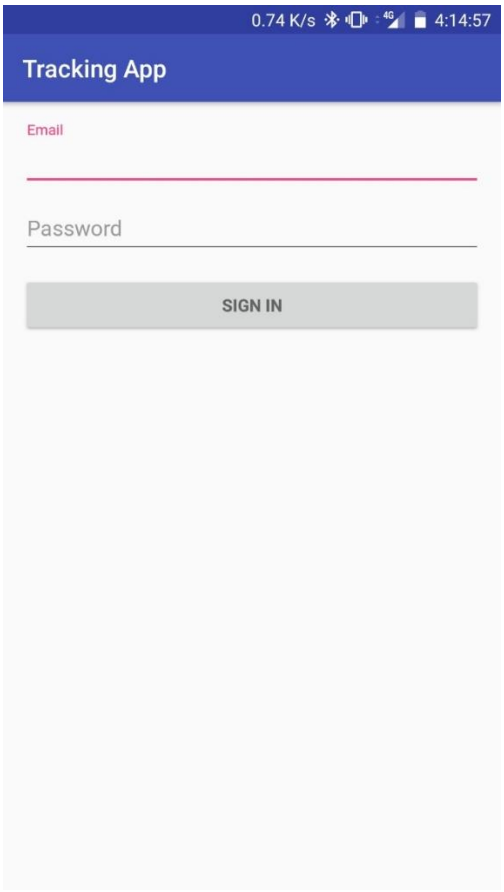


Рис. 4. Экран входа

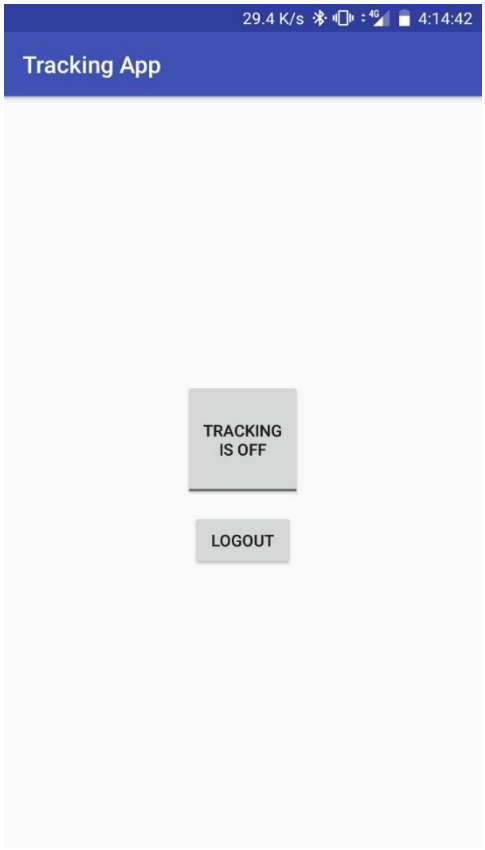


Рис. 5. Основной экран

6. Исследовательская часть

В процессе эксплуатации мобильного устройства его показатель автономности зависит не только от емкости установленного аккумулятора, но и от множества других параметров. В том числе и от программного обеспечения, которое используется на нем. Когда устройство не используется, операционная система Android переходит в состояние, называемое глубокий сон. Чем дольше устройство находится в этом состоянии, тем меньше садится батарея. Однако некоторые приложения, работающие в фоне, могут будить смартфон и выполнять различные функции, разряжая аккумулятор.

Мобильное приложение, разработанное в рамках данного курсового проекта, работает преимущественно в фоне, не требуя действий от пользователя устройства. Когда приложение будит устройство, оно начинает выполнять функцию запроса данных о текущем местоположении, что также является энергозатратной операцией. Таким образом, появилась потребность исследовать различные алгоритмы опроса датчиков мобильного устройства и выбрать тот, который сводит энергозатраты к минимуму.

Алгоритм сводится к выбору оптимального интервала между запросами к датчикам. В качестве первого варианта был выбран интервал в 10 секунд между запросами. Затем приложение было протестировано мобильном устройстве в течение дня. Результат показал, что приложение с такой частотой запросов, тратит значительную часть емкости аккумулятора. На рисунках 6, 7 можно увидеть статистику разряда батареи. Мобильное приложение на рисунках называется «Tracking App».

