Performance analysis framework for base station placement using IEEE 802.11

Introduction and Motivation

The successful placement of an aerial base station requires knowledge of the locations of the user equipment. This data must be collected continuously to provide up-to-date information for a placement algorithm.

To prepare the experimental evaluation of the placement of the aerial base stations, the framework for the performance analysis needs to be developed.

Requirement Overview

Its main objective is to receive the GPS location data from several Android-based telephones that send this data via UDP sockets.

The site data collector must store the received data in an internal database and then make this data available to the placement algorithm. Another part of this framework is the performance evaluation module, which sends the data over the network and records the amount of data sent.

It is required to have a system that can receive from clients information about their GPS coordinates and Wi-Fi signal quality. Based on this information, an optimization algorithm must be applied, which predicts how to change the position of the Wi-Fi station to increase capacity concerning all connected clients.

Architecture Constraints

Table 1: Architecture Constraints

Number	Description
1	For the phone there should be used native technologies to access the sensors on-board
2	To validate the framework, use Wi-Fi radio connection
3	There should be a GUI interface to visualize the data
4	Python programming language should be used
5	Phones should send messages with GPS coordinates in UDP diagrams

Context and Scope

The **GPS_Tracker** set of projects intended to be used for the optimization of mobile users' signals. Optimization aims to find the best positions for the base stations represented by the UAVs.

Business context

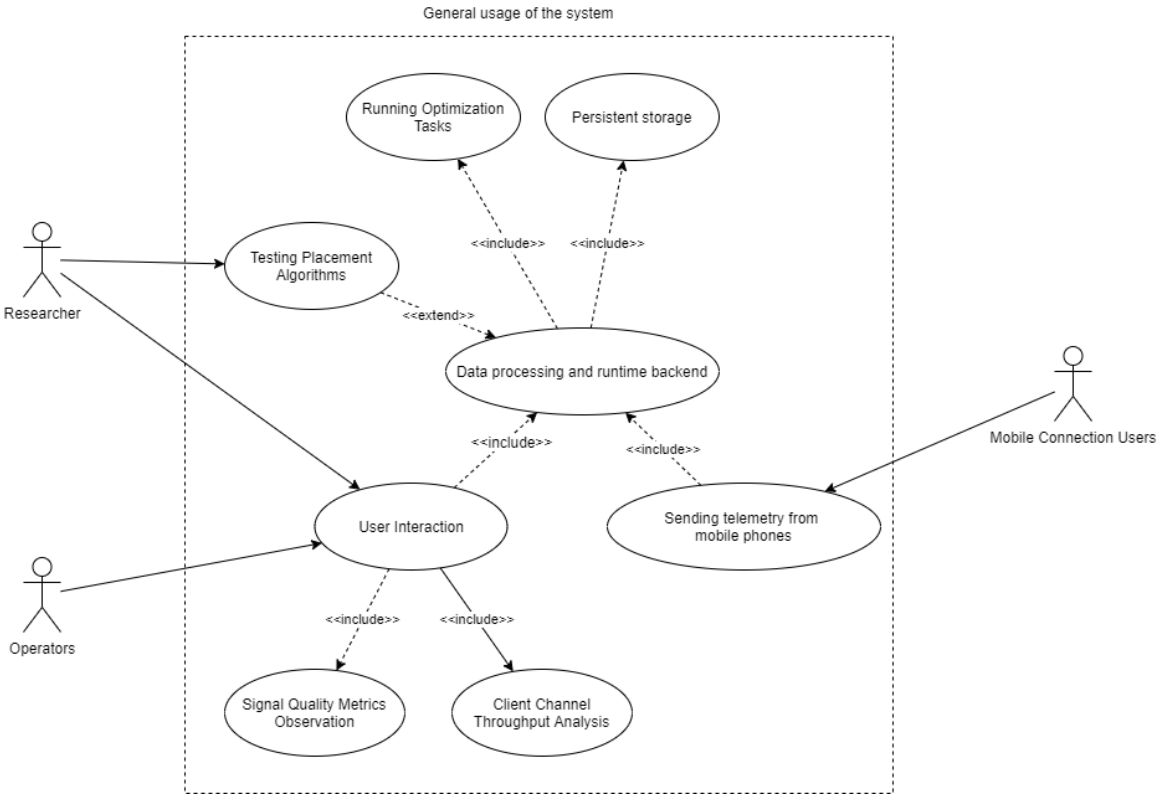

Figure 1: Business context representation

There are three types of actors:

1. Mobile Connection Users - the volunteers that allowed to install the special software GPS_Android on their mobile phones. They are connected to the access points provided by the network which efficiency we seek to increase.
2. Operators - the owners of the mobile network who are interested in the increased network throughput. They want to optimize either technical characteristics of the radio technologies or optimize the layout of the access points.
3. Researchers - scientific guys who want to test the layout optimization algorithms on the telemetry data from mobile connection users. They also can consult Operators what how to exactly treat the information provided by **GPS_Tracker**.

Also, that is worth describing the activities performing in the system. There are three important use-cases:

1. User interaction - There is a web application that allows users to interact with the backend to fetch and observe information stored. Also, there is a purpose to filter data and run optimization tasks.

2. Sending telemetry data from mobile phones - each mobile connection user has a special program installed on the phones. That is an important part of the software because it provides the real info on the radio network quality attributes.
3. Data processing and runtime backend - There is a set of the program running in the backend that performs a lot of processing operations to serve the user requests. No direct interaction from the actors required.

Also, there are other activities included in or to extend these three use-cases:

Table 2: Main use-cases description

Use-case	Description
Signal Quality Metrics Observation	Each telemetry message from the mobile connection users includes information about the location and current wireless connection RSS. That information is shown via different figures available in the UI.
Client Channel Throughput Analysis	Each client performs uplink and downlink throughput evaluation. These throughput evaluation drawn are to be observed and analyzed in **GPS_Frontend*.
Testing Placement Algorithms	There is a simple and extensible approach on how to add additional optimization algorithms to test. There is a unified interface to access telemetry data.
Running Optimization Tasks	**GPS_Tracker** can run several optimizations in parallel. The researches can compare the results of different algorithms.
Persistent storage	The telemetry from the users as well as optimization task results are saved persistently in well-known JSON format. Further, the data can be imported in other analytic tools.

