Bachelor's thesis

Information and Communications Technology

2019

Matti Lindholm

IOT-BASED ASSET TRACKING AND MEASUREMENT MONITORING PLATFORM

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2019 | 38 pages, 2 in appendices

Matti Lindholm

IOT-BASED ASSET TRACKING AND MEASUREMENT MONITORING PLATFORM

The number of IoT devices connected to the Internet is increasing every day. Some of these devices are sensor devices that enable passive and efficient data acquisition in construction sites, buildings and vehicles. These features allow close monitoring of changing conditions and movements of assets such as persons.

This thesis contains an implementation of a web user interface (UI) designed for condition monitoring and asset tracking use-cases. The background architecture which provides the data for this UI is provided by an IoT platform. The implementation of the IoT platform is documented along with the software frameworks utilized to implement it. The goal of this thesis is to provide an overview of this specific IoT platform and document the most important features of its UI which was developed as a real-life work development task.

This thesis concludes that IoT data can be effectively utilized using software frameworks and thus IoT related software solutions can be efficiently implemented. This thesis also contains numerous practical examples of utilizing these described software frameworks so that their practicality can be understood. The documented IoT platform was developed at Fidera Ltd.

KEYWORDS:

IoT, IoT Platform, VueJS, LoopBack, Web Application, Tracking, Monitoring

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2019 | 38 sivua, 2 liitesivua

Matti Lindholm

IOT-POHJAINEN KOHTEIDEN JA OLOSUHTEIDEN SEURANTA-ALUSTA

Internetiin kytkettyjen laitteiden määrä kasvaa päivä päivältä. Osa näistä laitteista on anturilaitteita, jotka mahdollistavat passiivisen ja tehokkaan tiedonkeruun esimerkiksi rakennustyömailla ja asuinrakennuksissa. Nämä ominaisuudet mahdollistavat muuttuvien olosuhteiden ja kohteiden, kuten henkilöiden ja ajoneuvojen seurannan.

Tämä opinnäytetyö sisältää verkkokäyttöliittymän toteutuksen, joka on tehty tähän tarkoitukseen. Käyttöliittymän tausta-arkkitehtuuria kutsutaan IoT-alustaksi, josta tarjotaan yleiskuva sen toteuttamiseen käytettyjen ohjelmistokehysten lisäksi. Opinnäytetyön tarkoitus on dokumentoida siinä kuvattu IoT-alusta ja toteutetun käyttöliittymän tärkeimmät ominaisuudet.

Opinnäytetyön johtopäätös on, että IoT-dataa voidaan hyödyntää tehokkaasti ohjelmistokehyksiä käyttäen, joka mahdollistaa IoT:hen liittyvien ohjelmistoratkaisujen toteuttamisen. Opinnäytetyö sisältää myös runsaasti käytännön esimerkkejä siitä, miten näitä kuvattuja ohjelmistokehyksiä voidaan käyttää. Opinnäytetyössä kuvattu IoT-alusta toteutettiin Fidera Oy:ssa.

ASIASANAT:

IoT, IoT-alusta, VueJS, Vuetify, LoopBack, verkkosovellus, sensori

CONTENT

APPENDICES

Appendix 1. Pictures of Analytics UI

FIGURES

LIST OF ABBREVIATIONS

API Application Programming Interface. Contains methods that a client application such as a web page can use. For instance, an API can be used to request and record data.

CLI Command-Line Interface. A command-line based program for handling user commands.

CRUD Create, Read, Update and Delete. Acronyms for performing create, read, update and delete operations to a relational database.

DOM Document Object Model

GPS Global Positioning System

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IoT Internet of Things

JSON JavaScript Object Notation. Open-standard file and data interchange format consisting of key-value pairs.

npm Node package manager. Handles application code dependencies.

OS Operating System such as Windows, Macintosh or Linux.

REST Representational State Transfer

SRID Spatial Reference System Identifier

SRS Spatial Reference System

SQL Structured Query Language. A standardized query language developed by IBM that can be used to perform CRUD operations to a relational database.

TLS Transport Layer Security. Cryptographic protocol designed to provide security for communication over a computer network.

UI	User Interface
URL	Uniform Resource Locator. A web address which is a reference to a web resource.

1 INTRODUCTION

The Internet of Things (IoT) refers to the connection of devices to a network, through which they transmit data collected by sensors. The total number of worldwide installed IoT devices is estimated to rise to 75.44 billion by 2025. This is a fivefold increase in ten years time. (Statista, 2016). An IoT device is capable of transmitting data over a network without human or computer interaction. (TechTarget, n. d.). IoT devices such as sensors are able to provide measurements from their surroundings. However, this data needs to be stored and processed so it can be displayed in an application such as a user interface (UI). Services related to storing and processing measurement data are provided by what is called an IoT platform. There exists previous studies and thesis works documenting the concepts and technologies related to IoT systems and their implentations such as Internet of Things and IoT platforms. (Uppa, J. 2017). However, this thesis is different because it provides an overview using practical examples of how to build an IoT platform.

During 2019 an IoT platform was developed at Fidera Ltd. The platform contains features for several condition monitoring and asset tracking use-cases. The purpose of this thesis is to provide an overview of the discussed IoT platform, document the utilized software frameworks, explain how they can be applied to build applications and to work with IoT data. Besides focusing on documenting the used front-end and back-end technologies a web UI was built to provide functionality for these discussed features. The UI in question is titled Analytics UI. It was built as a real-life work development task and it is the work of the author of this thesis as a whole. The main UI features are documented and these include sensor-based condition monitoring and the ability to track assets such as persons, devices or vehicles and visualize their locations. The demand for such features is based on needs such as monitoring employee working conditions on construction sites and asset usage in order to prevent unwanted events such as inactivity or theft.

2 INTERNET OF THINGS

In order to understand IoT-based solutions it is necessary to have an overview of IoT and the related generic concepts. The objective of this chapter is to provide a brief overview of IoT and explain what is an IoT platform.

2.1 Overview

The objective of IoT is to make anything ’smart’ by enhancing everyday objects with the power of data collection. A sensor is a simple component which is able to observe and measure an event in the physical world. A sensor device is a small, low-cost and battery-operated device that measures these events and has the ability to transmit information to a network. An actuator is a component or a mechanism which is able to transform digital information to an action in the physical world thus being able to influence its surroundings. An example of this action would be turning another device on or controlling lights in a room. IoT network traffic is typically bidirectional which allows for remote control of the networks devices. The information collected by sensor devices is usually sent to cloud services where the information can be processed, analyzed and visualized in a user interface. (Uppa, J. 2017).

IoT has advantages that span across different areas of life and business. Examples of these include smart homes and environmental monitoring applications. The underlying architecture composed of various services provides this connectivity and is called an IoT platform.

2.2 IoT platform

An IoT platform offers communication, data processing, analytics, device management capabilities and software applications. It links machines and devices to cloud services where the information sent by them is stored, processed and provided for applications. Key features of IoT platforms include device management capabilities enabled by bidirectional network communication, open-ended life cycle of platform applications which means the creation and improvement over time of these applications, data analytics which includes basic descriptive analytics, visualizations and predictive

analytics. The platform has to be maintainable so that the existing system can be picked up and understood by new developers. The platform should also be secure and scalable so that new applications and devices may be supported in the future. (Perry, M. 2016).

3 PLATFORM BACK-END TECHNOLOGIES

This chapter discusses the underlying platform architecture of Analytics UI and the related software frameworks. The platform architecture is implemented using some of the newest technologies available which are especially suitable for working with IoT data. The circled sections in Figure 1 are the focus of this thesis and will be discussed in the following chapters.