Рис. 6. Использование батареи приложением с интервалом 10с

Рис. 7. Детали статистики приложения с интервалом 10с

В следующем тесте интервал опроса был увеличен до 30 секунд. При этом приложение показало меньший, но все равно значительный расход аккумулятора (рис. 8, 9).

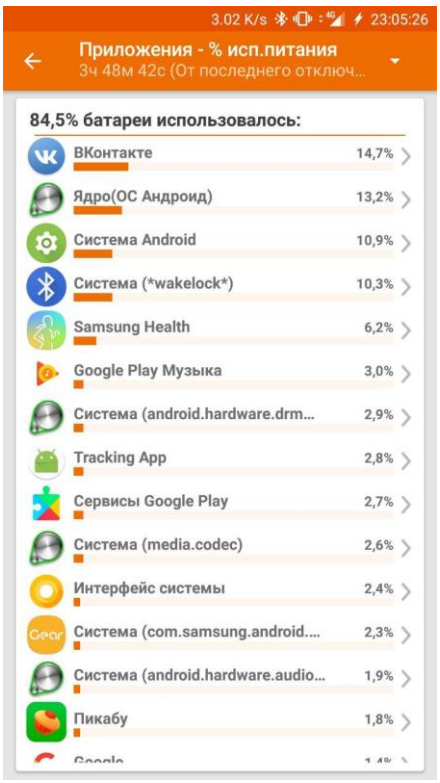


Рис. 8. Использование батареи приложением с интервалом 30с

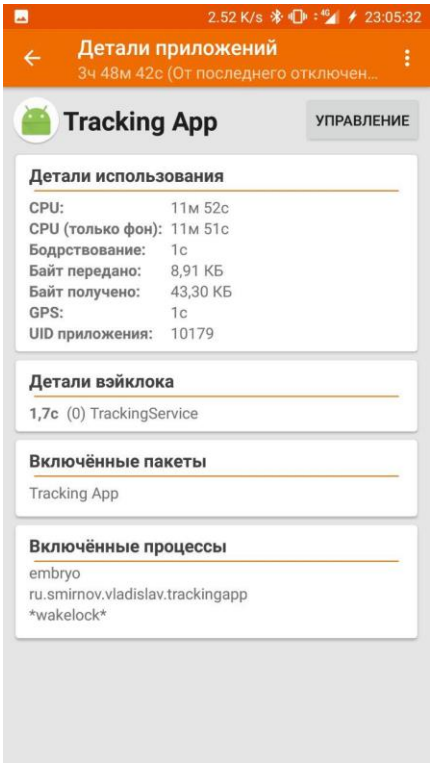


Рис. 9. Детали статистики приложения с интервалом 30с

Чтобы минимизировать потребление батареи был предложен способ менять интервал запросов в зависимости от изменения местоположения пользователя. Вначале интервал равняется минимальной исходной величине. Если разность местоположений меньше определенного порогового значения, то интервал постоянно увеличивается на определенную константу, до определенного максимального значения. Как только разность местоположений превышает заданную величину, интервал снова становится равным изначальному или минимальному.

В данном тесте в качестве минимального интервала было принято значение 30 секунд, в качестве максимального – 5 минут. Как шаг увеличения интервала было выбрано значение 30 секунд. Минимальная дистанция равна 50 метрам.

В результате тестирования в течение суток было обнаружено, что потребление батареи сильно уменьшилось (рис. 10)