Technical context

The following diagrams show the input/output interfaces for the main components.

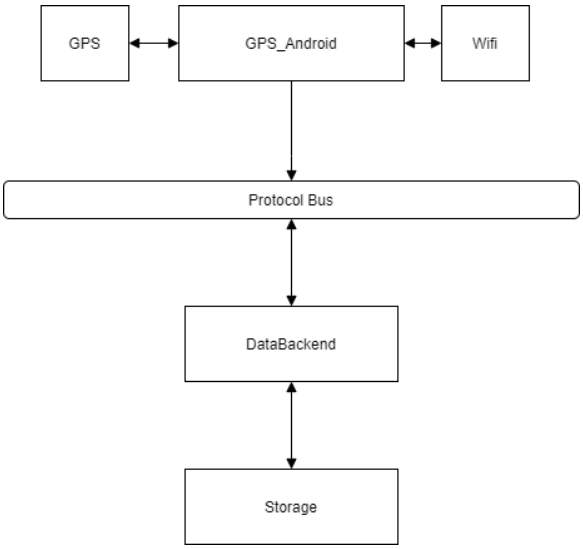

Figure 2: Abstract Architecture representation

Solution Strategy

Conceptual Design

Conceptually, the GPS_Tracker framework consists of several abstract components

Table 3: Conceptual Design components

Component	Description
Analyzer	An special entity that performs ETL operations on the data provided by ClientApp.
ClientApp	An entity installed on the user's phones that send information from the sensors.
MessageBroker	Protocol that the data from sensors is sent over.
DataBroker	A sensor data receiver part. The terminating side of MessageBroker.
DataBackend	A component that provides data access methods for the end clients.
DataVisualizer	An entity the user interacts with.
Storage	A program that stores sensor data reliably and has a well-defined access interface.

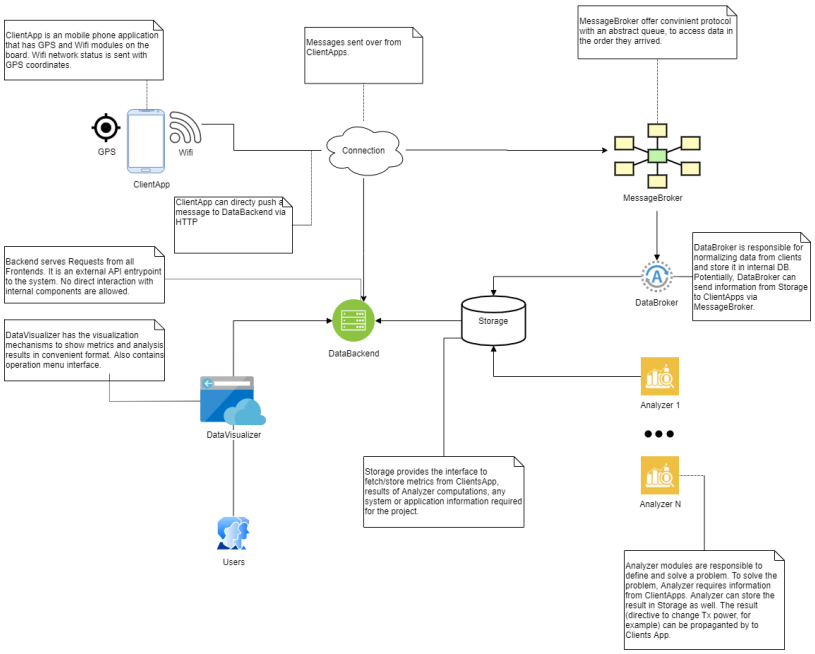

Figure 3: Conceptual Design Diagram

Solution Approaches

Performance

Goal

The messages receiving an operation performing must be accomplished as fast as possible.

Solution

To accomplish that, a protocol with very low latency but still reliable must be used. That must be a TCP/IP-based application protocol. There are two possible choices:

- TCP-based protocols for a reliable connection, but may have higher latency.
- UDP-based protocols for fast transmission without built-in reliable quality, but still has some of acknowledging mechanisms on the application level.

Moreover, all components must be aware of to be written with efficient programming techniques and components.

Reliability

Goal

The communication sides must have a way to check if the message received and interpreted correctly.

Solution

For that, there must be checks in the end-to-end protocol, as well as checks in the backend that received information stored.

Scalability

Goal

There must be an opportunity to easily increase the performance of the framework if required.

Solution

Publish/subscriber architecture pattern suits well the scalability requirement. Each component can be changed separately by different component regarding the proper interface implementation done.

Usability

Goal

The human-to-machine communication must have a user-friendly interface. There should be no problem with using the framework.

Solution

For better understanding, a proper documentation section is available for the user. The UI will be built using modern web technologies. That would give good user-experience.

Maintainability

Goal

Since the framework is quite complex, the additional complexity definitely will harm possible production installation.

Solution

To increase maintainability automation tools for development, deployment and maintenance used. The framework configured to provide easy-to-use instruments for the administrators.

Building Blocks

Level 1 Main Scope and Context

The following diagram represents the main component in the framework divided by four subsystems:

