Evaluation of FIWARE's Business API Ecosystem

MICHAEL JIVUNG
CARL TIDELIUS
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Evaluation of FIWARE's Business API Ecosystem

Michael Jivung
dat13mji@student.lu.se

Carl Tidelius
dat12cti@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Emma Fitzgerald

Examiner: Christian Nyberg

March 13, 2019

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

The modern society is getting more and more digitalized and the concept Internet of Things is growing. The data being collected and digital services require a place to be accessed and sold, thus setting the demand for a new kind of marketplace; an online IoT focused marketplace. The large EU supported open source platform for Smart Solutions, FIWARE, has created such a marketplace, namely the Business API Ecosystem (BAE). This paper aims to evaluate and review the BAE, giving a quality evaluation as well as comparing it to contemporary alternatives. This will help interested parties to make an educated decision when choosing an IoT marketplace.

The evaluation was done with a hands-on approach where the BAE was explored. The program was looked at from the perspective of both the administrator and the users. It was installed, configured, tested and exploratory used. A custom front-end was created to investigate its modifiability and the usability of its APIs. Performance testing of the APIs was also done.

The results of the evaluation shows that the Business API Ecosystem is a viable option for an IoT marketplace of high quality. However, it is a relatively young and small project that which, without the use of plugins and other software, lacks some much wanted IoT related functionality.

Preface

This is the final report for a master thesis project at the Electrical and Information Technology (IET) institution of the Faculty of Engineering in Lund. The project was done by the students Michael Jivung and Carl Tidelius who studied their final year at Information- och Kommunikationsteknik and Datateknik, respectively. The project was carried out i the second half of 2018 from September to January. The project was defined by the Lund based and IoT related company Sensative as they wanted help exploring new territory in the IoT field. The majority of the work was done at the office of Sensative with the help of the employee Ulrik Sjölin as supervisor. Emma Fitzgerald, researcher at the EIT institution, assisted as main supervisor and Christian Nyberg, university lecturer at the EIT institution was the project examiner.

We would like to thank Ulrik, Emma and Christian for all the valuable help they gave us as well as Sensative for the opportunity.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

Marketplaces have been an integrated part of society for centuries, where peers could offer and acquire different goods and services. Throughout the centuries, these marketplaces have evolved in different ways and the economy has flourished with them. As the internet emerged, marketplaces gained new ground to evolve further and they have become part of one of the largest industries in the world.

You have probably already been exposed to many different digital marketplaces, such as the App store or eBay, but these marketplaces focus on applications or physical objects. In our society, where data can be gathered through a myriad of things, a new marketplace is emerging. In integration with the Internet of Things (IoT), a marketplace for both services and data can be achieved and used by both organizations and private citizens. Private citizen can utilize the concept Sensing as a Service [22], to sell data collected by their sensors and benefit from data which otherwise would have been used only by them, and most likely only one time.

The exchange of data is becoming more and more predominant, especially with regards to IoT. The Internet of Things refers to both everyday devices and more sophisticated machinery with a built in connection to the internet. Through this, the devices can communicate over a network and send data to be processed by external systems. A lot of this data is only processed once and often only by the owner of the sensor or device. After the data has been used it is either discarded or put into databases for storing, often never to be used again. This results in waste of data, data that could be useful and valuable to other parties. This isolated storage of data is often called *data silos*. IoT marketplaces seek to diminish this waste.

An emerging new concept for society is the Smart City, where the city itself makes use of these smart IoT devices. Some Smart Cities have already emerged and are now pushing for a common standard for how to create a software platform for Smart Cities. One of these standards is called FIWARE which is an open platform offering generic enablers [16] containing open APIs to facilitate development of future internet applications. Another proposal which interested parties are pushing for is a marketplace for the data which the IoT devices are collecting.

Sensative, our Master Thesis employers, have decided to implement FIWARE into their own platform *Yggio*. Yggio is a horizontal, thin IoT platform designed

for Smart Properties, Smart Cities and other arbitrary IoT applications. Its purpose is roughly to act as a data broker between sensors/actuators and users/service providers. Sensative has therefore enlisted our services to investigate the possibility to utilize the open source capabilities of the FIWARE standard to create a Smart Marketplace. FIWARE has designed, together with TMForum, a digital marketplace called *Business API Ecosystem* (BAE). The main goal of this Master thesis was to evaluate and review this marketplace so that Sensative can decide on whether or not they want to integrate it in their platform.

1.1 Questions

Before we started the project we put together a series of questions which we would like to have answered by the end of the project. The questions are answered throughout the report, mostly in Chapter 5.

- Why are we evaluating the BAE instead of other alternatives?
- How complete is the BAE as an IoT-marketplace? Does it lack any desired functionality?
- How do other IoT-marketplace alternatives compare to the BAE?
- What kind of assets are possible to sell/buy on the marketplace?
- How does the delivery of the purchased products work?
- What are the supported payment methods? Is it possible to add arbitrary payment methods?
- How modifiable and extendable is the BAE? Can you create plugins/add-ons?
- How do legal aspects such as right of withdrawal of purchases work?
- How easy is the BAE to install and maintain?
- How easy is the BAE to use?
- What does the architecture of the BAE look like?
- How good is the performance of the BAE?
- What are the licenses for using the BAE?
- How popular is the BAE today? How active is the Github repository?

The architecture of the BAE presents a few more questions, such as modularity, object orientation, dependencies, and so on. This will be investigated more thoroughly throughout the report and as such the questions posed here do not go into details of the architecture.

Since this work has been done on our own computers, some limitations in software have posed a problem. As such we have proposed further investigations and work in the discussion of our results.

1.2 Structure of the report

This report starts off with related work in Chapter 2, Section 2.1 where we give the reader background information on FIWARE and the BAE and other software

that were needed during the project. This is followed by an in depth technical background of the BAE in Chapter 2, Section 2.2; what it consists of and how it works. In Chapter 3 we explain how we installed the BAE, how we built the React front-end and how performance testing and functional testing of the BAE were carried out. In Chapter 4 the results of the testing is shown. In Chapter 5 we evaluate the BAE and discuss it's strengths and weaknesses. We end the report with a conclusion in Chapter 6 in which we have concluded that the BAE is indeed a viable option as an IoT marketplace although it is lacking in certain areas.