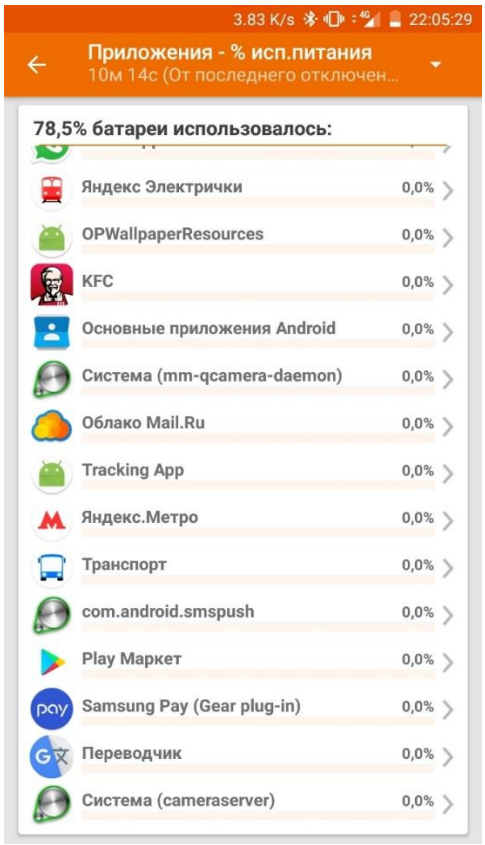


Рис. 10. Использование батареи приложением с регулируемым интервалом

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования было создана система, позволяющая вести хронологию местоположений пользователя, состоящая из трех основных подсистем:

- Web-сервер
- Web-приложение
- Android-приложение

Система позволяет пользователям сохранять историю своих местоположений на веб сервере с помощью мобильного приложения на платформе Android. А также просматривать свою хронологию или хронологию других пользователей, которые дали доступ на это. В качестве дополнительного функционала система позволяет пользователям создавать разрешенные зоны. Данные зоны позволяют товарищам пользователя своевременно получать информацию о том, что пользователь системы покинул данный регион.

Для оптимизации потребления энергии аккумулятора мобильным приложением было проведено исследование, заключающееся в поиске оптимального интервала между запросами к датчикам местоположения устройства. В результате тестирования разных подходов, был выбран динамический интервал, зависящий от того, перемещается ли в данный момент пользователь или стоит на месте. Таким образом была достигнута энергоэффективность мобильного приложения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GPS. [Электронный ресурс] // Wikipedia. URL: https://ru.wikipedia.org/wiki/GPS (дата обращения: 09.09.2017)
2. A-GPS. [Электронный ресурс] // Wikipedia. URL: https://ru.wikipedia.org/wiki/A-GPS (дата обращения: 09.09.2017)
3. Location-based service. [Электронный ресурс] // Wikipedia. URL: https://ru.wikipedia.org/wiki/Location-based_service (дата обращения: 10.09.2017)
4. Android. [Электронный ресурс] // URL: https://www.android.com/ (дата обращения: 10.09.2017)
5. Nginx. [Электронный ресурс] // URL: https://nginx.ru/ru/ (дата обращения: 10.09.2017)
6. POCO C++ Libraries. [Электронный ресурс] // URL: https://pocoproject.org/ (дата обращения: 10.09.2017)
7. mongoDB. [Электронный ресурс] // URL: https://www.mongodb.com/ (дата обращения: 10.09.2017)
8. ClickHouse. [Электронный ресурс] // URL: https://clickhouse.yandex/ (дата обращения: 10.09.2017)
9. REST. [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/REST (дата обращения: 10.09.2017)
10. MaterializeCss. [Электронный ресурс] // URL: http://materializecss.com/ (дата обращения: 10.09.2017)
11. Material Design. [Электронный ресурс] // URL: https://material.io/guidelines/ (дата обращения: 10.09.2017)
12. Yandex.Maps API. [Электронный ресурс] // URL: https://tech.yandex.ru/maps/ (дата обращения: 10.09.2017)
13. Google Cloud Messaging. [Электронный ресурс] // URL: https://developers.google.com/cloud-messaging/ (дата обращения: 10.09.2017)

ПРИЛОЖЕНИЕ А

Интерфейсы классов Web-сервера

add_area_handler.hpp

```
#ifndef PATH_TRACKING_ADD_AREA_HANDLER_HPP
#define PATH_TRACKING_ADD_AREA_HANDLER_HPP

#include "auth_handler.hpp"

class AddAreaHandler : public AuthHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    AddAreaHandler(mongocxx::pool &pool);
};

#endif //PATH_TRACKING_ADD_AREA_HANDLER_HPP
```

delete_area_handler.hpp

```
#ifndef PATH_TRACKING_DELETE_AREA_HANDLER_HPP
#define PATH_TRACKING_DELETE_AREA_HANDLER_HPP

#include "auth_handler.hpp"

class DeleteAreaHandler : public AuthHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    DeleteAreaHandler(mongocxx::pool &pool);
};


#endif //PATH_TRACKING_DELETE_AREA_HANDLER_HPP
```