3.1 Platform architecture

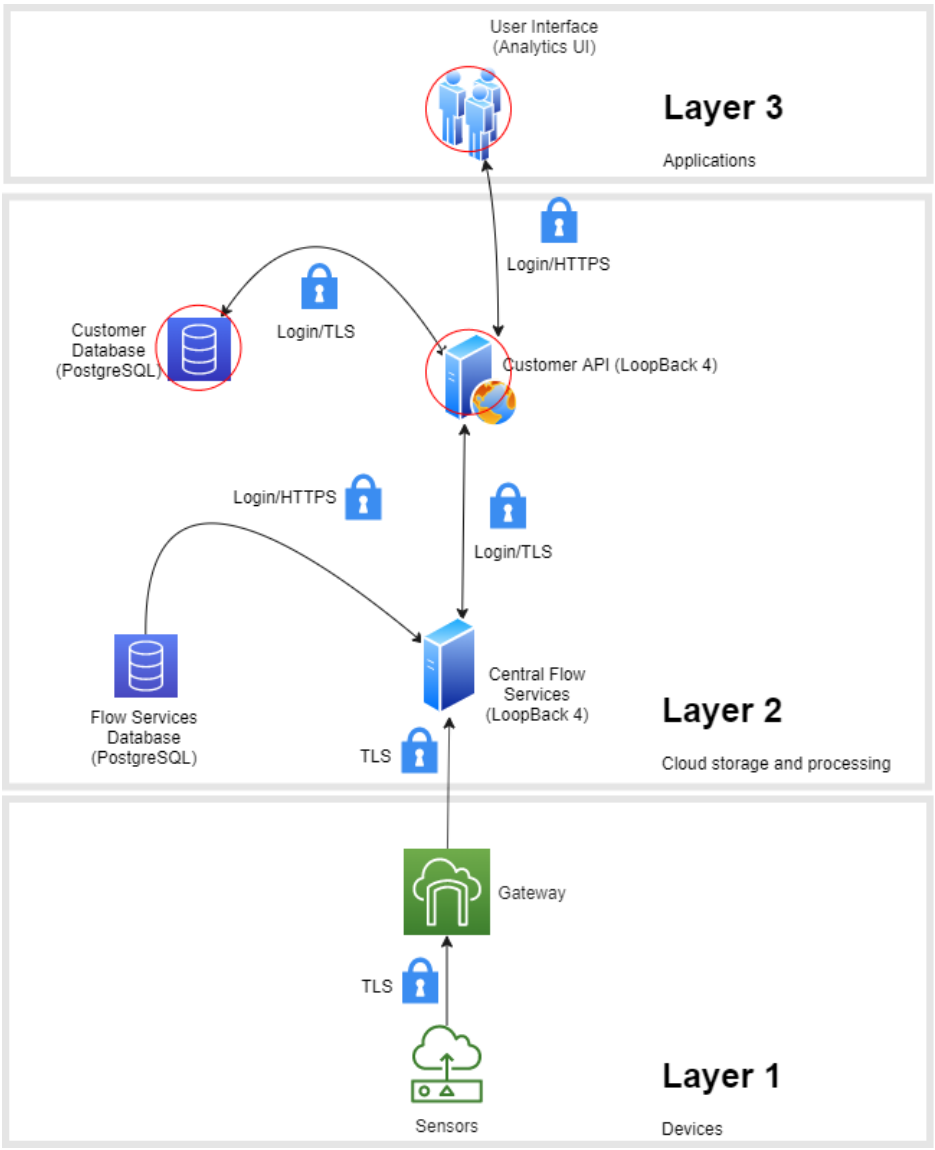


Figure 1. Depiction of the platform architecture

Platform architecture refers to the structure of the platform's software system. The architecture can be divided into three layers where each layer has its own functionality.

Layer 1 can be thought of as the device layer. It contains sensors and other devices that measure and collect information. This layer also includes the gateway devices which are responsible for transmitting the sensor data to the network.

Layer 2 contains the cloud storage and data processing services. Data sent by sensors and devices is stored to databases and processed in servers. 'Central Flow Services' is responsible for receiving and decoding sensor data and transmitting it to the correct customer API. The customer API then analyzes and processes the data into the correct format so it can be displayed in the user interface of a software application such as Analytics UI which is documented in Chapter 5.

Layer 3 includes the applications that are utilized by the end users of the platform. Any number of new applications may be added to the platform. The possibility of adding new applications and devices to the platform are key components considering platform scalability.

3.2 LoopBack4

The platform back-end servers have been implemented using LoopBack4 which is the latest version of the open-source LoopBack NodeJS framework. LoopBack4 will be referred to as 'LoopBack' in the following chapters. LoopBack enables you to quickly create APIs as microservices composed from existing back-end systems such as databases and has support for TypeScript which stands for strongly typed JavaScript. The APIs are exposed as endpoints for client applications. LoopBack uses the OpenAPI specification. (LoopBack, 2019a).

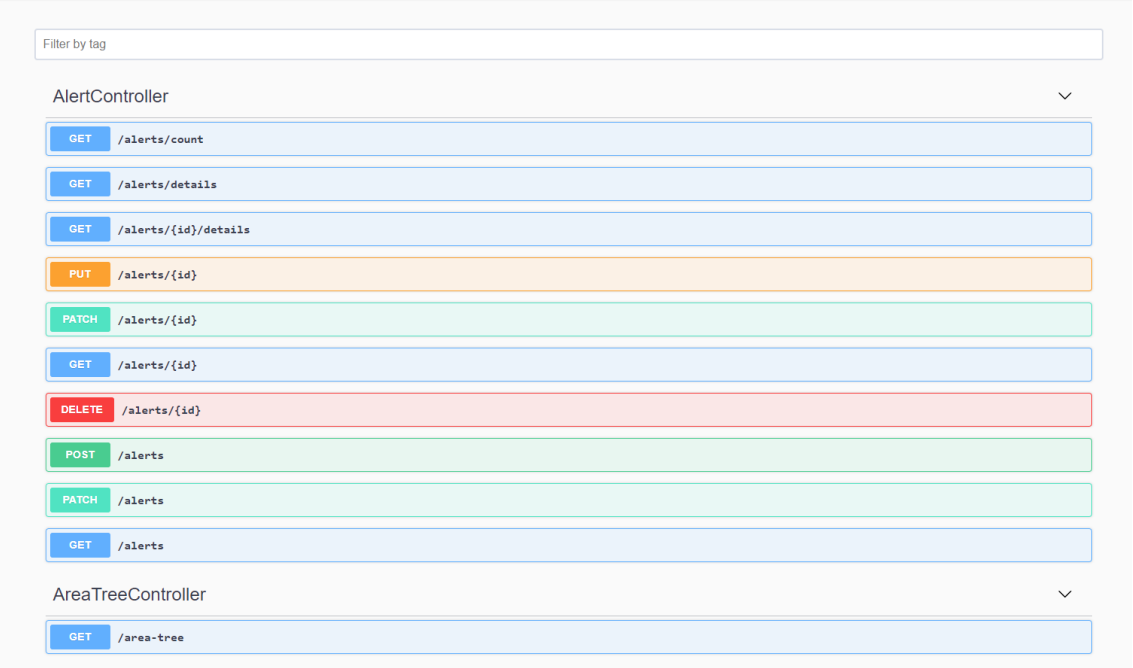


Figure 2. The browser-accessible OpenAPI Explorer of LoopBack4

OpenAPI is a specification for visualizing web services that use Representational State Transfer (REST). It is an HTTP application protocol based architectural style with guiding constraints for distributed hypermedia systems. Interaction with LoopBack's REST APIs is implemented using HTTP GET/PUT/POST/DELETE and PATCH methods. Request responses are in JSON format of which an example is shown in Figure 3. OpenAPI defines a standard interface description for REST APIs. In essence, it provides information about a service for humans and computers. LoopBack uses the OpenAPI specification to generate APIs as shown in Figure 2. (OpenAPI Initiative, 2019).

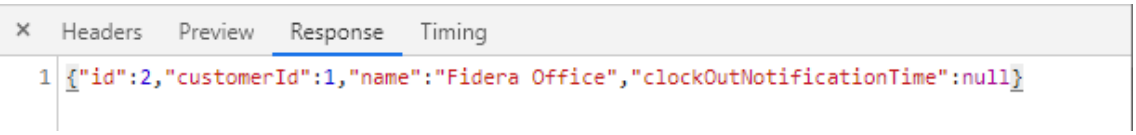


Figure 3. JSON response to a HTTP GET

3.2.1 Structure of a LoopBack application

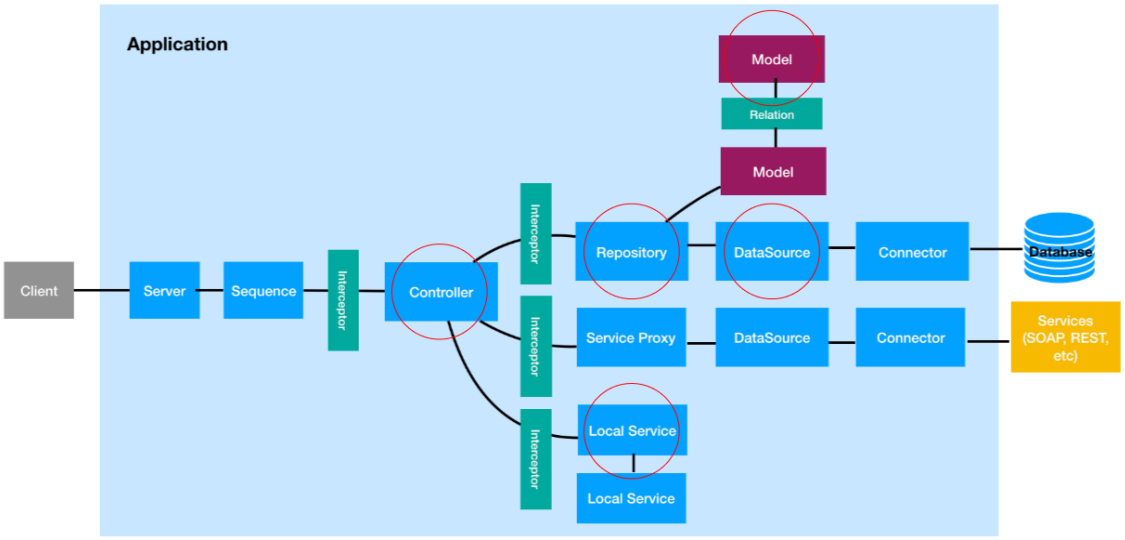


Figure 4. Key concepts of LoopBack (LoopBack, 2019b).