Chapter 2
Background

In this section background information that will be useful to have in the later chapters is given. IoT, digital marketplaces, FIWARE and the BAE are presented, followed by a more in depth technical background of the BAE. Other software that was used during the project is also presented, such as CKAN, Docker and the IdM.

2.1 Related work

2.1.1 Internet of Things and marketplaces

As of now, the primary market of data marketplaces is corporations and governments, since the quantities of data are usually large, but the market has been shifting towards business to consumer [25][27][26]. This means that data is mostly being exchanged between companies, but according to Gartner, Inc,[13] 80 percent of companies will fail to monetize their IoT data. Since there already exist data markets, it is surprising that so many companies will fail to monetize their data, but there is a reason. In [28], it is theorized that since users have access to such a large amount of free information, they have a low willingness to pay for data. It is not unreasonable to think that people will not think of the idea of monetizing their data, since they themselves would not be willing to buy data.

As mentioned, there already exist a few data marketplaces on the internet, such as BDEX[1] and Saleforce's data.com[4], and while they allow for the monetization of data, IoT data have some unique characteristics, which demand a more specific marketplace [17]. Firstly, the data that come from sensors owned by private citizens is not a large amount. Usually the owners only have a couple of sensors. Traditional data markets are designed for companies or governments, which means that the data will come from a large number of independent device owners. Secondly, the measurements that are taken from the sensors are not as precise when handling large amounts of data, that is, they are usually stored in archives and not updated in real time. The main reason to use sensors is to collect data and present it as it is. For instance, an owner of a temperature sensor would in most cases not benefit from seeing yesterdays data, and traffic map creators would demand data from connected cars that can be directly transformed into a real time traffic map to be able to notify driver about approaching traffic. The

point is that there exist scenarios where data is needed with a delay of only minutes or seconds. Thirdly, since data measurements are often purchased via subscription or in advance, and the current data marketplaces list data already collected, data consumers will have to sift through data offerings to find the ones that are listed to be updated at a certain time, and then purchase the data at the specified time. Lastly, IoT devices can collect sensitive information which would require that sensor owners would have to understand exactly what data they are selling and to whom. With this in mind, a marketplace specially defined for IoT-devices is needed, especially in larger scales such as Smart Cities.

IoT sensors are being massively produced around the world and according to Robert Bogue[24] it will reach a trillion devices in the coming decades. IoT will also have an annual economic impact of 3.9 - 11.1 trillion dollars in several different settings such as cities, retail environments and factories [19]. Although the market in itself will have a tremendous impact, the IoT adoption is slow. As such, to facilitate the adaptation to IoT, is is important that standards are implemented in the different settings to avoid future problems.

2.1.2 FIWARE

FIWARE is a framework of open source platform components, that together with third-party platform components, aims to accelerate the development of smart solutions [10]. It is an EU-supported project started in 2011, which consists of an open community of actors that contribute to the project. The FIWARE framework can be used by third-parties such as companies, organizations and cities in smart applications which can be in the domains of smart cities, smart industry, smart logistics etc.

The FIWARE framework is component based which means that one can pick and choose the parts that fit one's needs. The central part of the framework is the FIWARE Context Broker which is the only mandatory component. It handles context information, i.e. data, which is central in any smart solution. Additional FIWARE components or third-party components can then be built around the Context Broker. If a third-party platform uses the Context Broker it can label itself *Powered by FIWARE*. The Business API Ecosystem is however not part of the FIWARE standard open platform, and as such, does not require the context broker.

2.1.3 Business API Ecosystem

The Business API Ecosystem (BAE) [5] consists of the FIWARE Business Framework and a set of standard APIs provided by TMForum. The component allows for monetization of assets, both digital and physical, in the form of charging, accounting, revenue settlement and sharing. This component offers managing, publishing, and revenue generating possibilities to sellers of products, apps, data and services. The BAE was created and is maintained mainly by Francisco de la Vega, Senior Research Engineer at the Universidad Politécnica de Madrid.

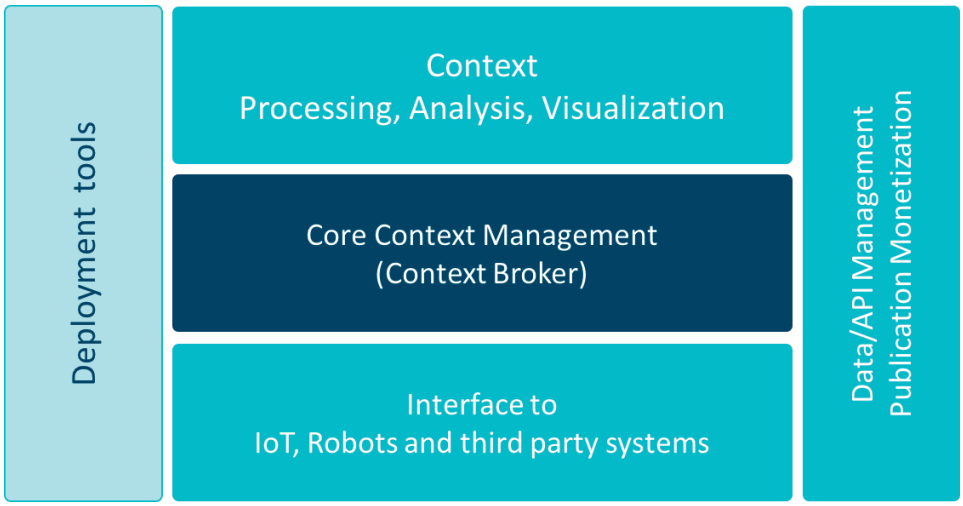

Figure 2.1: The architecture of the FIWARE framework.

Since there are many different functionalities provided, there also exists an established model for controlling users' privileges and interactions of the BAE. The component that is in charge of these roles and privileges is not the BAE. Instead the Identity Manager (IDM) takes care of this, and provides the means of controlling the users' privileges via roles. Users must at least have one role, but may have any number above that. The roles that provide privileges are as follows:

- Seller: Has the option of publishing offerings for others to acquire.
- Customer: Has the option of acquiring offerings. This is also the default role.
- Administrator: Manages certain settings, and can create categories.

The official documentation states that the main functionalities that the BAE offers are:

- Publication of new products and offerings.
- Acquisition of products and offerings.
- Offering payment.
- Billing management of acquired offerings.
- Access to all purchased services.
- Download of software if the offering is part of a downloadable service.
- Shipment tracking if the offering describes a physical product.
- Distribution of revenues between all the involved stakeholders.
- Management of service usage information in order to enable usage-based models.