add_friend_handler.hpp

```
#ifndef PATH_TRACKING_ADD_FRIEND_HANDLER_HPP
#define PATH_TRACKING_ADD_FRIEND_HANDLER_HPP

#include "auth_handler.hpp"

class AddFriendHandler final : public AuthHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    AddFriendHandler(mongocxx::pool& pool);
};


#endif //PATH_TRACKING_ADD_FRIEND_HANDLER_HPP
```

delete_friend_handler.hpp

```
#ifndef PATH_TRACKING_DELETE_FRIEND_HANDLER_HPP
#define PATH_TRACKING_DELETE_FRIEND_HANDLER_HPP

#include "auth_handler.hpp"

class DeleteFriendHandler : public AuthHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    DeleteFriendHandler(mongocxx::pool &pool);
};


#endif //PATH_TRACKING_DELETE_FRIEND_HANDLER_HPP
```

auth_handler.hpp

```
#ifndef PATH_TRACKING_AUTH_HANDLER_HPP
#define PATH_TRACKING_AUTH_HANDLER_HPP

#include <Poco/Net/HTTPRequestHandler.h>
#include <Poco/Net/HTTPServerRequest.h>
#include <Poco/Net/HTTPServerResponse.h>

#include <mongocxx/pool.hpp>

#include "user.hpp"

class AuthHandler : public Poco::Net::HTTPRequestHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response) override final;

protected:
    mongocxx::pool& pool;

    virtual void handleRequest(Poco::Net::HTTPServerRequest&
request, Poco::Net::HTTPServerResponse& response, const User&
user) = 0;

public:
    AuthHandler(mongocxx::pool& pool);
};


#endif //PATH_TRACKING_AUTH_HANDLER_HPP
```

curl.hpp

```
#ifndef PATH_TRACKING_CURL_HPP
#define PATH_TRACKING_CURL_HPP

#include <string>
#include <vector>

struct Curl {
    using Headers = std::vector<std::string>;

    bool sendPostMessage(const std::string& url, const
std::string& message, const Headers& headers = {}) const;

    Curl();
    ~Curl();
};

#endif //PATH_TRACKING_CURL_HPP
```

factory.hpp

```
#ifndef PATH_TRACKING_FACTORY_H
#define PATH_TRACKING_FACTORY_H

#include <Poco/Net/HTTPRequestHandlerFactory.h>

#include <regex>

class Factory final : public
Poco::Net::HTTPRequestHandlerFactory {
    typedef std::function<Poco::Net::HTTPRequestHandler*()>
handlerFactory;

    struct Endpoint {
        handlerFactory handler;
        std::string method;
    };

    std::vector<std::pair<std::regex, Endpoint>> routes;

    Poco::Net::HTTPRequestHandler* createRequestHandler(const
Poco::Net::HTTPServerRequest& request) override;

public:
    typedef Poco::SharedPtr<Factory> Ptr;

    template <class T>
    static auto wrap(auto&&... args) {
        return [=] { return new T(args...); };
    };

    void route(const std::string& location, const
handlerFactory& handlerFactory, const std::string& method);
};

#endif //PATH_TRACKING_FACTORY_H
```

format.hpp

```
#ifndef PATH_TRACKING_FORMAT_H
#define PATH_TRACKING_FORMAT_H

#include <sstream>
#include <vector>

std::string sprint(const char* format, const
std::vector<std::string>& args);

namespace std
{

inline std::string to_string(const bool x) { return x ? "true"
: "false"; }
inline std::string to_string(const char* x) { return x; }
inline std::string to_string(const std::string& x) { return x;
}

inline std::string to_string(const auto& x)
{
    std::ostringstream ss;
    ss << x;
    return ss.str();
}

}

inline std::string format(const char* s, auto&&... args)
{
    return sprint(s, { std::to_string(args)... });
}

#endif //PATH_TRACKING_FORMAT_H
```

log_handler.hpp

```
#ifndef PATH_TRACKING_LOG_HANDLER_H
#define PATH_TRACKING_LOG_HANDLER_H

#include "auth_handler.hpp"

#include <queue>
#include <mutex>

struct Log {
    std::string json;
    User user;
    std::time_t time;

    Log(const std::string& json, const User& user, std::time_t
time);
};

typedef std::vector<Log> Logs;

class LogHandler final : public AuthHandler {

    Logs& logs;
    std::mutex& logsMutex;

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    LogHandler(mongocxx::pool& pool, Logs& logs, std::mutex&
logsMutex);
};

#endif //PATH_TRACKING_LOG_HANDLER_H
```