The purpose of this chapter is to describe the most fundamental parts behind the application structure of a LoopBack API and the key concepts related to these parts which have been circled in Figure 4. LoopBack provides the possibility to create APIs as microservices which means that each API runs on an independent server. *Server* is an implementation for inbound transports like HTTP and protocols such as REST and listens on a specific endpoint such as 'http://localhost:3002' (hostname, port number), handles incoming requests and returns appropriate responses. *Application* is the central class for a module which contains the functionality of an API. (LoopBack, 2019b). The LoopBack Command-Line Interface (CLI) can be installed using the command-line with npm. The installation procedure is described in Figure 5.

```
npm i -g @loopback/cli
lb4 app
cd app
npm start
```

Figure 5. Installing LoopBack CLI, creating and starting an application

LoopBack CLI is used to create APIs that consist of modules. A module is created in a series of steps and can be initialized using the CLI command 'lb4 <class>' where '<class>' is replaced using the name of the desired module class. These classes are

called model, datasource, repository, controller and service. They are at the foundation of a LoopBack API.

Model contains a definition of a business domain object which is the representation of the data used by the API. For instance, a JSON object conforming to a model definition such as 'Person' can be passed in REST/HTTP payload which is used to create a new instance of the 'Person' model in a database. Model definitions can also be mapped to other forms like JSON and OpenAPI schemas. A model in LoopBack is JavaScript class. (LoopBack, 2019c). Example of a model definition of 'Person' is described in Figure 6.

Figure 6. Model definition of Person

```
export class Person {
    name: string;
    address: string;
    city: string;
    phoneNumber: string;
    email: string;
}
```

Two types of models exist for domain objects. *Value Object* does not contain an identity (ID) because its equality is based on the object as a whole. For instance, if two 'Person' objects contain identical key-value pairs, they are equal. *Entity* is a domain object with an ID and its equality is based solely on the ID. Two person objects that have the same ID are equal because the ID refers to the same person. (LoopBack, 2019c).

Datasource is the connection between sources of data such a database and an API. It is typically used together with a repository to provide access to data. (LoopBack, 2019d).

Repositories represent a Service interface which can be utilized to perform CRUD operations against an underlying database. For instance, a repository may contain SQL statements which can be utilized by a datasource. However, using LoopBack the developer may not need to write custom SQL statements because LoopBack provides filters and functions that can be utilized to perform database operations. An example of such an operation is provided in Figure 8. (LoopBack, 2019e).

Operation	SQL	HTTP
Create	INSERT	PUT / POST
Read	SELECT	GET
Update	UPDATE	PATCH / PUT / POST
Delete	DELETE	DELETE

Figure 7. CRUD acronyms with matching SQL and HTTP methods

Controllers are responsible for handling the request response-lifecycle of an API. A function can be utilized to deal with an incoming request such as HTTP GET to an API. (LoopBack, 2019f). CRUD acronyms such as GET are explained in Figure 7.

Figure 8. LoopBack controller for HTTP GET

```
@get('/position-observations/{id}', {
  responses: {
    '200': {
      description: 'PositionObservation model instance',
      content: {'application/json': {schema: getModelSchemaRef(PositionObservation)}},
    },
  },
})
async findById(@param.path.number('id') id: number): Promise<PositionObservation> {
  return this.positionObservationRepository.findById(id);
}
```

Figure 8 contains an example of the relation between a controller and a repository where the ‘findById’ function returns a function call to another repository. In this case the repository is named as the ‘positionObservationRepository’ which is responsible for making a data query to a database for position observations.

Services contain methods for performing remote or local operations. (LoopBack, 2019g). For instance, a function responsible for parsing data to the suitable format for a chart could be in a service. A service such as this could be called from a controller. Understanding relations between models, datasources, repositories, controllers and services enables the developer to create LoopBack APIs.

3.3 PostgreSQL

The databases of the platform have been implemented using PostgreSQL which is an open-source object-relational database with development history that spans three decades. PostgreSQL extends the SQL language, includes features for safely storing complicated data workloads and has powerful add-ons that are referred to as extensions.

This chapter will discuss the installation of PostgreSQL in Ubuntu versions 16, 18 and PostgreSQL extensions along with examples of how they can be best utilized for working with IoT data. (PostgreSQL, 2019a).

3.3.1 Installation

PostgreSQL is available in the default Ubuntu repository. To get the latest releases the PostgreSQL advanced package tool (apt) repository has to be added to the Operating System (OS). apt is a command-line tool for handling software packages for Debian and Debian based Linux distributions such as Ubuntu.

Before starting the installation, the Gnu Privacy Guard key (GPG) has to be first imported for PostgreSQL packages to enable the installation of the latest PostgreSQL versions. GPG is an encryption technique that was developed for use in e-mail exchanges but is now used in a number of applications such as code signing Linux and GitHub repositories. (PostgreSQL Wiki, 2019).

```
# Import GPG key and add the PostgreSQL repositories to system

# `lsb_release -c -s` returns the codename of your OS
sudo sh -c "echo 'deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -c -s`-pgdg main'
>> /etc/apt/sources.list.d/pgdg.list"

wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

# Download and install PostgreSQL (v. 11)
sudo apt-get install postgresql-11

# Switch to the user 'postgres'
sudo su postgres

# Create a database called 'exampledb'
/usr/bin/createdb exampledb

# Connect to the PostgreSQL database server
psql

# Connect to the 'exampledb' database
\c exampledb
```

Figure 9. Installing PostgreSQL, creating and connecting to a database

After importing the GPG key, PostgreSQL can be installed using the command-line by adding PostgreSQL repositories to the OS, downloading and installing PostgreSQL as

described in Figure 9. After a successful installation, PostgreSQL databases can be created. SQL statements can be run to the database using the ‘psql’ command-line tool.

3.3.2 Extension: TimescaleDB

TimescaleDB is an extension built on top of PostgreSQL for handling time-series data which is data that represents how a process changes over time. Time-series data has three important characteristics. It is time-centric which means that records always have a timestamp, data is only inserted, not updated and recent. IoT data can be classified as time-series data. Like PostgreSQL TimescaleDB is also open-source. (Timescale, Inc. 2019a).

After the installation of PostgreSQL, a matching version of TimescaleDB may be installed using the command-line.

```
# Add Timescale's personal package archive (PPA)
sudo add-apt-repository ppa:timescale/timescaledb-ppa
sudo apt-get update

# Install TimescaleDB matching the installed PostgreSQL version
sudo apt install timescaledb-postgresql-11

# Connect to a database
sudo su postgres && psql && \c exampledb

# Extend the database with TimescaleDB
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

Figure 10. Installing TimescaleDB

TimescaleDB installation contains the following steps that are listed in Figure 10:

- Adding TimescaleDB’s personal package archive to the OS
- Installing TimescaleDB using apt
- Connecting to an existing PostgreSQL database using the ‘psql’ command-line tool and running the TimescaleDB SQL extension statement

The purpose of TimescaleDB is to facilitate working with time-series data. For example, using common aggregate functions such as average or mean on time-series data is easy using TimescaleDB. When visualizing large amounts of data, it is often necessary to select a picking interval to limit the amount of data points in a chart. For instance, it would

be possible to count averages of every selected 10 or 100 values. (Timescale, Inc. 2019b).

Figure 11. Utilizing the 'time_bucket' function of TimescaleDB