Usage Scenarios

To follow the blueprint of the API's structure of relationships, the creation of an offering requires a catalog and a product specification. The catalog has to be created to publish the offerings, while the specification has to be created for the offerings to be able to offer an actual product or a service. So, the first step is to create the catalog, which is straightforward. The user would have to be assigned the role *Seller*, to be able to access the section *My Stock*. From there, the user can create a catalog. This requires the user to fill in a Name for the catalog, and an optional description of what the catalog contains. After this, the user would have to edit the catalog's status to *Launched* to be able to use the catalog. After a catalog has been created, the user can create a product specification. This specification details the information of the product to be sold. In figure 2.2 the steps to create a product specification can be seen.

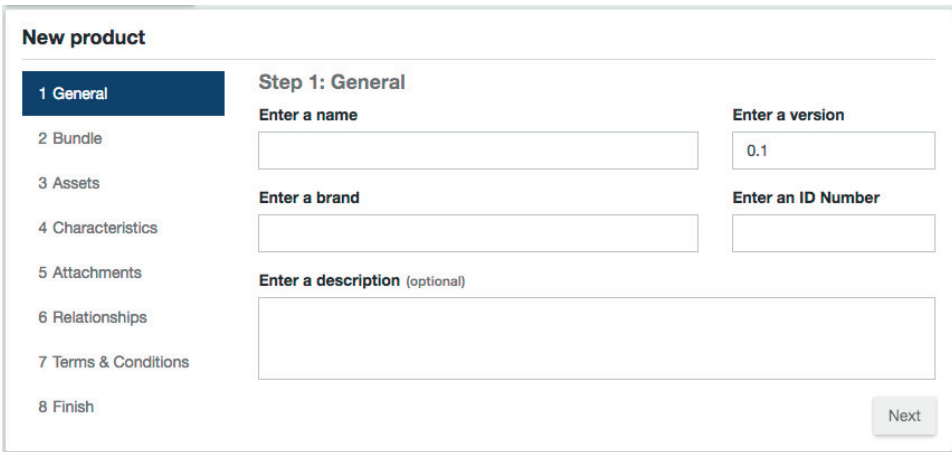

Figure 2.2: Creation of a product specification.

After creating a product specification with the preferred data, an offering can be created. An offering can either be defined by a bundle of offerings, or by a product specification, but not by both. It must then be put in a catalog and optionally a category. Price plans can be created if the seller wishes the product to have a cost, and lastly a Revenue Sharing plan must be selected, even if there is not a configured price (e.g. free products). This process can be seen in Figure 2.3.

After a product has been created and launched, as done in the product specification and catalogue creation, the product can be acquired by users other than the creator of the offering.

When a user with the Customer role is logged in on the platform, that user can purchase the offered products. The user can filter the products either by searching by the type of products associated with the offering or by selecting a catalog. The user can then select the offering he/she wishes to acquire and put it in the

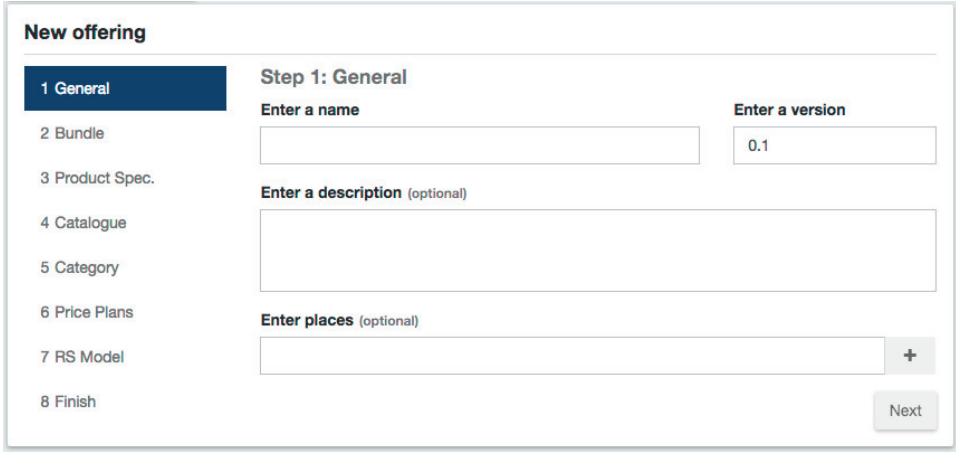

Figure 2.3: Creation of offering.

shopping cart, then confirm and checkout. In order to do this the user must have a shipping address configured in the BAE and a PayPal account associated with the email-address used when signing up. The BAE will request the information necessary to make the payment and then download the transaction invoice and reroute to the service provider to access credentials from the provider. After the process is completed, the user will own the offering and will be able to download the invoice and get necessary access credentials depending on what type of offering the user has acquired.

2.1.4 Comprehensive Knowledge Archive Network

Comprehensive Knowledge Archive Network (CKAN) is a tool for making open data websites[3]. With it the user can manage and publish collections of data, in CKAN called *datasets*. CKAN consists of a back-end that stores and handles the data as well as a front-end website for users to access the data. The front-end allows users to search published data and download it. The data can also be previewed using maps, graphs and tables. CKAN is commonly used by entities who collect a lot of data, such as governments and research institutions. It is open source and is developed by a community of contributors. It is also modified and extended upon by an even larger community of developers.

2.1.5 Docker

Docker is similar to a virtual machine, with so called containers that host clean setups of databases and programs, like MySQL and the FIWARE components. It is created by Docker, Inc., residing in San Francisco. All docker images/builds are hosted on a repository not unlike Github and can be shared with peers. The key benefit with Docker is that it allows users to package an application with all of its dependencies into a standardized unit for software development. With this, one

can skip all the time consuming manual installations and not have to worry about conflicting pre-installed software. Instead, each package is run in a container which can be linked to other containers and thus setting up an environment which is clean from other conflicting software and with all dependencies installed and linked in the containers. Via Docker, it is possible to install each and every component and link them with APIs and each other with a docker-compose.yml file, which acts like a blueprint for how each container is linked and from where it should fetch images. However to be able to utilize all the features of the BAE, another GE must be installed, the Identity Manager.