logout_handler.hpp

```
#ifndef PATH_TRACKING_LOGOUT_HANDLER_HPP
#define PATH_TRACKING_LOGOUT_HANDLER_HPP

#include "auth_handler.hpp"

class LogoutHandler final : public AuthHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    LogoutHandler(mongocxx::pool& pool);
};


#endif //PATH_TRACKING_LOGOUT_HANDLER_HPP
```

me_handler.hpp

```
#ifndef PATH_TRACKING_ME_HANDLER_H
#define PATH_TRACKING_ME_HANDLER_H

#include "auth_handler.hpp"

class MeHandler final : public AuthHandler {

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;

public:
    MeHandler(mongocxx::pool& pool);
};


#endif //PATH_TRACKING_ME_HANDLER_H
```

server.hpp

```
#ifndef PATH_TRACKING_SERVER_H
#define PATH_TRACKING_SERVER_H

#include "factory.hpp"

#include "log_handler.hpp"
#include "track_handler.hpp"
#include "signup_handler.hpp"
#include "signin_handler.hpp"
#include "me_handler.hpp"
#include "logout_handler.hpp"
#include "add_friend_handler.hpp"
#include "delete_friend_handler.hpp"
#include "add_area_handler.hpp"
#include "delete_area_handler.hpp"
#include "curl.hpp"

#include <Poco/Util/ServerApplication.h>

#include <atomic>
#include <condition_variable>

class Server final : public Poco::Util::ServerApplication {

    static Curl curl;

    Logs logs;
    std::mutex logsMutex;
    std::atomic_bool isRunning;
    std::condition_variable workerCv;

    void logWorker(clickhouse::Client& client);

    int main(const std::vector<std::string>& args) override;

};


#endif //PATH_TRACKING_SERVER_H
```

signin_handler.hpp

```
#ifndef PATH_TRACKING_LOGIN_HANDLER_H
#define PATH_TRACKING_LOGIN_HANDLER_H

#include <Poco/Net/HTTPRequestHandler.h>
#include <Poco/Net/HTTPServerRequest.h>
#include <Poco/Net/HTTPServerResponse.h>

#include <mongocxx/pool.hpp>

class SigninHandler final : public
Poco::Net::HTTPRequestHandler {

    mongocxx::pool& pool;

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response) override;

public:
    SigninHandler(mongocxx::pool& pool);
};


#endif //PATH_TRACKING_LOGIN_HANDLER_H
```

signup_handler.hpp

```
#ifndef PATH_TRACKING_SIGNUP_HANLDER_H
#define PATH_TRACKING_SIGNUP_HANLDER_H

#include <Poco/Net/HTTPRequestHandler.h>
#include <Poco/Net/HTTPServerRequest.h>
#include <Poco/Net/HTTPServerResponse.h>

#include <mongocxx/pool.hpp>

class SignupHandler final : public
Poco::Net::HTTPRequestHandler {

    mongocxx::pool& pool;

    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response) override;

public:
    SignupHandler(mongocxx::pool& pool);
};


#endif //PATH_TRACKING_SIGNUP_HANLDER_H
```

track_handler.hpp

```
#ifndef PATH_TRACKING_TRACK_HANDLER_H

#define PATH_TRACKING_TRACK_HANDLER_H


#include "auth_handler.hpp"


#include <clickhouse/client.h>


class TrackHandler final : public AuthHandler {


    static thread_local clickhouse::Client client;


    void handleRequest(Poco::Net::HTTPServerRequest& request,
Poco::Net::HTTPServerResponse& response, const User& user)
override;


public:

    TrackHandler(mongocxx::pool& pool);

};


#endif //PATH_TRACKING_TRACK_HANDLER_H
```

user.hpp

```
#ifndef PATH_TRACKING_USER_HPP
#define PATH_TRACKING_USER_HPP

#include "json.hpp"

struct Coordinate {
    double latitude;
    double longitude;

    Coordinate() = default;
    Coordinate(double latitude, double longitude);
};

struct Area {
    std::string id;
    std::string name;
    std::vector<Coordinate> coordinates;

    Area(const std::string& id, const std::string& name, const
std::vector<Coordinate>& coordinates);
};

struct User {
    std::string id;
    std::string email;
```