```
SELECT time_bucket('30 days', 100) AS month, MAX(temperature) as max_temp
FROM readings
GROUP BY month
ORDER BY month DESC;
```

SQL query described in Figure 11 utilizes TimescaleDB's 'time_bucket' function to count monthly averages of temperature readings and returns the averages of every 100 readings of n-amount of temperature readings. Utilizing similar SQL queries provides data for common chart-based visualizations of which examples are included in Appendix 1.

3.3.3 Extension: PostGIS

This chapter introduces a technology due to its potential applicability for location data related use-cases described in Chapter 5. PostGIS is a spatial database extender for PostgreSQL that adds support for geographic objects so location queries may be run in SQL. Although PostGIS is designed for spatial data, it is also possible to store non-spatial data in a PostgreSQL database which has been extended with PostGIS. (PostGIS n.d. a). PostGIS adds spatial functions to the database such as area, distance, union, intersection and special geometrical data types. These spatial functions may be used to answer questions like 'How close is the nearest asset?' or 'Is this asset inside a certain area?'. It has a geometry column with data in a specific coordinate system defined by a Spatial Reference System Identifier (SRID). (PostGIS n.d. b).

SRID is part of a spatial reference system (SRS) which is a coordinate-based system used to locate geographical entities. SRID is a unique value which is used to unambiguously identify local spatial coordinate system definitions that are different systems for assigning coordinates to locations. (OGC, 2019).

Figure 12. SQL query to a PostGIS-extended PostgreSQL database

```
SELECT person.name
FROM city, person
WHERE ST_Contains(city.geom, person.geom)
AND city.name = 'Turku';
```

Figure 12 contains an SQL query which returns information about the locations of persons. The query calls a PostGIS function named ‘ST_Contains’. This function returns a Boolean value which is *true* if the person is in Turku and *false* if the person is not in Turku. With features such as the ones described in this chapter, PostGIS can be utilized in creating modern geofencing applications.

4 PLATFORM FRONT-END TECHNOLOGIES

The focus of this chapter is in the front-end frameworks utilized in the development of Analytics UI. The term ’front-end’ is commonly used to describe the user interface of a web application.

Frameworks in general provide functionalities that can be extended by writing additional code. This chapter will discuss a front-end framework called VueJS as well as Vuetify which is a material component framework. Vuetify provides a base style for UI components and an easy-to-use component API. VueJS will be referred to as Vue in the following chapters. The use of a front-end framework such as Vue includes valuable features such as:

- Generic ’out of the box’ functionalities
- Open-source, reliable and tested code base which is maintained
- Tools for fast prototyping
- Comprehensive documentation

4.1 Basics of VueJS

Vue is a framework for building user interfaces. It was created by Evan You after working for Google using AngularJS. Vue is fast, easy to pick up and integrate with other libraries and implements a component-based architecture. A Vue project can be initialized by installing the Vue CLI and creating a project template. (You, E. 2019a).

4.1.1 Vue CLI

Creating a Vue project starts by installing the Vue CLI which will be referred to as the CLI in the following chapters. The CLI offers a collection of official plugins for fast application creation and management. Additionally, a full graphical user interface is also provided. The CLI can be installed globally using the Node Package Manager (npm) which is the default package manager for the JavaScript runtime environment NodeJS. (You, E. 2019b). The installation and the creation of a template Vue application is described in Figure 13.

```
npm install npm@latest -g
npm install -g @vue/cli-service-global
vue create my-vue-app
```

Figure 13. Installing npm, Vue CLI and creating a template Vue application

npm can be used to install packages that are reusable pieces of software which can be downloaded from registries. When a package is added to an application, the application is dependent on the package, so the package becomes an application dependency. The list of application dependencies is stored in a file called ‘package.json’. If ‘package.json’ is present in a project application dependencies can be installed by running the command ‘npm install’ in the project folder using the command-line.

4.1.2 Component-based architecture

Vue implements a component-based architecture which means that a Vue application is composed out of components. Component-based architecture is an architecture model that is based on reusable sets of functionalities encapsulated as components. Components have their own structure, methods, APIs and they are reusable. For instance, a component may be a part which makes up a segment of a user interface. Components are also updated when a change occurs in the data used by the component. This feature is called reactivity and it is based on the link between the component data and the Document Object Model (DOM). (Shapiro, D. 2016).

The DOM is an interface which treats the HTML document as a tree structure where every node is an object that represents a part of the document. The DOM can be manipulated using programming languages such as JavaScript. (W3C, 2005).

Components have a specific code structure. The HTML code of a component is wrapped inside ‘template’ tags. ‘script’ tags contain the component’s JavaScript code such as data structures and functions.

Figure 14. Basic component structure

```
// ParentComponent.vue
<template>
  <div>
    <h1>This is the parent component</h1>
    <ChildComponent :childData="parentData" v-if="childComponent" />
  </div>
</template>

<script>
export default {
  name: 'ParentComponent',
  data: () => ({
    childComponent: false,
    parentData: 'This is the parent data passed to the child',
  }),
  components: {
    ChildComponent: () =>
      import('@/components/ComponentExample/ChildComponent.vue'),
  },
  mounted: function() {
    this.childComponent = true;
  },
};
</script>

// ChildComponent.vue
<template>
    <p>{{ childData }}</p>
</template>

<script>
export default {
    name: 'ChildComponent',
    props: {
        childData: {
            type: String
        }
    },
}
</script>
```

Components can contain components and it is possible to pass data between them. This is a basic component feature of which an example has been implemented in Figure 14 where:

- Two components have been created and named ‘ParentComponent’ and ‘ChildComponent’. In practice these components are two different files which have been included in Figure 14 for demonstration purposes

- ParentComponent contains a data structure of type string named ‘parentData’ which is passed on to the ChildComponent using the name ‘childData’
- The passed data is declared in the ChildComponent using the ‘props’ object which contains the name and type declarations of the passed data

Components have a lifecycle which contains different stages of initialization. These stages are called lifecycle hooks that are functions that give the developer an opportunity to run code at specific stages of initialization which a Vue instance (a component) goes through. These stages include template compiling, setting up data observation, mounting and updating the component instance to the DOM when data used by the component changes. (You, E. 2019c).

Figure 15. Utilizing the lifecycle hooks of Vue

```
created: async function() {
  await this.getData();
  this.interval = setInterval(
    function() {
      this.getData();
    }.bind(this),
    60000,
  );
},
beforeDestroy: function() {
  clearInterval(this.interval);
},
```

Figure 15 contains an example where the ‘created’ lifecycle hook of Vue has been utilized. ‘created’ contains a function called ‘getData’ which sends an HTTP GET to an API. Using JavaScript’s ‘setInterval’ function ‘getData’ is set to run every 60 seconds so each minute another request is triggered, and data gets refreshed. The ‘beforeDestroy’ lifecycle hook is utilized to clear the interval to stop the occurring requests before the component is destroyed.

Conditional rendering is a feature of Vue and can be used to decide based on a condition if an element will be rendered. Vue offers specific conditional directives such as ‘v-if’ which can be used directly in HTML to decide if an element such as a component will be rendered. For example, using the ‘v-for’ directive it would be possible to dynamically create a list of items on the screen or use ‘v-if’ to display a progress bar during updates as in Figure 16 where ‘progressBar’ refers to a Boolean variable.

Figure 16. Using the 'v-if' directive of Vue to display a progress bar

```
<v-progress-linear v-if="progressBar" :indeterminate="true">
</v-progress-linear>
```

Framework features such as reactivity, lifecycle hooks and conditional rendering make the developer's work easier. The generic 'out of the box' functionalities described in this chapter are at the foundation of building Vue applications.

4.2 Building large-scale applications with VueJS

Vue is suited for building large-scale applications. There are specific packages that are part of Vue's core repositories and increase the conveniency of building large-scale applications. This chapter will discuss these packages that can be installed as application dependencies using npm.

4.2.1 State management using Vuex

Once the application grows larger and more complex, the need for modularity is increased. This necessitates splitting the code into several components based on their respective uses. Sharing data between components can be implemented using a package called Vuex. Vuex is a state management library for Vue applications. Vuex contains a store which all the applications components can share. This reduces the need of repeating component-specific code such as often used API calls and data structures. (Vuex, n.d.). How the different parts of Vuex interact with each other is described in Figure 17.

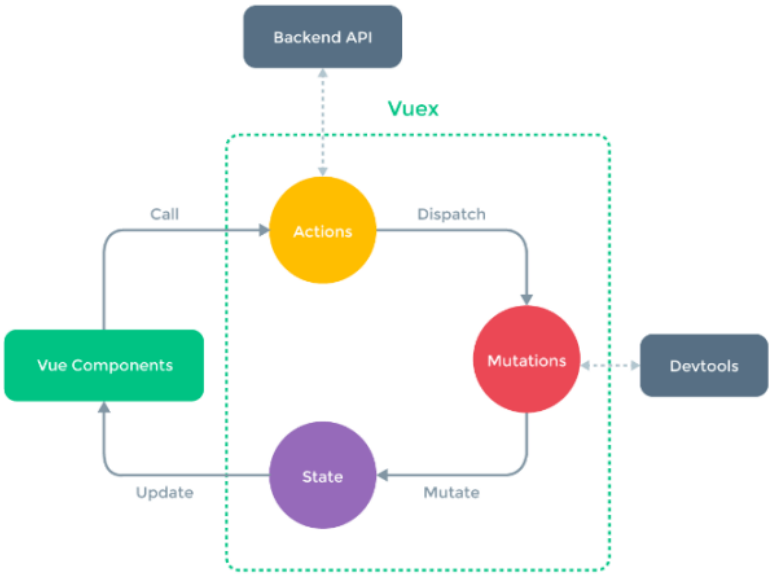


Figure 17. How Vuex manages state (Vuex, n. d.)

Vuex 'store' consists of four parts:

- State contains the data structures and variables which can be imported to any of the application components
- Actions are asynchronous functions that contain API calls or calls to synchronous functions called mutations
- Mutations are used to mutate data stored in state. Mutations do not have a return value
- Getters are functions which are used only to retrieve data out of state

Vuex becomes handy when it is necessary to share functions and data between several components. However, it also increases the complexity of the application. Using Vuex does not prevent having component-specific data structures and variables.

4.2.2 Routing

Component routing can be implemented using 'vue-router' which is the official router for Vue. vue-router provides nested mapping for views and a modular component-based

router configuration. Implementing routes for components makes them URL accessible. (You, E. 2019d).

Figure 18. Creating routes using vue-router