2.1.6 Identity Manager

The Identity Manager - KeyRock (IDM) is another FIWARE GE developed by a number of people at the Universidad Politécnica de Madrid. The IDM working in tangent with two other GEs, the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP) to provide authentication and authorization security to services and applications, in our case the BAE [12].
To be able to install and run the IDM there are three software requirements, namely:

- MySQL 5.7
- NodeJS
- npm

The IdM's current main concepts are Users, Organizations, and Applications. Users have a registered account in the IdM and can manage their organizations and register applications. Organizations are a group of users that share resources of an application, that is, roles and permissions. Users can be members or owners of these organizations. Applications work as the client role in the OAuth 2.0 architecture and will request protected user data. Applications are able to authenticate users with ClientID and ClientSecret which identifies the application.

2.2 Technical background

This section will go into further detail of the BAE which is this project's main focus.

2.2.1 Architecture

The FIWARE Business Framework consists of four different components. The following are required for the BAE to be able to run properly:

- **The Business Ecosystem Charging Backend** which provides charging functionality, such as connection to charging platform (PayPal, etc), the required actions to charge the user for the products they are using/acquiring. In addition, it also manages the lifecycle of the digital products, activating them when they are being used/acquired and deactivating them when the

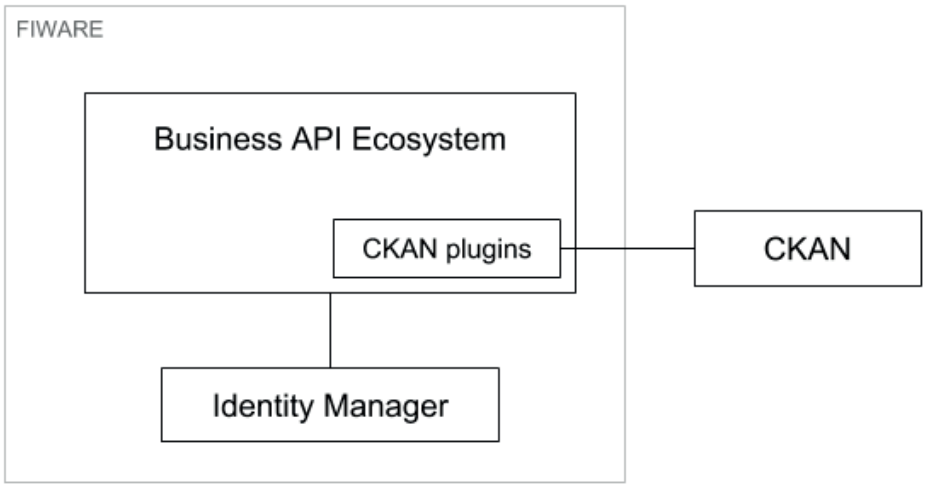

Figure 2.4: Relationship of the used software

users subscription ends or is cancelled. The Charging Backend also handles information about different pricing models, the accounting information, and the revenue sharing reports.

- **The Business Ecosystem RSS** which provides the revenue settlement, so that different stakeholders can distribute the revenues in a flexible way, and share the different revenues derived from the usage of the platform.
- **The Business Ecosystem Logic Proxy** which provides authentication via a back-end, the web portal/front-end and acts as the API orchestrator validating user requests, including authentication, authorization, and the content of the request from a business logic point of view.
- **The TMForum standard APIs** which consists of:
  - the Catalog Management API
  - the Product Ordering Management API
  - the Product Inventory Management API
  - the Party Management API
  - the Customer Management API
  - the Billing Management API
  - the Usage Management API

The dependencies for installing and running the BAE include:

- APIs and RSS:
  - Java
  - Glassfish
  - MySQL

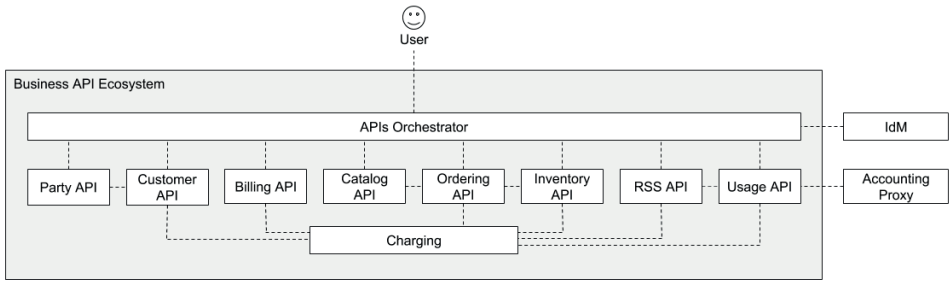

Figure 2.5: The architecture of the BAE.

- Charging Backend:
  - Python
  - MongoDB
  - wkhtmltopdf
- Logic Proxy:
  - Node.js (including npm)

The reader of this paper should be familiar with the majority of this software except for possibly Glassfish, wkhtmltopdf and Node.js. As such, a short description of them will follow. **Glassfish** is an application server for the JAVA EE platform that helps to create web applications and provides a server environment to run them. **Wkhtmltopdf** is a command line tool that can render HTML into PDF. **Node.js** is a JavaScript run-time environment that executes JavaScript code outside of a browser, often used for back-end server applications.

In order to install the BAE one can take two different approaches: manual installation or using Docker. For the manual installation there is a provided installation script that makes the process automated. The BAE installation guide states that the program has only been tested on Ubuntu, Debian and CentOS, and so these are considered to be the supported operating systems.

2.2.2 Models

Each API in the BAE includes a set of models. Besides being actual models in the APIs and the database, they are a good way to get a better understanding of the system. All the models are listed and described below.

- Party API
  - **Individual**: a physical person.
  - **Organization**: a group of people identified by shared interests or purpose, e.g. a company.
- Customer API

  - **Customer**: an individual or organization that buys products and services.
  - **Customer Account**: an account for the customer to manage billing aspects. Can contain customer tax exemption, related accounts, contact information, customer relation, account balances and payment plans.

- Billing API

  - **Billing Account**: a detailed description of a customer's bill structure.
  - **Applied Customer Billing Charge**: an amount, usually of money, for which a person or an organization is financially liable.
  - **Settlement Note Advice**: the settlement is about transferring money received by a CSP to a partner. The settlement is notified to the partner with a settlement note advice containing details in settlement lines.