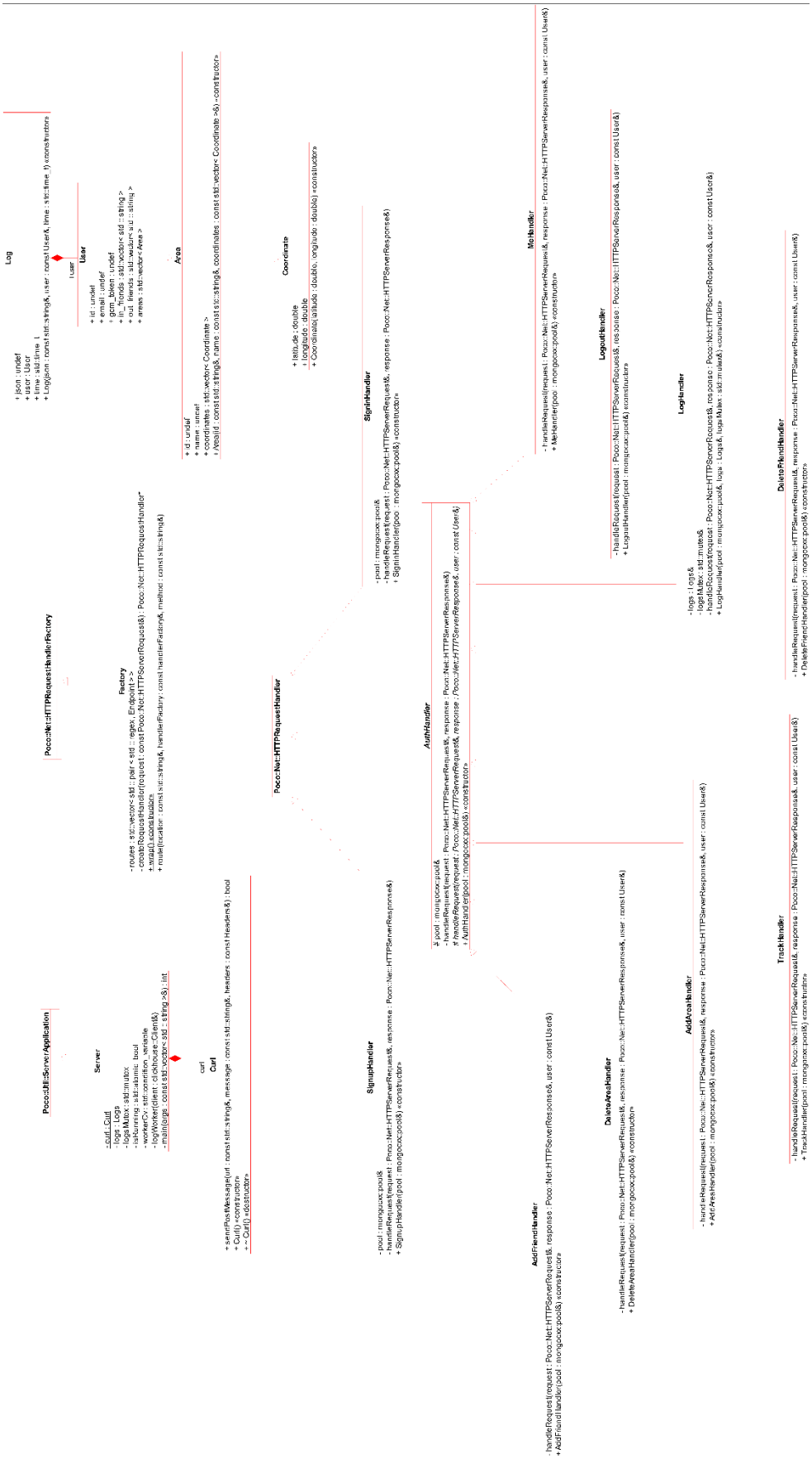
```
    std::string gcm_token;
    std::vector<std::string> in_friends;
    std::vector<std::string> out_friends;
    std::vector<Area> areas;
};


void from_json(const nlohmann::json& j, Coordinate&
coordinate);
void to_json(nlohmann::json& j, const User& user);


#endif //PATH_TRACKING_USER_HPP
```