```
import Router from 'vue-router';

Vue.use(Router);

const router = new Router({
    routes: [
        {
          path: '/notfound',
          name: 'NotFound',
          component: () => import('@/components/NotFound.vue'),
        },
        // Example of dynamic route matching
        {
          path: '/user/:id',
          component: () => import('@/components/Settings/User.vue'),
          props: true,
        },
    ]
});

export default router;
```

vue-router can be installed using npm. A folder called ‘router’ is created in the application root folder which has a file called index.js. This file contains the router object where different routes can be defined for components thus making them accessible by URL which is a reference to a web resource. Components may also be lazy loaded which means that the component is loaded to memory only when the component-specific route is accessed. To do this in a Vue application, the component can be imported using a dynamic import syntax described in Figure 18.

Figure 19. Listening to route changes

```
router.beforeEach((to, from, next) => {
  if (!localStorage.getItem('token') && !store.state.auth.token && to.path !== '/login') {
    next('/login');
  } else if (!to.matched.length && to.path === '/') {
    next('/od/map');
  } else if (!to.matched.length) {
    next('/notfound');
  } else {
    if (
      !store.state.auth.username &&
      localStorage.getItem('token') &&
      localStorage.getItem('user')
    ) {
      store.commit('authSuccess', {
        username: localStorage.getItem('user'),
        token: localStorage.getItem('token'),
      });
      store.dispatch('getCurrentUser');
    }
    next();
  }
});
```

vue-router can be set to listen to route changes using its ‘beforeEach’ function so that appropriate action can be taken when a specific route is accessed. A route change happens when the user tries to access a route such as ‘/login’ and is triggered using the ‘next’ function. An example of listening for route changes using vue-router’s ‘beforeEach’ function is provided in Figure 19 where the following checks and actions occur:

- If a token is not found the user is redirected to the login page
- If the user has a token and the route change matches an existing route the user is allowed to access the route
- If the route change is not matched with an existing route then the user is routed to the ‘/notfound’ route that could contain a notification such as ‘404, page not found’

4.2.3 Environment variables

Vue supports the use of environment variables. These are specific files that can be used for build management. Builds are compiled JavaScript modules. In essence a module is just a file.

For instance, the production and staging builds may utilize a different API which has an API specific URL. If the URL is defined using an environment variable, it is possible to define a different API URL for each build.

Figure 20. Binding environment variables in package.json

```
"scripts": {
  "dev": "vue-cli-service serve --mode dev",
  "build:prod": "vue-cli-service build --mode prod",
  "build:staging": "vue-cli-service build --mode staging",
},
```

Figure 20 contains an example of binding environment variables in ‘package.json’. By binding ‘build:prod’ to ‘vue-cli-service build –mode prod’ in the scripts object of the ‘package.json’ file we can now initiate a production build with the configuration set by the production-specific environment variable by writing ‘npm run build:prod’ in the CLI. ‘prod’ refers to ‘.env.prod’ which is a file and an environment variable. The CLI will set up a production build where all the compiled modules are gathered to a folder named ‘dist’ which can be deployed to a file server.

4.3 Vuetify

Vuetify is a material design component framework for Vue which can be installed using the Vue CLI. Material design is a definition for qualities that may be expressed using UI regions, surfaces and components. (Material Design, 2019).

Vuetify provides components that implement material design guidelines and offers decent cross-browser support and an easy to learn component API where each component such as a button or a toolbar has an API which can be utilized to customize the component. (Vuetify, 2019a). Vuetify can be added to the project using the command-line, navigating to the project folder and running ‘vue add vuetify’.

Figure 21. Utilizing Vuetify in a Vue component

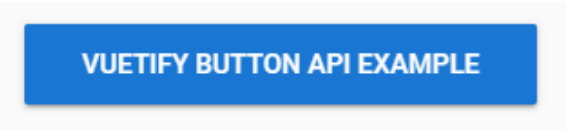
```
<template>
  <v-app>
    <!-- Vuetify components -->
  </v-app>
</template>

<script>
  export default {
    name: 'VueVuetifyExample'
  }
</script>
```

After adding Vuetify to the project Vuetify components are ready to be utilized using the 'v-component' syntax. This means that Vuetify component names start with the 'v-' prefix and can be added to the HTML code section of a Vue component. All Vuetify elements must be created within the 'v-app' tags as shown in Figure 21.

Figure 22. Utilizing the component API of a Vuetify button

```
<v-btn large color="primary">
  Vuetify button API example
</v-btn>
```



The component API of Vuetify allows for rapid development once the developer is moderately familiar with the framework. However, the components provide limited customization options and applications developed with Vuetify will look alike to some degree. An example of utilizing the component API of a Vuetify button is described in Figure 22 where the size and color of a button is defined using the button's API.

Due to the rise of smartphones, responsivity is required from modern web applications. Responsive applications are applications designed to work well on all screen-sizes. Some elements such as Vuetify's data tables are naturally responsive and Vuetify offers material design viewport breakpoints that can be attached to Vuetify components. These breakpoints are JavaScript helper classes and may be used to hide and show the layout components based on the size of the user's screen. They have aliases such as 'md' which stands for the screen size of medium. An example of the use of these breakpoints is demonstrated in Figure 23. (Vuetify, 2019b).

Figure 23. Using a breakpoint to hide a component