- Catalog API

  - **Product Specification**: a description of a product.
  - **Product Offering**: represents entities that are orderable from the provider of the catalog. This resource includes one or many product specifications and pricing information.
  - **Catalog**: used to group product offerings.
  - **Category**: used to group product specifications.

- Ordering API

  - **Product Order**: used to place an order between a customer and a service provider or between a service provider and a partner and vice versa.

- Inventory API

  - **Product**: a product a purchaser has acquired.

- Usage API

  - **Usage**: an occurrence of accessing a product, which is of interest to the business and can have charges applied to it.

2.2.3 Supported asset types

The BAE supports both digital and non-digital products. When creating a product specification the seller must choose whether the product is digital or not. If digital is chosen the seller is presented with a list of available digital asset types. These asset types depend on which plugins have been installed on the BAE instance (see Section 2.2.4 for more on plugins). A fresh install of the BAE includes no plugins and so no digital asset types are supported out-of-the-box. What kind of digital asset types that can possibly be supported is arbitrary as a developers can create

custom plugins. If a non-digital product is chosen no further information on asset type is required.

2.2.4 Plugins

When a seller is to create a new product specification for a digital product he/she has to choose an asset type. A fresh install of the BAE includes no asset types as they must be added through plugins. The BAE plugins are solely used for defining digital asset types. The majority of the assets require validation and activation mechanisms. These can be very different, and for that reason asset types require their own plugins, since then the creator of the plugin can code their own mechanisms. Validation is the way of making sure that the provided asset is of the right type. Activation is the way of activating the product after the purchase has been done, so that the customer can get access to it. There are six FIWARE-created plugins available:

- Basic File
- Basic URL
- CKAN Dataset
- CKAN API Dataset
- Umbrella Service
- WireCloud Component

Basic File and Basic URL are different from other plugins in that they do not require validation or activation processes. They allow for publication of any type of file or URL respectively. CKAN Dataset and CKAN API Dataset define an asset type that is a dataset offered by a CKAN instance (see Section 2.1.4 for more information). Umbrella service is for selling services secured with an API Umbrella proxy which is an API management platform for exposing web service APIs. WireCloud Component defines an asset type intended to manage and monetize different WireCloud components. WireCloud is a FIWARE GE [9] and is out of the scope of this report.

The installation of a plugin is simple with just one command in the Charging Backend CLI. All you need is a zip file of the plugin in question. The existing FIWARE-created plugins can be downloaded from their respective Github repositories.

It is also possible to create your own plugin. The plugin must consist of a JSON file with all the meta data as well as a Python script. The script should contain implementations of all the event handlers that you want to implement. There are 11 event handlers in total, which are called at different times during the creation, acquisition and activation process of a product. An event handler can contain any sort of code but must return an error or no error on success.

2.2.5 CKAN in the BAE

CKAN is supported in the BAE and is the main way to sell data on the marketplace. It can be installed in three different ways: from a package, from source and

with Docker. Installation from a package is the easiest and quickest way but requires Ubuntu as OS. Installation from source is the most complex way but is the most configuration friendly and can be installed on any arbitrary OS. Docker installation is easier than a source installation but is more complex and gives better flexibility than a package installation.

A survey from 2013 [26] showed that XML, CSV, XLS, JSON and RDF were the most supported data file formats for data marketplaces. All these file formats are supported in CKAN. It can also store data externally, meaning that the data is stored somewhere else on the web, and simply linked to via a supplied URL.

The BAE supports the use of CKAN for providers to offer data [7]. There are two different plugins in the BAE for supporting CKAN: CKAN Dataset and CKAN API Dataset. These plugins enable sellers to select CKAN Dataset/CKAN API Dataset as asset types when creating a product. To create a product with a CKAN Dataset or a CKAN API Dataset the seller must provide an asset URL that points to the CKAN Dataset that resides inside a CKAN instance, as well as the media type of the data e.g. JSON, CSV etc. The plugins validate that the asset URL points to a valid CKAN Dataset and that the seller has the rights to create a product that sells the provided dataset. They also manage the access to the dataset of the customers who acquire it.

In particular, these plugins are able to validate the dataset, validate the rights of the seller creating a product specification to sell the provided dataset, and manage the access to the dataset of those customers who acquire it. The difference between the two plugins is that CKAN API Dataset expects the data to be served by an external API that is secured with the FIWARE security framework. It uses the IdM roles and permissions to validate the permissions of the seller as well as grants customers access rights to the dataset.

However, CKAN does not provide a way to publish data that only certain users can access, i.e. private datasets. It also does not manage the access rights to published datasets. To be able to do this CKAN can be extended with two extensions: ckanext-oauth2 and ckanext-privatedatasets. Ckanext-oauth2 allows users to login through a OAuth2 instance, in our case the FIWARE IdM. Ckanext-privatedatasets allows datasets to only be accessed by a set of selected users.

2.2.6 Sanity check and testing

The BAE documentation provides some sanity check procedures that can be used by the system administrator to verify that the installation has been successful and that the system is ready for testing. These procedures include:

- **List running processes**: A command that lists relevant processes to check if the Glassfish server, Charging Backend, Logic Proxy, MongoDB and MySQL are up and running.
- **List network interfaces**: A command that lists ports in use to check if the expected ports are included.
- **Databases check**: Two commands that check whether MongoDB and MySQL are running.

After the sanity checks are done testing can be started. There are two types of tests included in the BAE: unit tests and end-to-end tests. The Charging Backend, the RSS and the Logic Proxy have their own included unit tests.

Now that we have the necessary information we are ready to move on to the next chapter where we explain what we actually did during the project.

Chapter 3
Methodology

In this chapter we explain what we did during the project. Firstly, the installation of the BAE is explained. Secondly, the creation of our own React front-end is presented. Lastly, we explain how we carried out performance testing as well as functional testing of the BAE.

3.1 Installation of the BAE

Firstly we installed the BAE using Docker which is the simplest method of installing. It is just a matter of downloading the BAE from github, set the working directory to the docker directory and running the docker command. This will download and install all the necessary software. By using docker however, one is limited by a pre-built image of the software and it is not possible to modify the code.

In order to be able run a the BAE where we could modify the code, which was necessary for our project, a manual install was required. Since we both were using Mac OS X, which is not considered a supported operating system, we set up a virtual machine on which we installed Ubuntu. With the use of the installation script and after some minor configurations the BAE was successfully installed.