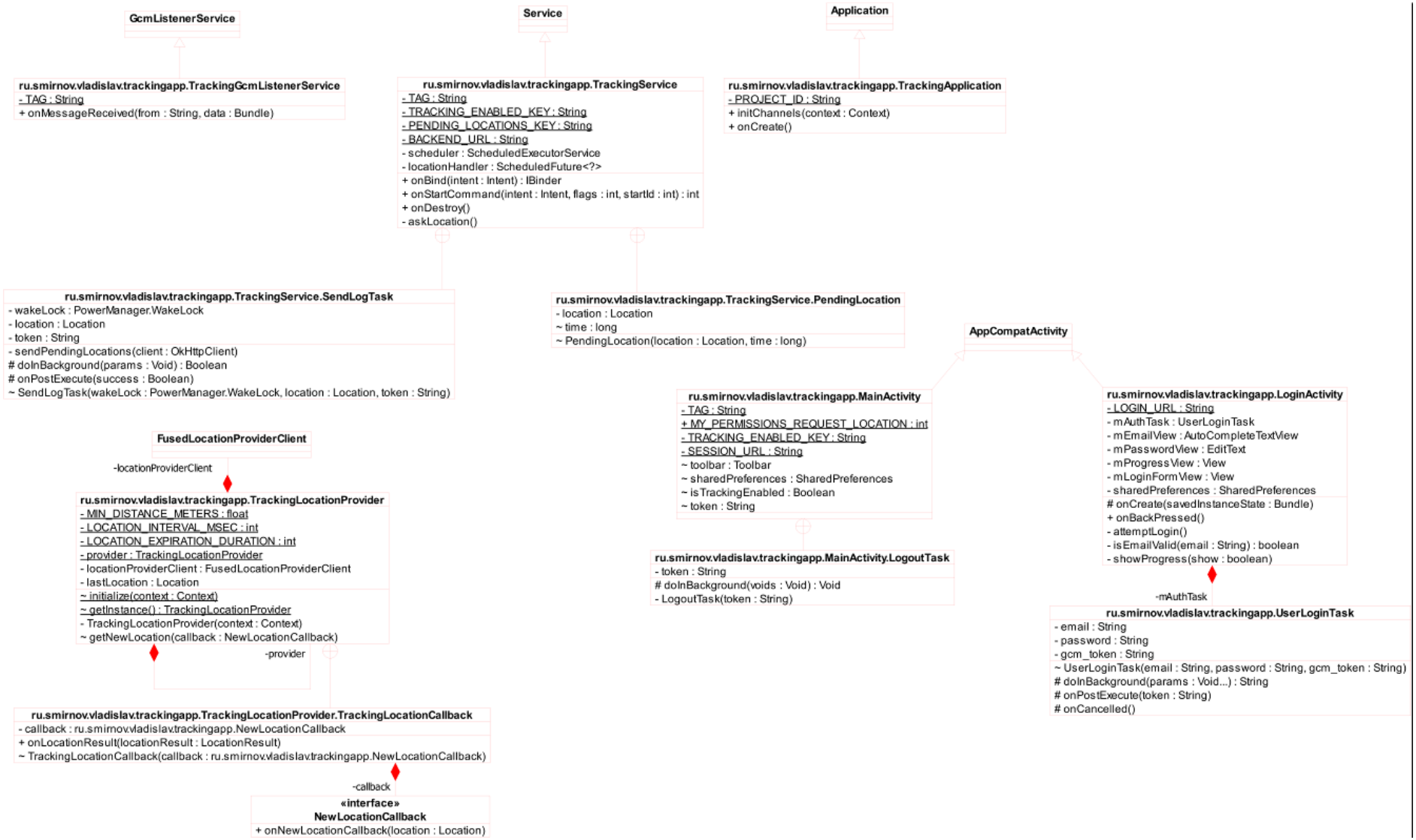
utils.hpp

```
#ifndef PATH_TRACKING_UTILS_HPP

#define PATH_TRACKING_UTILS_HPP


#include <istream>

#include <mongocxx/collection.hpp>

#include <mongocxx/pool.hpp>

#include <Poco/Net/HTTPServerResponse.h>


#include "user.hpp"

#include "curl.hpp"


std::string readAll(std::istream& stream);

mongocxx::collection usersCollection(mongocxx::pool::entry&
client);

void sendBadRequest(Poco::Net::HTTPServerResponse& response);

void sendUnauthorized(Poco::Net::HTTPServerResponse&
response);

std::string generateToken();

std::string sha256(const std::string& value);

bool isIntersects(const Area& area, float latitude, float
longitude, float accuracy);


void sendPush(const Curl& curl, const std::string& userName,
const std::string& areaName, const std::string& token);


#endif //PATH_TRACKING_UTILS_HPP
```