```
<v-btn class="hidden-md-and-down" large color="primary">
  Vuetify material design viewport breakpoint example
</v-btn>
```

By combining a front-end framework such as Vue with a material component framework like Vuetify an application developer is equipped with tools to efficiently develop user interfaces for modern web applications.

5 PLATFORM USER INTERFACE

The practical part of thesis documents Analytics UI which is a web user interface developed using Vue and Vuetify. Analytics UI is the work of the writer of this thesis as a whole. This chapter will discuss the main use-cases the application was created for and provide an overview of the main application features.

5.1 Use-cases for the application

Analytics UI was developed to answer various use-cases related to industrial condition monitoring and tracking applications. These use-cases include:

- Measurement of environmental attributes such as temperature and humidity. For instance, monitoring employee working conditions or the drying of cement
- Sudden changes in environmental attributes. For instance, noticing such changes could prevent an accident or the breakdown of a machine
- Tracking of assets such as persons, vehicles or devices. For example, this could be used to optimize routes, the use of personnel and prevent theft
- Monitoring device and vehicle use-times to optimize the number of vehicles in use. For example, construction companies often rent vehicles. Having unused vehicles would increase a company's monetary expenses

Condition monitoring related use-cases also require various configuration options and settings offered by Analytics UI.

Analytics UI includes visualizations for various types of object recognition-based camera analytics for improving customer service processes. These are chart visualizations that have their own section in the UI. These include:

- Customer counting and speed of service calculation for fast food restaurants. The data provided by this feature can be used by restaurant owners to optimize the number of employees so the greatest number of customers can be served
- Customer counting for shopping malls and stores

5.2 Main application features

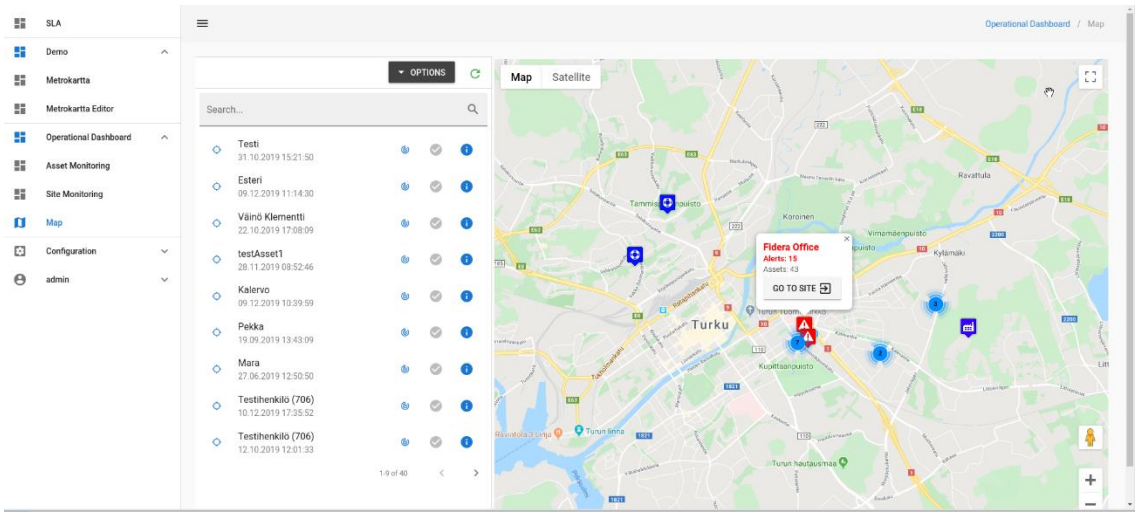


Figure 24. Operational Dashboard of Analytics UI

Analytics UI has a view called Operational Dashboard and it contains the most important application features related to the condition monitoring use-cases discussed in Chapter 5.1. While various types of devices are supported by the platform back-end which provides the data for the user interface this chapter will discuss sensor devices for the sake of simplicity.

Features related to condition monitoring are visualizations of sensor measurement data related to asset use times and attributes such as temperature and humidity. These visualizations include:

- Bar chart visualizations of asset use times
- Line chart visualizations of sensor measurements

Sensors can be configured, and attribute related limits set so that alerts are generated when a set limit is breached. This enables the user to acknowledge and process a sensor alert and to take whatever physical actions are necessary by the changed conditions.

Sites, assets and sensors are visualized on an instance of Google Maps shown in Figure 24. This includes:

- Acknowledging alerts
- Viewing locations of sites, assets, sensors and their status. Sensors may be attached to assets or set to stationary locations

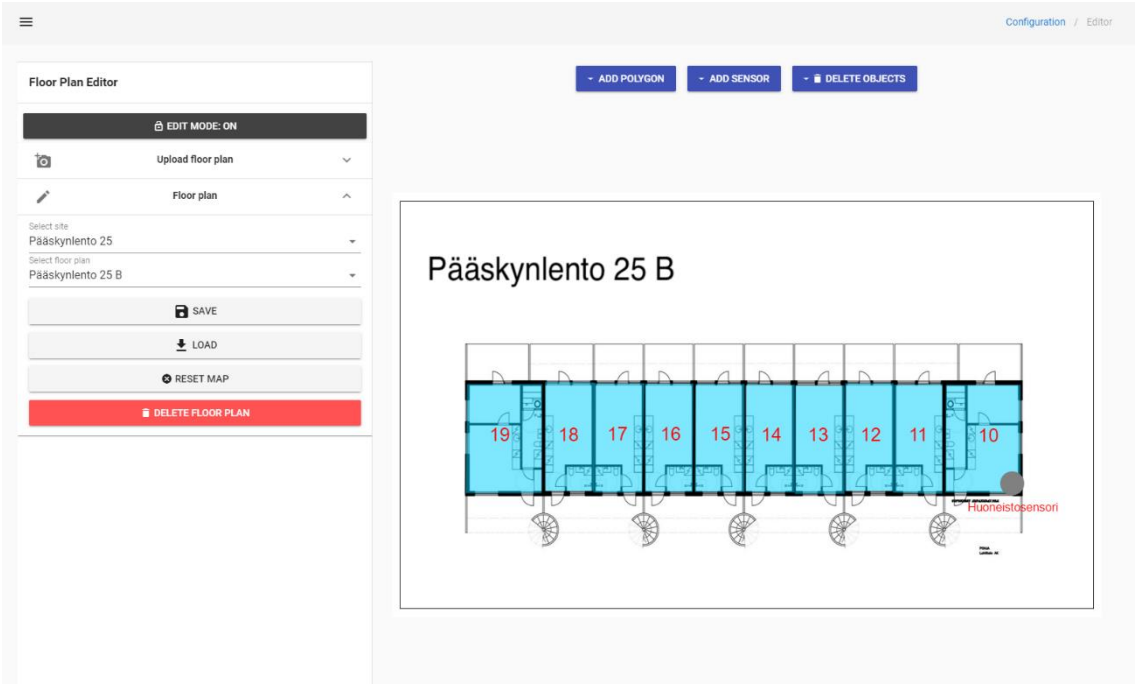


Figure 25. Managing site floor plan images

Assets and sensors are also visualized on an interactive site-specific floor plan image which provides more accurate information about a site such as a building which cannot be visualized accurately enough on a map. Figure 25 is an image of the interactive floor plan which includes features such as:

- Uploading site floor plan images of standard image formats
- Creating and saving areas on the image where sensors can be attached so the following features are possible:
    - Visualization of areas with activity such as an active sensor
    - Active alerts

The map and floor plan visualizations of alerts and sensors are one option for visualizing this information. List-based views are also provided.

5.3 Configuration options and settings

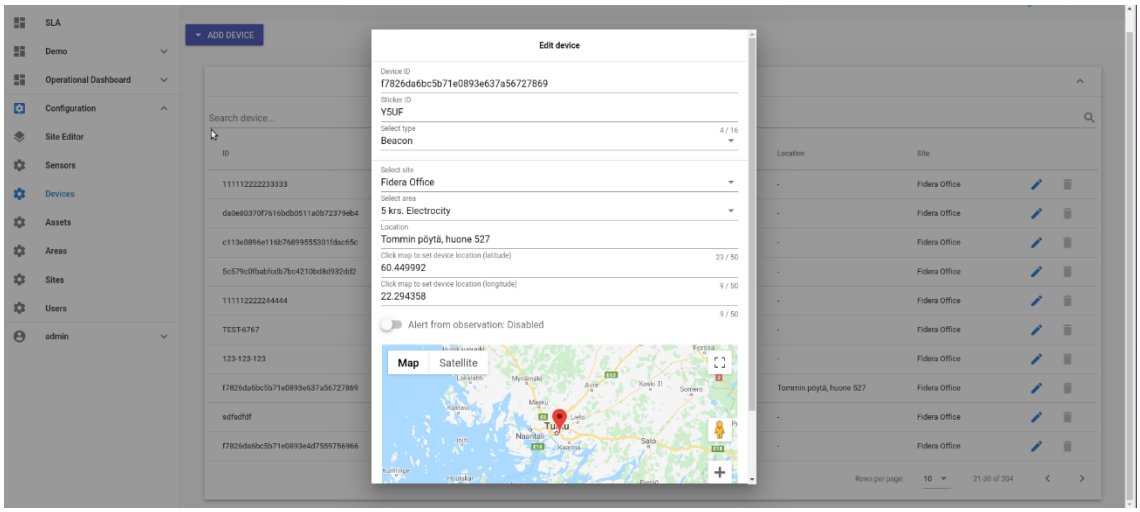


Figure 26. Device configuration

Features described in the previous chapter are based on devices such as sensors. These devices need to be configured before they are functional. The devices also need to be taken to the physical location they are intended to provide measurement data from. Figure 26 is an image of the configuration menu where configuration options are offered for:

- Sensors which includes:
  - Adding sensors to the system
  - Setup of sensor attribute alert limits
  - Formation of sensor groups
  - Attaching sensors to assets
  - Setup of static device GPS locations
- Support for other device types such as Bluetooth beacons with device type specific configuration options