- Client Subsystem
- Processing Subsystem
- Storage Subsystem
- UI Subsystem

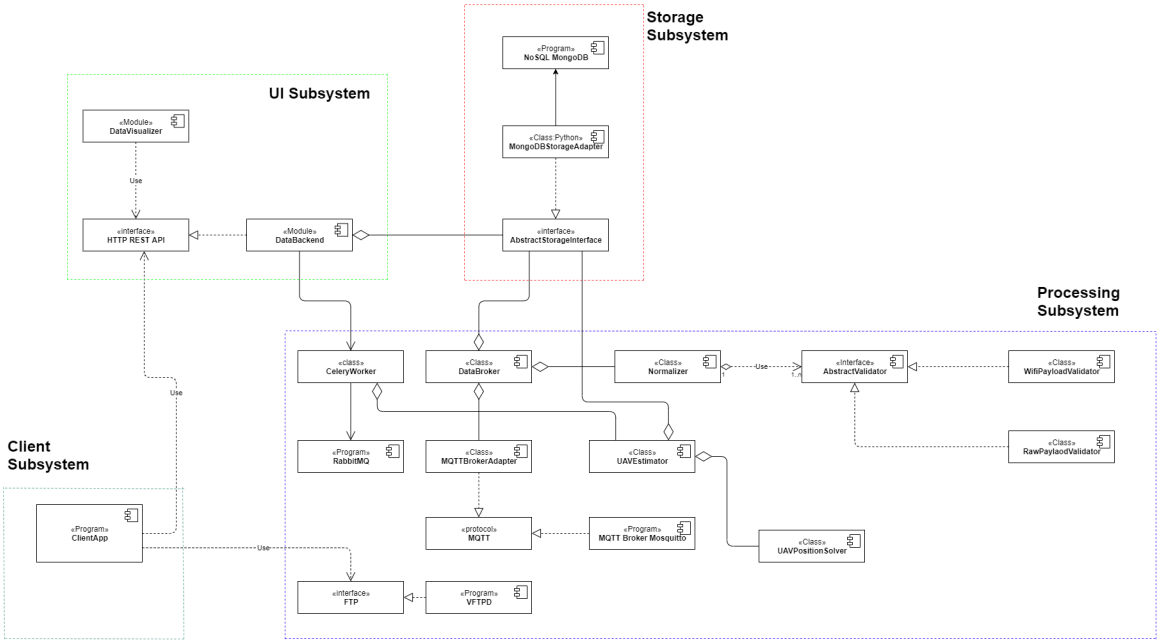

Figure 4: Level 1 Main scope

Motivation

The main motivation of this structure is the flexibility gaining by dividing the responsibility of implementation. Since the project is not so difficult, it involves a lot of different technologies and participants.

Component consideration

The framework highly utilize already completed components and protocols. The most important off-the-shelf components here are:

1. Interfaces
   - **AbstractStorageAdapter** - specify methods of accessing the data in a strict format.
   - **AbstractValidator** - defines methods to check if a message has a valid telemetry data format.
   - **HTTP REST API** - defines methods on how to access the stored data through HTTP requests.
2. Protocols
   - **FTP** - used to measure uplink/downlink throughput.
   - **MQTT** - used to communicate the telemetry messages to the MQTT Broker.
3. Program
   - FTP-server - VSFTPd - implementation of FTP server.
   - MQTT Broker - Eclipse Mosquitto - implementation of MQTT Protocol Broker.
   - NoSQL Database - MongoDB - a NoSQL database to store messages in BSON format.
   - Queue Broker - RabbitMQ - a message queue used to register the users scheduled tasks.

Level 2. Subsystems

Processing Subsystem

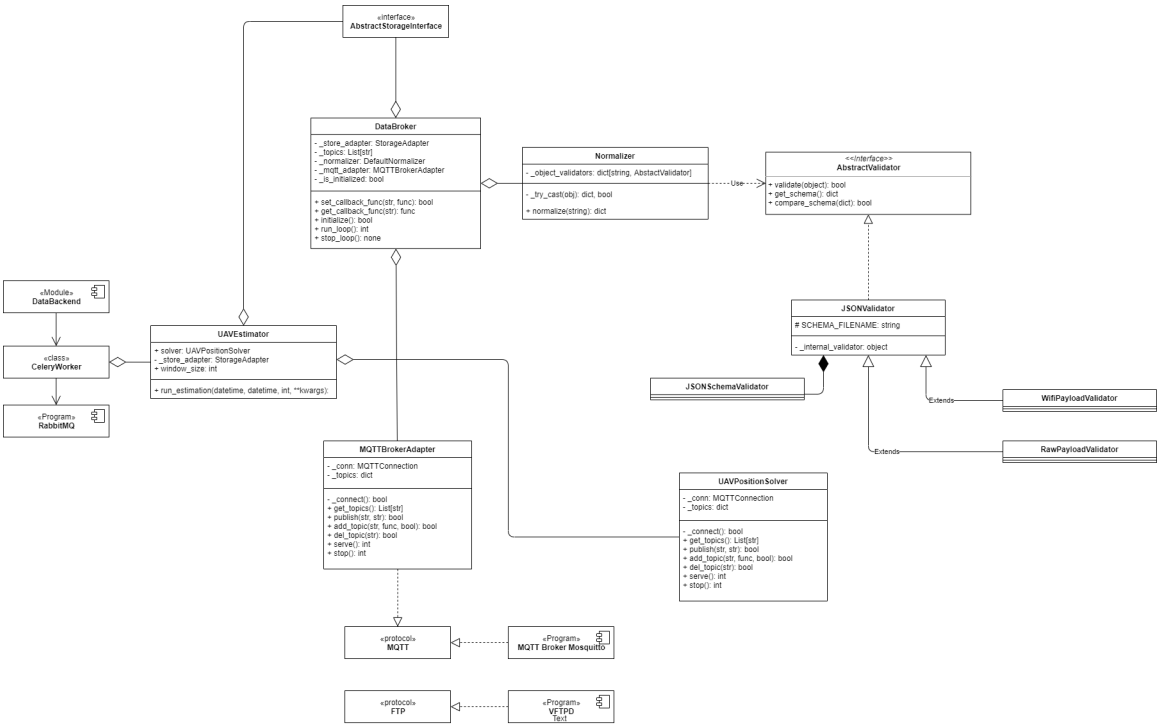

Figure 5: Processing Subsystem

Description

Processing Subsystem includes all functions to receive, prepare, transform, analyze and save the data. From the conceptual design phase it includes the following components:

- Analyzer
- MessageBroker
- DataBroker (deprecated, use direct pushing to DataBackend via HTTP)

These components currently implemented in Python. Some code regarding placement optimization algorithms are private and provided to the production environment as installation packages of code written in Python as well.

Open Issues

MQTT Broker doesn't hold message for the future

Since we use quite simple MQTT protocol, the used implementation of MQTT Eclipse Mosquitto doesn't hold messages if here no subscribers for that message's topic. It requires at least one subscriber

or set up properly QoS to guarantee that the message will be delivered and consumed properly.