We ultimately decided that a native installation on Mac OS X would be easier for us than using a virtual machine. Since the dependency software is also available on Mac OS X this should be possible. Following below is a complete guide how to manually install the BAE on MAC OS X. The guide is based on the official installation guide [12] with some tweaks for making it work for Mac OS X.

Before the installation can begin all the dependencies have to be downloaded and installed. This is easily done by using the package management system Homebrew with the following command:

```
brew install [dependency]
```

The dependencies include:

- Java 8
- MySql

- Maven
- Python 2.7
- Python 3
- MongoDB
- Wkhtmltopdf
- Node and NPM

The provided installation script was used with some minor changes. The database information has to be changed to the correct values for the local MySQL setup.

```
DBUSER = "MY_MYSQL_USER"
DBPWD = "MY_MYSQL_PASSWORD"
DBHOST = "MY_MYSQL_HOST"
DBPORT = MY_MYSQL_PORT
```

The branch of all the repositories was changed from *v6.4.0* to *develop*.

```
{
    ...
    "branch": "develop",
    ...
}
```

Also some printouts were added to make it more verbose. The script can be accessed from [30]. The script requires Glassfish, MySQL and MongoDB to be running before it is executed:

```
asadmin start-domain // Glassfish
sudo /usr/local/mysql/support-files/mysql.server start // MySQL
mongod // MongoDB
```

To start the installation download the script, move it to an empty directory and run the script with the following command:

```
./install.py all
```

After the installation some configuration have to be done to the config.js file in the Logic Proxy directory. The host and port of which the Logic Proxy can be accessed has to be set:

```
config.port = MY_LOGIC_PROXY_PORT
config.host = 'MY_LOGIC_PROXY_HOST' // e.g. 'localhost'
```

Some OAUTH2 information for the IDM has to be set:

```
{
    'server': 'MY_IdM_URL', // e.g. 'http://localhost:3001'
    'clientID': 'MY_BAE_APP_CLIENT_ID',
    'clientSecret': 'MY_BAE_APP_CLIENT_SECRET',
```

```
    'callbackURL': 'MY_CALLBACK_URL',
    ...
}
```

3.2 React front-end

The Sensative IoT platform Yggio is built in React.js, and all their pages are uniform in design, which meant that the design that FIWARE has implemented in their front-end was unwanted. As such, one of our tasks was to present a mock-up of how a front-end, built in React.js, could be used with the APIs that TMForum had implemented. We took note of how the new version of Yggio would look, so that our implementation would mimic the design of the platform it would be integrated into.

We used the boilerplate "create-react-app" because of the low learning threshold. The first task was to mimic the design, and we created a non-functional page which had the same sections as FIWARE's implementation, but presented, in our opinion, in a more logical and intuitive way.

FIWARE's marketplace can be seen below in figure 3.1. The structure of offered products and sections (e.g. My Inventory, My Stock, etc.) have been used in our design, as well as can be seen in Figure 3.2.

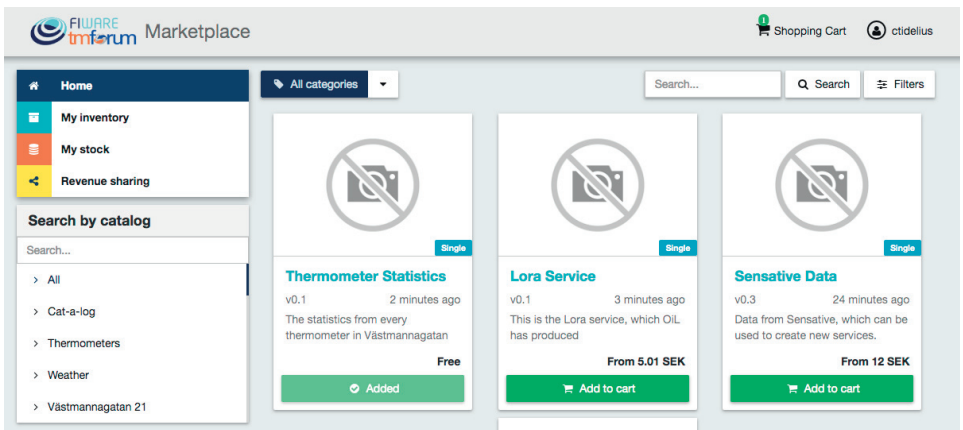

Figure 3.1: Fiwares out-of-the-box marketplace.

The source-code for FIWARE's marketplace did not give us the required insight to help us when implementing our own front-end because of its complexity and language (PUG). We attempted to learn as much as we could from their implementation, but realized that the API-calls that were made when using FIWARE's BAE, and the Apiary [6] which hosted the knowledge base for TMForum's APIs, would give us the basic information on the functionality that we needed.

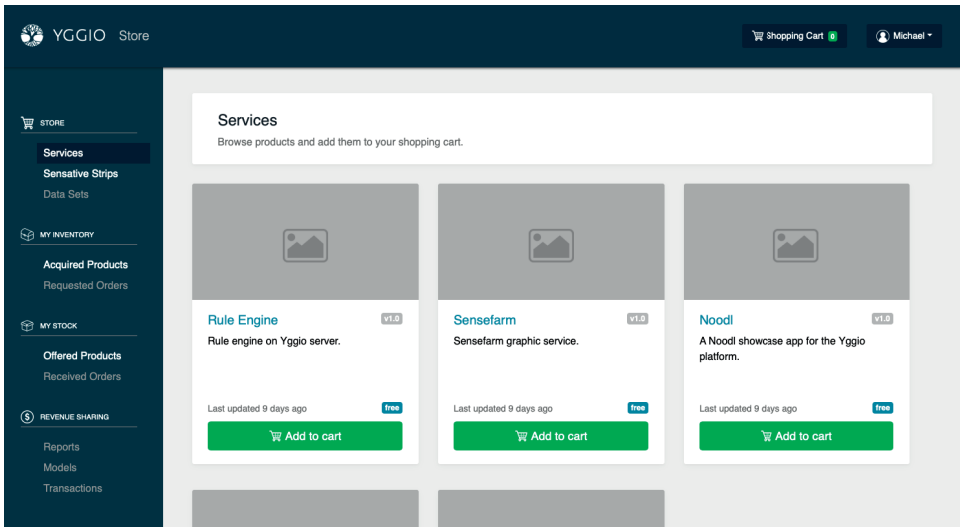

Figure 3.2: Our design of Sensative's marketplace.