- Assets such as:
  - Persons, vehicles or employees with basic descriptive fields such as name, e-mail, phone number and description
- Sites that have:
  - Name and a GPS location. Sensors and devices generally belong to sites and are located on site related areas
- Areas

These discussed configuration options contain features for CRUD operations. Features for user management are included and they contain features for managing user accounts. These are:

- Basic user account management such as changing the user's password
- Admin users have the option of managing sites, creating and editing user accounts with site-related rights

By utilizing these configuration options and settings, Analytics UI can be used to create user accounts, insert devices to the system and configure them ready for use.

CONCLUSION

This thesis dealt with a real-life work development task where a web user interface was built with features for industrial condition monitoring and asset tracking use-cases. The UI in question is part of an IoT platform which connects IoT devices and others to services with the purpose of providing data for applications such as web user interfaces.

The theoretical part of this thesis demonstrates how specific back-end and database frameworks can be applied to implement the server applications and databases of a scalable IoT platform. The theoretical part also documents how the front-end framework VueJS can be applied together with a material design framework called Vuetify. The practical part of this thesis documents the main features of the platform's UI which has been implemented using the technologies described in the theoretical part of this thesis. The discussed UI contains features such as Google Maps based tracking of assets with sensors and other devices, indoor tracking, sensor-based condition monitoring and vast device configuration and user management options. The UI in question is titled Analytics UI and it is the work of the writer of this thesis as a whole.

It can be concluded that with the proper utilization of back-end and front-end frameworks web applications may be built fast. Utilizing frameworks such as LoopBack and VueJS save development time as they contain ready-made implementations that can be used to build APIs and user interfaces efficiently. While having a good understanding of programming in general helps in the utilization of these frameworks a lot of time is spent learning the frameworks inner workings such as VueJS's component-based architecture or the utilization of LoopBack's various classes. Each of these frameworks would have enough topics in itself for a thesis so they could not be discussed in-depth, instead this thesis provides a comprehensive overview of their use with hands-on examples in the making of an IoT platform.

Since this thesis is limited in detail but gives an overview of an IoT platform and its essential parts, a natural extension of this work would be to discuss each part separately and in more detail. It should be noted that the implementation of an IoT platform is by no means limited to the technologies presented in this thesis, but they serve as an example of an implementation. The presented technologies may be used in server, database and user interface implementations whether they are or are not IoT related. If it is disputed whether the implementations presented in this thesis are applicable, it should be known that the discussed IoT platform is currently used by customers, some of which represent large and well-known companies. The IoT platform was developed at Fidera Ltd.

REFERENCES

Statista. (2016). Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions). (Referenced on 17.11.2019). https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

TechTarget. (n. d.). Internet of Things. (Referenced on 17.11.2019). https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

Uppa, J. (2017). Sensorit, laitteet ja aktuaattorit. (Referenced on 1.11.2019). https://www.theseus.fi/bitstream/handle/10024/138552/Uppa_Jari.pdf?sequence=1&isAllowed=y

Perry, M. (2016). Evaluating and choosing an IoT platform. (Referenced on 6.11.2019). https://www.oreilly.com/learning/evaluating-choosing-iot-platform

LoopBack. (2019a). Crafting LoopBack 4. (Referenced on 9.11.2019). https://loopback.io/doc/en/lb4/Crafting-LoopBack-4.html

OpenAPI Initiative. (2019). OpenAPI specification. (Referenced on 9.11.2019). http://spec.openapis.org/oas/v3.0.2#openapi-document

LoopBack. (2019b). Key concepts. (Referenced on 9.11.2019). https://loopback.io/doc/en/lb4/Concepts.html

LoopBack. (2019c). Model. (Referenced on 9.11.2019). https://loopback.io/doc/en/lb4/Model.html

LoopBack. (2019d). Datasource. (Referenced on 9.11.2019). https://loopback.io/doc/en/lb4/DataSources.html

LoopBack. (2019e). Add a repository. (Referenced on 10.11.2019). https://loopback.io/doc/en/lb4/todo-tutorial-repository.html