One of the possibilities is to use more advanced publisher/subscriber systems such as Apache Kafka, but that has its drawbacks.

Storage Subsystem

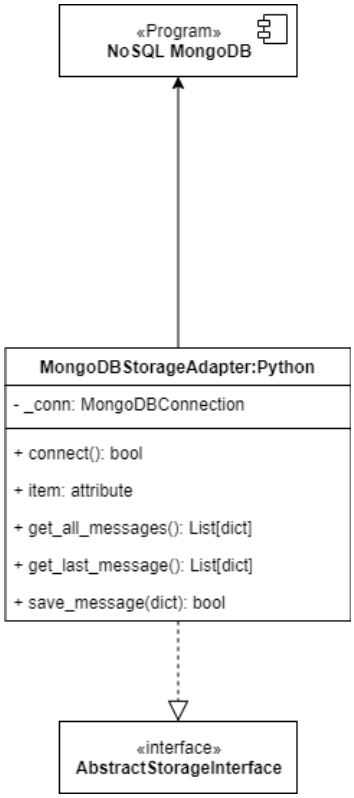

Figure 6: Storage Subsystem

Description

Storage Subsystem implements the function to access the stored data. It hides all complexity and preparation phases to get the information in the required form. This is the **Storage** component from the conceptual design phase.

Currently, only MongoDB is supported as final storage.

The MongoDB adapter is available only in Python currently.

User Interface Subsystem

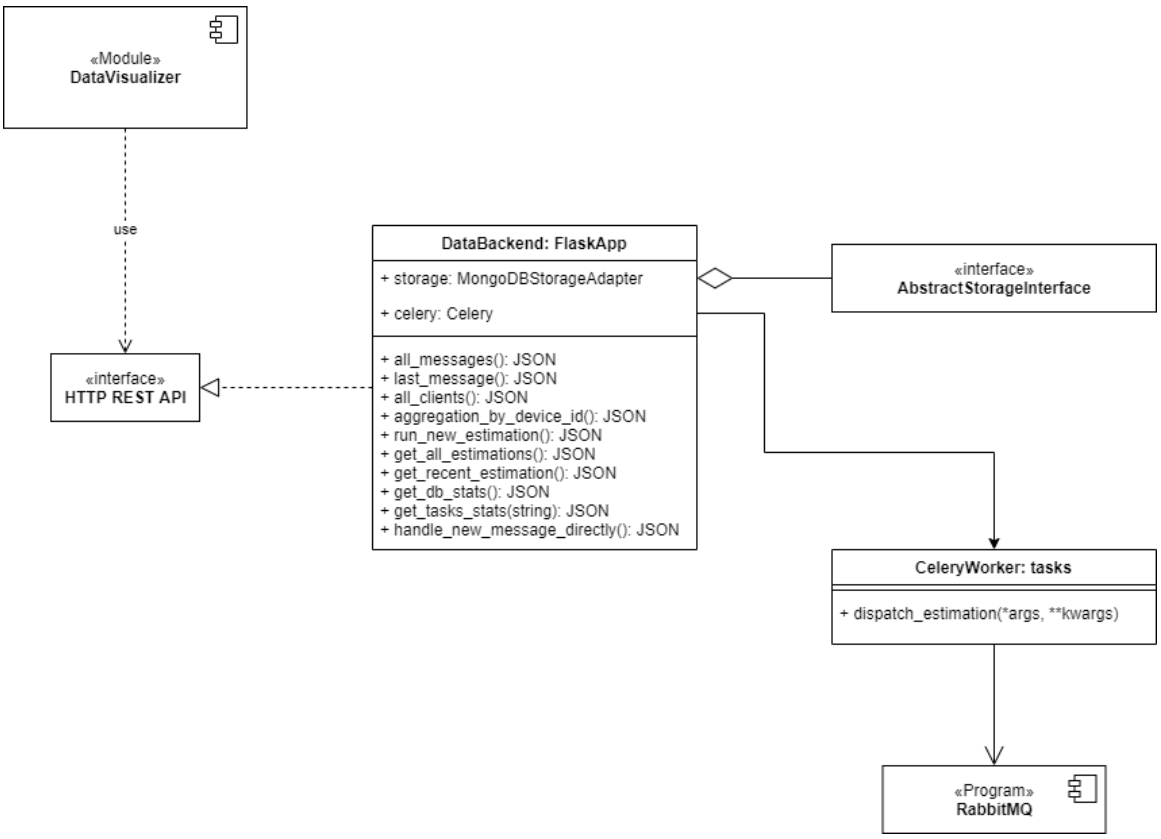

Figure 7: UI Subsystem

Description

The DataBackend is written in Python. It has access to the Celery Worker infrastructure that requires RabbitMQ. This is needed to properly register the task by the user, so it quite tangled.

The tasks performed in the background so feedback of the web backend server is quite fast. The users can check the task status through REST API (not implemented, //TODO).

This subsystem includes the components from the conceptual design phase:

- DataBackend
- DataVisualizer

The DataVisualizer is implemented as a web Single Page Application written with web technologies.