3.2.1 Back-end

Because of the complexity of FIWARE's source code, the FIWARE backend was not integrated in the mock-up. An attempt to integrate the FIWARE IDM (identity manager) to be able to use an OAUTH2 token in the header in the API calls was made because the Apiary documenting the TMForum APIs specified that each and every call must include a token to authenticate the request. This is necessary for a production version of the software, but as we noticed if the API was called directly, and not through the BAE front-end which provided some security and validation of api calls, one could circumvent the need of having to use OAUTH2 tokens. As such, it was decided to implement our back-end to host a database with information about products in the shopping cart, similar to FIWAREs shopping cart approach.

Our implementation of the BAE used a few different API calls to list, create and update products. In the FIWARE BAE there are multiple steps one has to go through in order to create a product, which our implementation simplified in order to make it easier for the user to understand. As described earlier in Section 2.1.3 an offering must relate to a product specification. In order to simplify this, when creating a product in our BAE two calls were made to the API; one to create a product specification and one to create an offering relating to this specification.

During the implementation of upgrading a product, it was discovered that the user would have to increase the version number in order to update other information about the product. This was not the case in FIWARE's BAE, where we discovered that they had made a method to override the APIs restrictions. It was also not presented as a required parameter in the Apiary [6].

3.3 Performance testing

To test the performance of the BAE three external programs were used; Valgrind, Api-benchmark and LoadUI.

3.3.1 Valgrind

Valgrind is a tool used for memory debugging, memory leak detection and profiling. While Valgrind is primarily used to debug C/C++ programs, because it works directly with program binaries, it works with programs written in any programming language, be they compiled, just-in-time compiled, or interpreted. Valgrind was used on the Logic Proxy, Charging Backend and the Glassfish server (i.e. the APIs). When used with the Logic Proxy we noticed that the program could not be run successfully, as it checked for memory leaks, ran into errors and then exited with a segmentation fault. Without Valgrind it ran without errors.

3.3.2 Api-benchmark

Api-benchmark is a node.js tool that measures and compares performances of APIs. Its Github page can be found in [8]. The scripts we used can be seen in the Appendix A. Note that we tested our local instance of the BAE and so the results should differ from a production instance running on a remote server where network delays come into play. The result that we were mainly interested in was the mean of the response time, but it also gave other results such as:

- Number of operations per second
- Sample arithmetic mean
- Sample standard deviation
- Margin of error
- Standard error of the mean
- Sample variance

While all APIs were tested, although only the most basic GET-requests were made, two of the calls made required an OAUTH-token so those were not taken into consideration.

3.3.3 LoadUI

LoadUI is another tool for testing API performance. We used it to see how the APIs would handle different loads. We checked the average response time when 1, 10 and 25 requests were sent every second in five minutes straight. Only one API, namely the Catalog Management API, was tested since the other APIs should behave in the same way.

3.4 Running the test scripts

The Logic Proxy tests were run by navigating to its directory and running the command:

```
npm test
```

The RSS tests were run by navigating to its directory and running the command:

```
mvn test -fae
```

The Charging Backend tests were run by navigating to its directory, activating the virtualenv and then running the command:

```
python src/manage.py test
```

In the next chapter the results of the performance testing and functional testing is presented.

Chapter 4
Results

In this chapter the results of the performance testing and functional testing are presented and briefly discussed. A more in depth discussion of the results and its consequences is given in the next chapter.

4.1 BAE Performance

4.1.1 Valgrind

When running Valgrind with the Charging Backend and the Glassfish server (i.e. the APIs) the results were inconclusive, although showing no indication of memory loss. When running Valgrind with the Logic Proxy the following results were returned:

```
LEAK SUMMARY:
    definitely lost: 0 bytes in 0 blocks
    indirectly lost: 0 bytes in 0 blocks
    possibly lost: 7,528 bytes in 119 blocks
    still reachable: 256,169 bytes in 201 blocks
    of which reachable via heuristic:
    newarray           : 2,568 bytes in 1 blocks
    suppressed: 54,325 bytes in 160 blocks

ERROR SUMMARY: 117 errors from 117 contexts
(suppressed: 16 from 16)
```

and the program did not start correctly. This would mean that the program has memory leaks which could possibly result in problems for the program, but since the program did not start, we could not see any definitive results in Valgrind of whether the memory loss would effect the program during its lifetime. However, when running the program during a few days a monitoring of the activity monitor showed no indications of any unusual memory consumption.

4.1.2 Api-benchmark

The results from the Api-benchmark script can be seen below, in figures 4.1 to 4.5