LoopBack. (2019f). Controllers. (Referenced on 10.11.2019). https://loopback.io/doc/en/lb4/todo-tutorial-controller.html

LoopBack. (2019g). Services. (Referenced on 7.12.2019). https://loopback.io/doc/en/lb4/Services.html#overview

PostgreSQL. (2019a). What is PostgreSQL. (Referenced on 9.9.2019) https://www.postgresql.org/about/

PostgreSQL Wiki. (2019). Quickstart. (Referenced on 17.11.2019). https://wiki.postgresql.org/wiki/Apt

Timescale, Inc. (2019a). What is time-series data. (Referenced on 3.8.2019). https://docs.timescale.com/latest/introduction/time-series-data

Timescale, Inc. (2019b). time_bucket(). (Referenced on 3.11.2019). https://docs.timescale.com/latest/api#time_bucket

PostGIS. (n.d. a). About PostGIS. (Referenced on 4.8.2019). https://postgis.net/

PostGIS. (n.d. b). Description. (Referenced on 4.8.2019). https://postgis.net/docs/ST_SRID.html

OGC. (2019). Spatial referencing by coordinates (ISO 19111:2019). (Referenced on 4.8.2019). https://www.opengeospatial.org/docs/as

You, E. (2019a). What is Vue.js. (Referenced on 4.8.2019). https://vuejs.org/v2/guide/

You, E. (2019b). Components of the system. (Referenced on 5.7.2019). https://cli.vuejs.org/guide/#components-of-the-system

Shapiro, D. 2016. Understanding component based architecture. (Referenced on 23.10.2019). https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238

W3C. (2005). What is the Document Object Model. (Referenced on 5.7.2019). https://www.w3.org/DOM/#what

You, E. (2019c). Instance Lifecycle Hooks. (Referenced on 6.7.2019). https://vuejs.org/v2/guide/instance.html#Instance-Lifecycle-Hooks

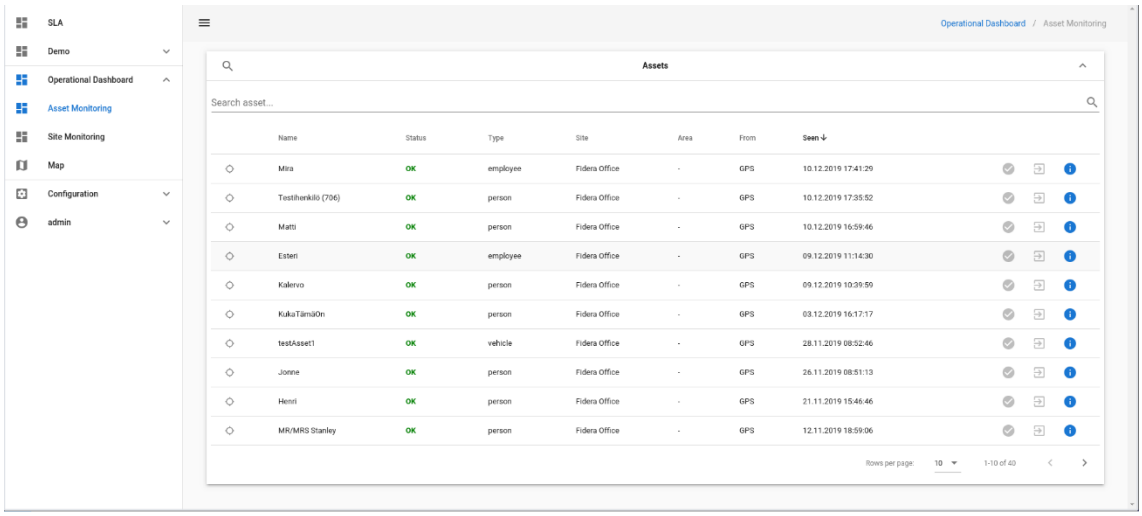
Vuex, (n. d). What is Vuex. (Referenced on 6.7.2019). https://vuex.vuejs.org/

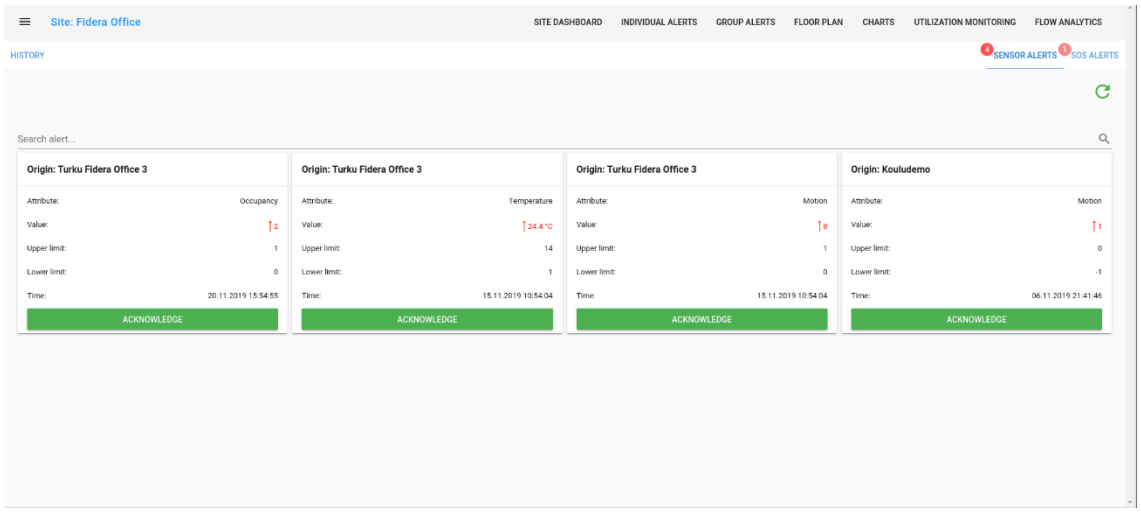
You, E. (2019d). Introduction. (Referenced on 6.8.2019). https://github.com/vuejs/vue-router

Material Design. (2019). Introduction. (Referenced on 1.11.2019). https://material.io/design/introduction/#

Vuetify. (2019a). What’s the difference. (Referenced on 5.7). https://vuetifyjs.com/en/introduction/why-vuetify

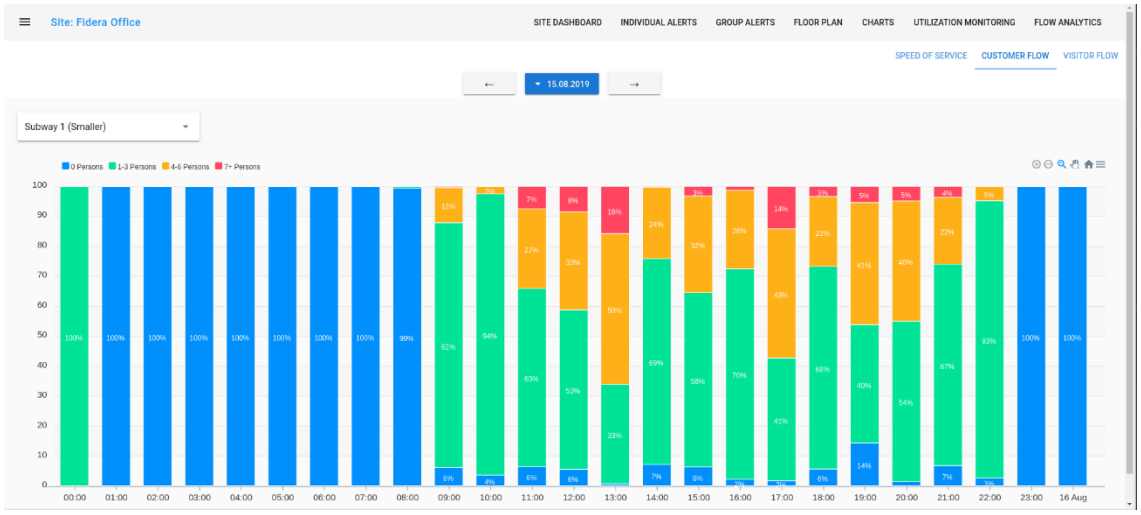
Vuetify. (2019b). Breakpoints. (Referenced on 5.7.2019). https://vuetifyjs.com/en/customization/breakpoints


SLA
Demo
Operational Dashboard
Asset Monitoring
Site Monitoring
Map
Configuration
admin
Operational Dashboard / Asset Monitoring
Assets
Search asset...
Name Status Type Site Area From Seen
Mira OK employee Fidera Office - GPS 10.12.2019 17:41:29
Testihenkilö (706) OK person Fidera Office - GPS 10.12.2019 17:35:52
Matti OK person Fidera Office - GPS 10.12.2019 16:59:46
Esteri OK employee Fidera Office - GPS 09.12.2019 11:14:30
Kalervo OK person Fidera Office - GPS 09.12.2019 10:39:59
KukaTämäOn OK person Fidera Office - GPS 03.12.2019 16:17:17
testAsset1 OK vehicle Fidera Office - GPS 28.11.2019 08:52:46
Jonne OK person Fidera Office - GPS 26.11.2019 08:51:13
Henri OK person Fidera Office - GPS 21.11.2019 15:46:46
MR/MRS Stanley OK person Fidera Office - GPS 12.11.2019 18:59:06
Rows per page: 10
1-10 of 40

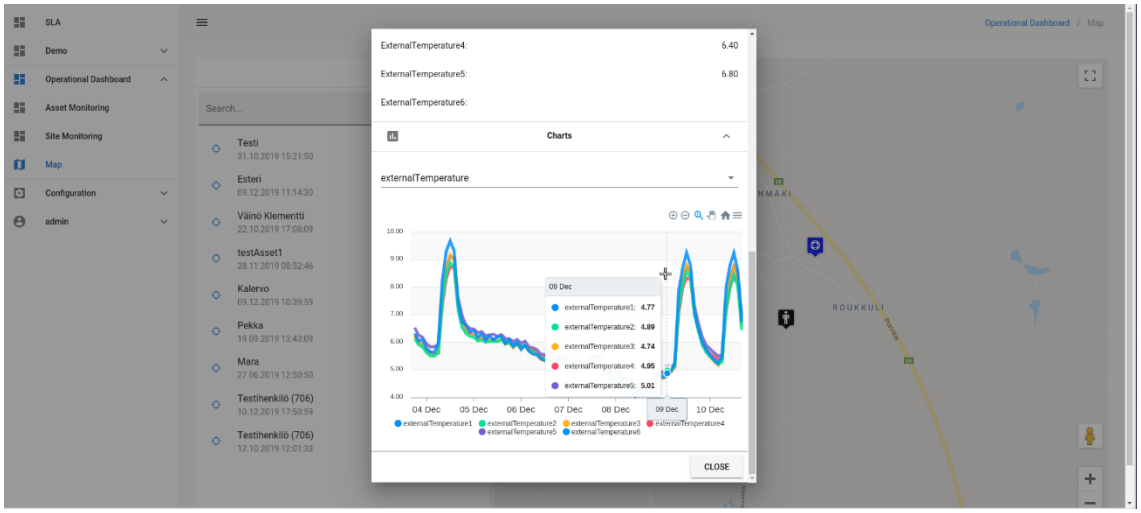

Site: Fidera Office
SITE DASHBOARD
INDIVIDUAL ALERTS
GROUP ALERTS
FLOOR PLAN
CHARTS
UTILIZATION MONITORING
FLOW ANALYTICS
HISTORY
SENSOR ALERTS
SOS ALERTS
Search alert...
Origin: Turku Fidera Office 3
Attribute: Occupancy
Value: 2
Upper limit: 1
Lower limit: 0
Time: 20.11.2019 15:54:55
ACKNOWLEDGE
Origin: Turku Fidera Office 3
Attribute: Temperature
Value: 24.4 °C
Upper limit: 14
Lower limit: 1
Time: 15.11.2019 10:54:04
ACKNOWLEDGE
Origin: Turku Fidera Office 3
Attribute: Motion
Value: 8
Upper limit: 1
Lower limit: 0
Time: 15.11.2019 10:54:04
ACKNOWLEDGE
Origin: Kouludemo
Attribute: Motion
Value: 1
Upper limit: 0
Lower limit: -1
Time: 06.11.2019 21:41:46
ACKNOWLEDGE


Site: Fidera Office
SITE DASHBOARD
INDIVIDUAL ALERTS
GROUP ALERTS
FLOOR PLAN
CHARTS
UTILIZATION MONITORING
FLOW ANALYTICS
DAY WEEK MONTH 3 MONTHS 6 MONTHS YEAR
08.11.2019 - 08.12.2019
RESET
SCALING
LAST MINIMUM MAXIMUM AVERAGE
Tilakäyttö, Fidera TKU-toimisto
Temperature
26.00
25.00
24.00
23.00
22.00
21.00
20.00
09 Nov 10 Nov 11 Nov 12 Nov 13 Nov 14 Nov 15 Nov 16 Nov 17 Nov 18 Nov 19 Nov 20 Nov 21 Nov 22 Nov 23 Nov 24 Nov 25 Nov 26 Nov 27 Nov 28 Nov 29 Nov 30 Nov Dec '19 02 Dec 03 Dec 04 Dec 05 Dec 06 Dec 07 Dec 08 Dec
Turku Fidera Office 2
Turku Fidera Office 3
Humidity

Appendix 1 (2)


Site: Fidera Office
SITE DASHBOARD
INDIVIDUAL ALERTS
GROUP ALERTS
FLOOR PLAN
CHARTS
UTILIZATION MONITORING
FLOW ANALYTICS
SPEED OF SERVICE
CUSTOMER FLOW
VISITOR FLOW
15.08.2019
Subway 1 (Smaller)
0 Persons
1-3 Persons
4-6 Persons
7+ Persons


Site: Fidera Office
SITE DASHBOARD
INDIVIDUAL ALERTS
GROUP ALERTS
FLOOR PLAN
CHARTS
UTILIZATION MONITORING
FLOW ANALYTICS
SPEED OF SERVICE
CUSTOMER FLOW
VISITOR FLOW
DAY
WEEK
MONTH
07.12.2019
07.12.2019
RESET
Kamppi
inside
outside


SLA
Demo
Operational Dashboard
Asset Monitoring
Site Monitoring
Map
Configuration
admin
Operational Dashboard / Map
ExternalTemperature4: 6.40
ExternalTemperature5: 6.80
ExternalTemperature6:
Charts
externalTemperature
CLOSE