Client Subsystem

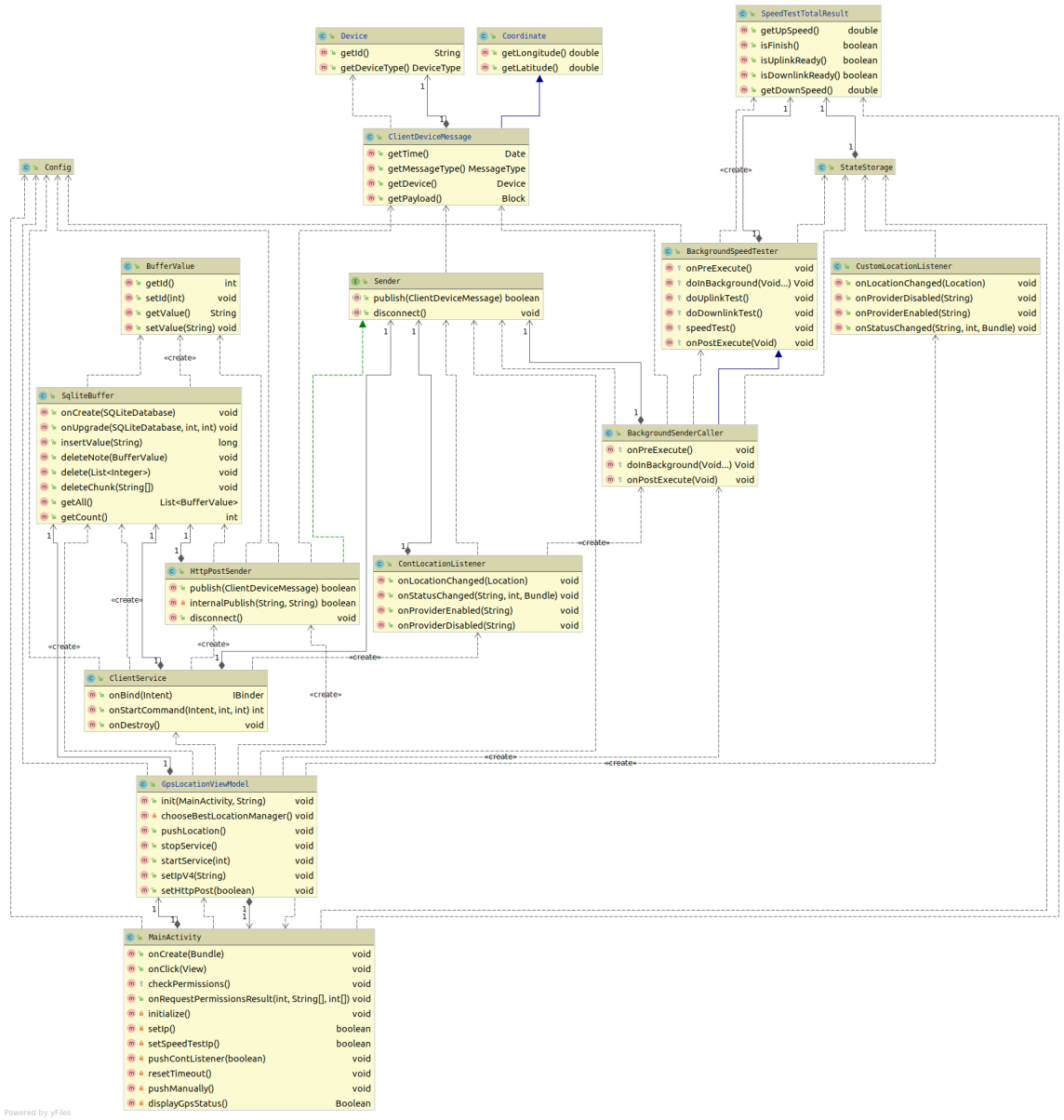

Figure 8: AndroidOverview

Android application implements MVVM architecture pattern and contains classes to collect, store and send data about location and wifi connection.

Component	Description
MainActivity	Application entry point. This class contains data which will be initialize when application is started. `onCreate` method create view using `initialization()`, check permission using `checkPermissions()`, add handlers for every button and input field.
GpsLocationViewModel	Class for init and store location manager, creating background services using `startService`, call data sender and connection quality measurement tasks
StateStorage	Storage to store last knowing location and speed test results. `MainActivity` listen to this class and update data on User Interface if data was changed
Config	Class to store project configuration, such as sender configuration, timeout, min location change handling and etc.
ClientService	Background service which retrieve connection quality and send it with given periodicity
SqliteBuffer	Sqlite data storage for messages, which were not send to selected server due to connection error
BufferValue	Data Model for Sqlite buffer storage, which contains uniq id and message for sending
Sender	Interface for send client connection quality information using `publish()` method
HttpPostSender	Sender http connection implementation using post method. If connection is not reachable, all data is stored in `SqliteBuffer`
ContLocationListener	Listener for background ClientService, this class listen location changes with the given period and call background task to send data using given `Sender`
CustomLocationListener	Listener for send data manually using `push manually` button on User Interface, This listener update last knowing location in `StateStorage`
BackgroundSpeedTester	Task to make background uplink and downlink speed tests for wifi connection using given server ip
BackgroundSenderCaller	Background task for sending data
SpeedTestTotalResult	Data model, which contain uplink and dowlink test results
Device	Data model, which contains information about device
Coordinate	Data model, which contains coordinates

Component	Description
ClientDeviceMessage	Data model, which contains information about device, coordination and connection quality

Deployment View

Requirements

Operation System

GPS_Tracker OS

The framework intended to run on Linux-based machines.

Tested OS:

- Debian 10/11
- CentOS 7

GPS_Android

The version of Android must be 4.0+ (API level 14)

GPS_Frontend

The user should use one of the modern version of web-browser HTML5-compatible:

- Google Chrome
- Mozilla Firefox
- Opera
- Apple Safari

The browser should allow running JavaScript files.

Hardware

There are no special hardware requirements.

GPS_Android Hardware

The Android phone must have Wi-Fi and GPS adapters.

Configuration

GPS_Tracker and GPS_Frontend can be configured via a configuration file provided in the projects or via the OS environment variables.

For the exact configuration instructions please check each of the detailed project configuration sections.

Deployment Cases

Both GPS_Tracker and GPS_Frontend designed to be easily deployed. There are possible two cases:

1. BareMetal Deployment - the administrator should be aware of manually starting the software elements, or configure the OS properly (via systemd, InitV scripts, etc.)
2. Docker container deployment - the projects provide Docker image description that can be started with `docker`, and a `docker-compose.yml` file to maintain the deployment phase more properly.