ПРИЛОЖЕНИЕ Б

UML диаграммы Web-сервера и Android-клиента




GcmListenerService
Service
Application
ru.smirnov.vladislav.trackingapp.TrackingGcmListenerService
- TAG : String
+ onMessageReceived(from : String, data : Bundle)
ru.smirnov.vladislav.trackingapp.TrackingService
- TAG : String
- TRACKING_ENABLED_KEY : String
- PENDING_LOCATIONS_KEY : String
- BACKEND_URL : String
- scheduler : ScheduledExecutorService
- locationHandler : ScheduledFuture<?>
+ onBind(intent : Intent) : IBinder
+ onStartCommand(intent : Intent, flags : int, startId : int) : int
+ onDestroy()
- askLocation()
ru.smirnov.vladislav.trackingapp.TrackingApplication
- PROJECT_ID : String
+ initChannels(context : Context)
+ onCreate()
ru.smirnov.vladislav.trackingapp.TrackingService.SendLogTask
- wakeLock : PowerManager.WakeLock
- location : Location
- token : String
- sendPendingLocations(client : OkHttpClient)
doInBackground(params : Void) : Boolean
onPostExecute(success : Boolean)
~ SendLogTask(wakeLock : PowerManager.WakeLock, location : Location, token : String)
ru.smirnov.vladislav.trackingapp.TrackingService.PendingLocation
- location : Location
~ time : long
~ PendingLocation(location : Location, time : long)
AppCompatActivity
ru.smirnov.vladislav.trackingapp.MainActivity
- TAG : String
+ MY_PERMISSIONS_REQUEST_LOCATION : int
- TRACKING_ENABLED_KEY : String
- SESSION_URL : String
~ toolbar : Toolbar
~ sharedPreferences : SharedPreferences
~ isTrackingEnabled : Boolean
~ token : String
ru.smirnov.vladislav.trackingapp.LoginActivity
- LOGIN_URL : String
- mAuthTask : UserLoginTask
- mEmailView : AutoCompleteTextView
- mPasswordView : EditText
- mProgressView : View
- mLoginFormView : View
- sharedPreferences : SharedPreferences
onCreate(savedInstanceState : Bundle)
+ onBackPressed()
- attemptLogin()
- isEmailValid(email : String) : boolean
- showProgress(show : boolean)
FusedLocationProviderClient
-locationProviderClient
ru.smirnov.vladislav.trackingapp.TrackingLocationProvider
- MIN_DISTANCE_METERS : float
- LOCATION_INTERVAL_MSEC : int
- LOCATION_EXPIRATION_DURATION : int
- provider : TrackingLocationProvider
- locationProviderClient : FusedLocationProviderClient
- lastLocation : Location
~ initialize(context : Context)
~ getInstance() : TrackingLocationProvider
- TrackingLocationProvider(context : Context)
~ getNewLocation(callback : NewLocationCallback)
-provider
ru.smirnov.vladislav.trackingapp.MainActivity.LogoutTask
- token : String
doInBackground(voids : Void) : Void
- LogoutTask(token : String)
-mAuthTask
ru.smirnov.vladislav.trackingapp.UserLoginTask
- email : String
- password : String
- gcm_token : String
~ UserLoginTask(email : String, password : String, gcm_token : String)
# doInBackground(params : Void...) : String
# onPostExecute(token : String)
# onCancelled()
ru.smirnov.vladislav.trackingapp.TrackingLocationProvider.TrackingLocationCallback
- callback : ru.smirnov.vladislav.trackingapp.NewLocationCallback
+ onLocationResult(locationResult : LocationResult)
~ TrackingLocationCallback(callback : ru.smirnov.vladislav.trackingapp.NewLocationCallback)
-callback
«interface»
NewLocationCallback
+ onNewLocationCallback(location : Location)