Each running element may start on a separated machine as long as:

1. A proper configuration performed
2. A network connection is available (NAT allowed, check for the ports if opened)

Bare-Metal Deployment

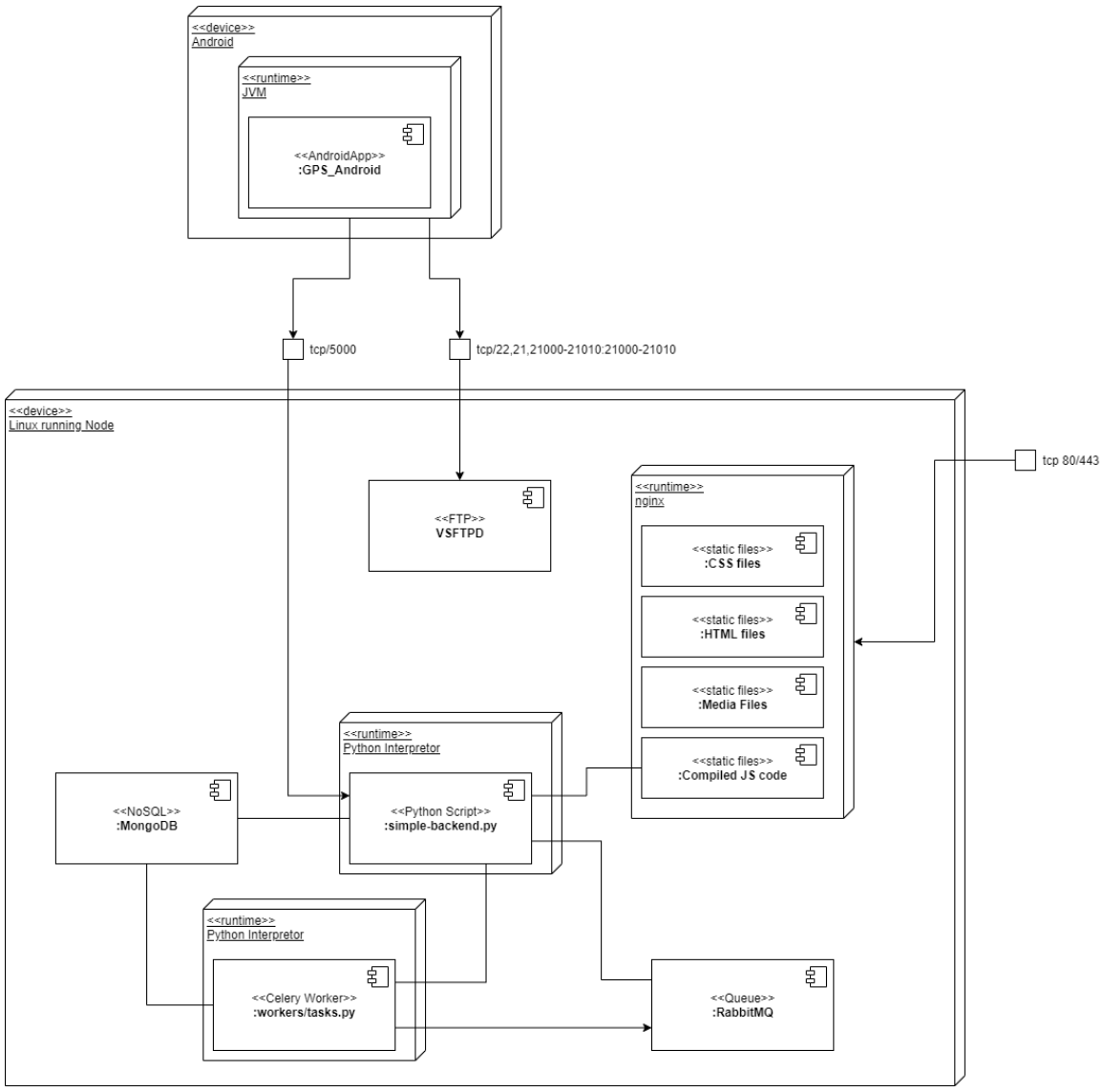

Figure 9: Bare-Metal Deployment case

Containerized Deployment

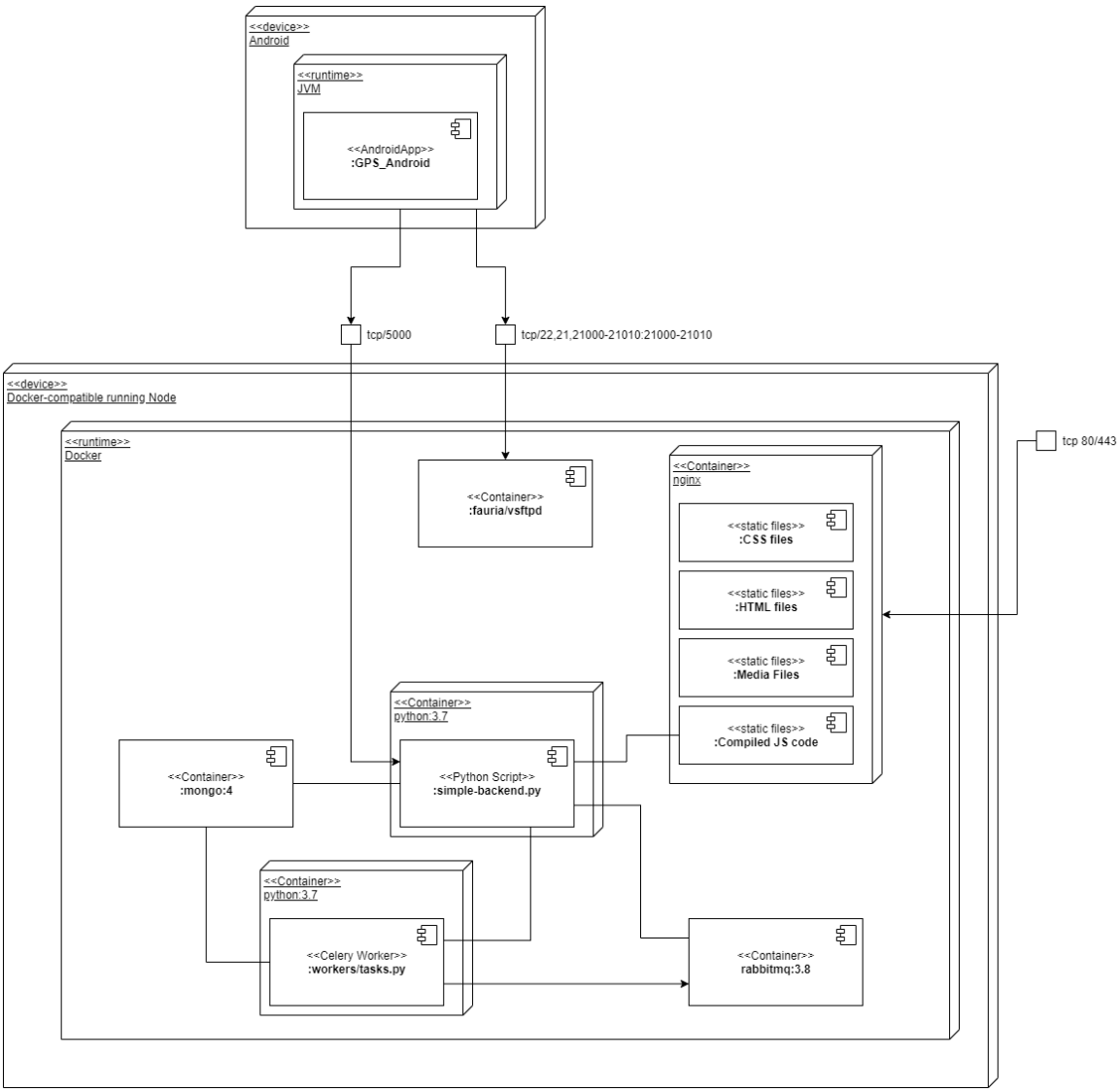

Figure 10: Docker Container Deployment case

Crosscutting concepts

GPS_Android

Used approach	Reason
In case if a message cannot be sent it saved in the local SQLite database. In the next iteration, the app first tries to send saved messages	If there is a poor signal quality that will help to save the measurement to resend them after connection become better.

GPS_Tracker

Used approach	Reason
Used Worker pattern	`Celery` workers allow to parallelize task execution.
MQTT Broker as Message Broker	MQTT Protocol is a lightweight reliable protocol for IoT intended application
MongoDB as storage for message	Use NoSQL MongoDB increase development velocity as well as performance, especially for JSON-formatted data.

GPS_Frontend

Used approach	Reason
Backend is built on Flask Python Web Framework	Flask is a light, easy-to-use framework that requires a little effort but highly extensible.
Frontend app is built using Vue.js framework	Vue.js is perfectly suited for fast development but still has good design and extensions.

Architecture Decisions

1. Android Native Development
2. Decision on GPS_Frontend techniques development
3. Decision on GPS_Tracker techniques development
4. Decision on the storage component - MongoDB
5. Consideration on the protocol used to send data from GPS_Android

Android Native Development

Description

At the beginning of the group project development, it said that a mobile app for client coordinates transfer would be necessary.

Although reasons were **against** selecting Android:

- nobody in our team has an experience developing using Android

Decision

Eventually, we **agreed** upon designing the app on the Android platform based on the following reasons:

- two members of our team write on the Java language
- Android mobile operating system has the biggest share on the market
- large and developed community
- possibility to involve many UEs to the experiment

Status

Accepted

Consequences

- we have got an app which is easy to install and use
- minimum of 6 UEs took part in our experimental phase, not a problem to involve even more
- additional changes to the app are easy to add
- huge opportunities to evolve and enlarge the project

Decision on GPS_Frontend techniques development

Description

The GPS_Frontend is a UI to access the functions of other components of the framework. Thus, it would have complicated behavior. The best-suited architecture for that kind of task is Single Page Application (SPA).

There are many possible web frameworks exist that may help to build SPA applications but the most famous and functional are:

- Angular
- React
- Vue.js

Decision

Only one person in the team had the experience developing a web application with the Vue.js framework.

The team member discussed and decided that:

- Angular has reach functionality, but too complex for that task.
- React is more flexible, but requires more time to get working in.
- Vue.js is a simple framework with an excellent documentation but has a lack of components in the default configuration. It requires additional libraries and components to design required functions.

Finally, the team decided to use the Vue.js framework as the core for **GPS_Frontend**.

Also there is a set of dependencies specified:

- “font-awesome” - A set of fonts.

- “@fortawesome/fontawesome-free” - A set of fonts.
- “apexcharts” - A library do draw figures using Web.
- “axios” - A library to work with HTTP(S) requests.
- “bootstrap” - An open source toolkit/framework for developing with HTML, CSS, and JS.
- “bootstrap-vue” - A wrapper to use Bootstrap components natively in Vue.js.
- “core-js” - An standard library to extend the number of components and operation.
- “d3v4” - level JavaScript library for manipulating documents based on data using HTML, SVG, CSS.
- “d3-colorbar” - An extension to d3 library to work with colors.
- “jquery” - rich JavaScript library.
- “moment” - A convenient library to work with time.
- “pc-bootstrap4-datetimepicker” - a DatetimePicker component compatible with Bootstrap.
- “plotly.js-dist” - A library do draw figures using Web.
- “portal-vue” - A Vue component to render your component’s template anywhere in the DOM.
- “underscore” - A JavaScript library that provides a whole mess of useful functional programming helpers without extending any built-in objects.
- “vue” - The Progressive JavaScript Framework to build web application.
- “vue-apexcharts” - A wrapper to use ApexChart components natively in Vue.js.
- “vue-axios” - A wrapper to use Axios components natively in Vue.js.
- “vue-bootstrap-datetimepicker” - datetimepicker components natively in Vue.js.
- “vue-router” - An router extension to Vue.js.
- “vuex” - A storage library extension to Vue.js.

Status

Accepted

Consequences

Advantages:

- We received a well-designed, good-looking and user-friendly UI.
- It can be easily changed and adapted to new requirements.
- Rich set of features and further development.
- High SPA performance.
- UI can be accessed easily from different devices, good portability.

Disadvantages:

- The difficulty of data analysis in JavaScript, that language probably is not the best suit for that kind of task.
- Some very complex figures (like Heatmap) may busy the whole program drawing a large dataset.
- Due to the big number of components, the JS code compilation takes a relatively long time.
- Dependency on the user’s web browser.
- Vue.js is a young project, it may suffer a lack of functions provided by more matured frameworks.

Decision on GPS_Tracker techniques development

Description

The GPS_Tracker is implemented in Python language. This is a dynamic interpreting language. That is known the performance is not so perfect compared to static compiled languages, but it has higher changeability property.

Another problem is its dynamic nature, programmers should add more codes to check that the variables they are working on have the right type, right values and so one.

Also, Python doesn't have a proper parallel threading mechanism due to GIL.

To summarize, there is a set of problems:

- Possible performance degradation.
- Ambiguous interfaces, more safe checks.
- Parallel execution, scalability is not so high.

Decision

In order to mitigate the consequences of using Python as the main language for GPS_Tracker we decided:

- Use additional code documentation and code annotation to clarify the meaning of Python code, helps IDE to derive expected behavior and check code validity.
- Use Worker architecture pattern (`Celery` library). That will help to add scalability to task execution and simplify parallel code programming.
- Use JetBrain Pycharm IDE as the main IDE for programmers to use

Status

Accepted

Consequences

- Worker pattern implemented in `Celery` requires additional component to store information about registered tasks and perform synchronization between workers. RabbitMQ is a message queue used to store that information.

Decision on the storage component - MongoDB

Description

At the beginning of our group study it was clear all messages from UEs must have been stored for further processing/representation graphically in CnC.

We decided to use the NoSQL MongoDB database to store messages.

There were some **aspects** leading us in our way of choice:

- the message structure changed as the project is growing
- The JSON message type dictated by Python usage
- availability is more important than consistency (ACID)

- queries must be fast to collect and represent data in real-time
- easy to integrate and use

Decision

As a result, MongoDB was selected for the reasons:

- Rustam has experience of using it
- fits aspects listed above more than other databases

Status

Accepted

Consequences

- The query language used in MongoDB is not SQL-compatible, required efforts to learn
- There is not strict data schema, you can change the structure of message fields easily, but also it requires more checks for validity.

Consideration on the protocol used to send data from GPS_Android

Description

During the group study solution design it became clear our system needs **a mediator** which can control incoming messages. In particular:

- avoid loss of messages
- provide the availability to many devices at once
- provide smooth transmission to the processing subsystem

One of the initial requirements were to use UDP protocol.

Decision

First, we decided to **NOT** use UDP-based protocol, especially raw UDP sockets:

- UDP connection are not reliable, you have to check the server receives messages correctly
- Calculation of metrics may be very time- and resource-consuming, so we cannot afford to lose the result of works by message losing
- A raw socket programmed software have to include additional checks, high-level integration is limited.

Therefore, we decided to use TCP-based protocols.

First, we used MQTT protocol because it well-suited for IoT-based application. MQTT is useful for the reasons:

- fitting to criterion above
- reliability
- lightweight

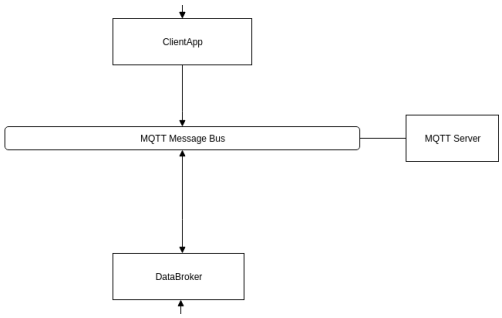

Figure 11: mqtt-justification

However, the results in the first 3 experimental attempts have shown that MQTT is not adapted for the system. Consequently, we switched to HTTP protocol to send messages directly to backend server in GPS_Tracker.

Status

Accepted, corrected

Consequences

- A simple, supported and widely adapted protocol is used, further integration with TLS if needed
- Possible additional overhead due to headers (can be solved in HTTP/2)

Risk and technical debt

Table 8: Risk and technical debt description.

Number	Description
1	There is still a lack of testing in both GPS_Frontend and GPS_Tracker.
2	Currently, MQTT enabled protocol is not stable in GPS_Android, we haven't figured out why GPS_Tracker sometimes doesn't correctly receive messages, requires to find out the reason why. So, better to rely on HTTP protocol sending method.
3	GPS_Frontend might have a long list of unnecessary dependencies that influences the final size of code supplied to users.
4	Android programming requires some pattern to be followed. Since there are limitations of programming approaches to be used as well as lack of proper programming experience, GPS_Android should be reviewed and optimized.
5	After "push continuously", switching off that button actually doesn't stop background task, therefore there is still task to test speed and send messages

Number	Description
6	“Simplex” estimation method cannot converge for 2 clusters and produce the unreliable result

Glossary

Table 9: Glossary List

Item	Description
GPS_Frontend	A web-based application to provide a user with a convenient user interface.
GPS_Tracker	A name of the framework and as well a name for the backend software running for processing and storing telemetry data
GPS_Android	An Android application to send the telemetry information about the current mobile network from the users’ phones to GPS_Tracker.
Telemetry Data	A message sent from the users’ phone. Includes RSS level, uplink/downlink throughput measurement, GPS coordinates.
Broker	A special architectural pattern, program, that is the point of communication between components.
MQTT	Message Queuing Telemetry Transport, an publisher/subscriber-based application protocol running over TCP/IP, suitable for IoT application.
JSON	JavaScript Object Notation, a lightweight data-interchange format.
MongoDB	A NoSQL database storing data in BSON (Binary JSON) format.
Celery	A framework to implement a program using “Worker” architecture design.
RabbitMQ	An in-memory queuing message broker.
RSS	Received Signal Strength.
GPS	Global Positioning System.
UE(s)	User Equipment.
UAV(s)	Unmanned Aerial Vehicle.
AP	Access Point

Project Documentation

Usage

Generate a PDF file:

```
pandoc *.md -H disable_float.tex -o "Performance analysis framework for base station placement usi
```

Description

Welcome to the documentation!

The documentation based on arc42 suggested template.

The following sections are available:

1. Introduction and Motivation
2. Solution strategy
3. Contex and Scope
4. Solution Strategy
5. Building Blocks
6. Deployment View
7. Crosscutting concepts
8. Architecture Decisions
9. Risk and Technical Debts
10. Glossary

Also, you can find out more detailed information on the sections:

- About System Components - link
- About Using Data structures - link
- Description of Protocols and Interfaces - link
- Diagrams and Schemes - link
- About System Launching - #TODO:
- About System Testing - link