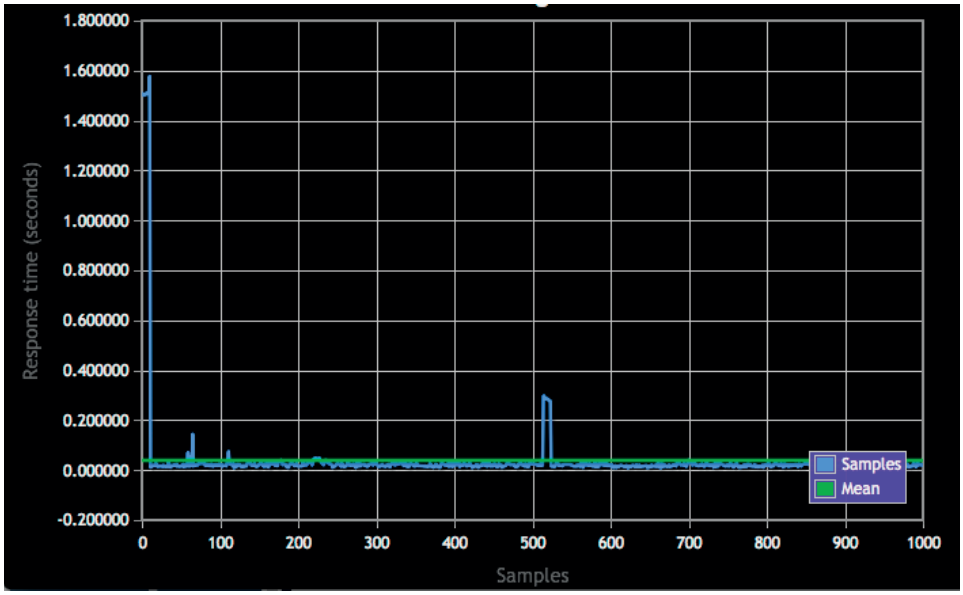

Figure 4.1: Api-benchmark test results for the Billing Management API

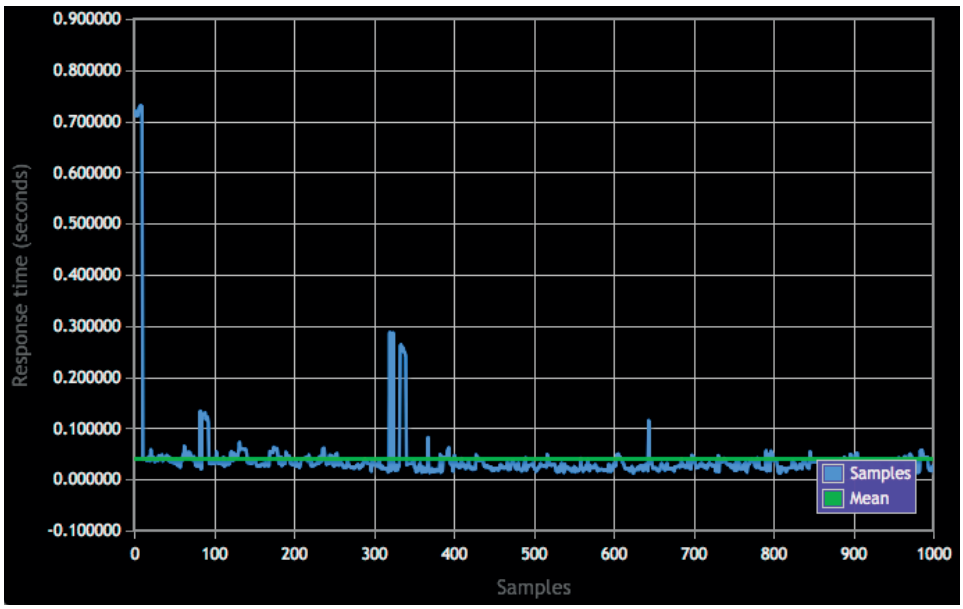

Figure 4.2: Api-benchmark test results for the Catalog Management API

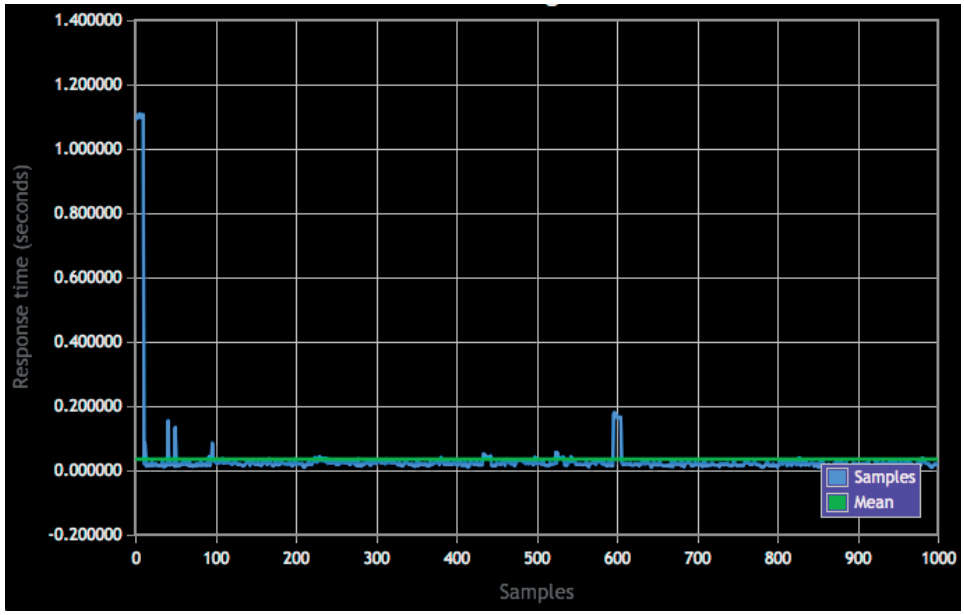

Figure 4.3: Api-benchmark test results for the Customer Management API

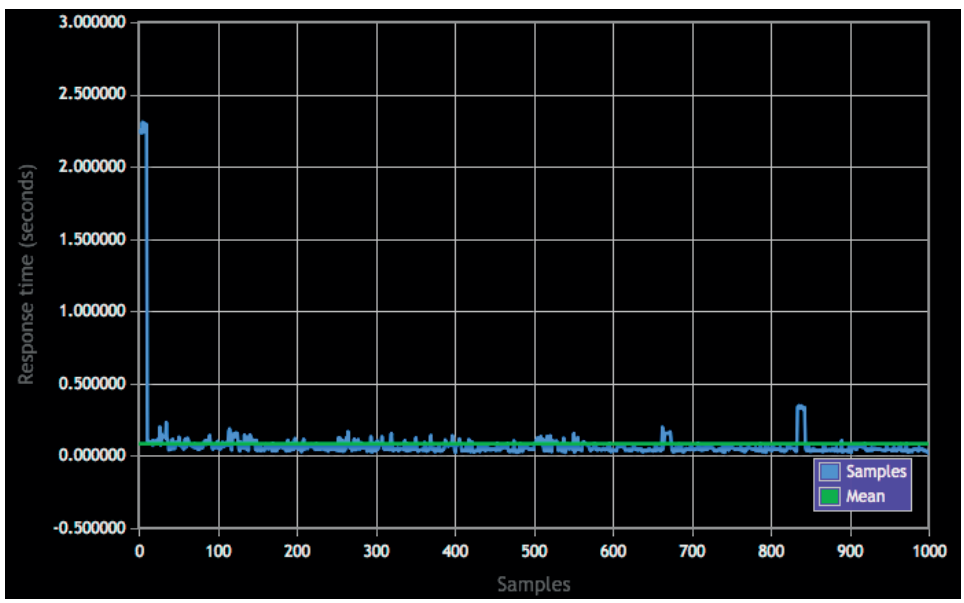

Figure 4.4: Api-benchmark test results for the Order Management API

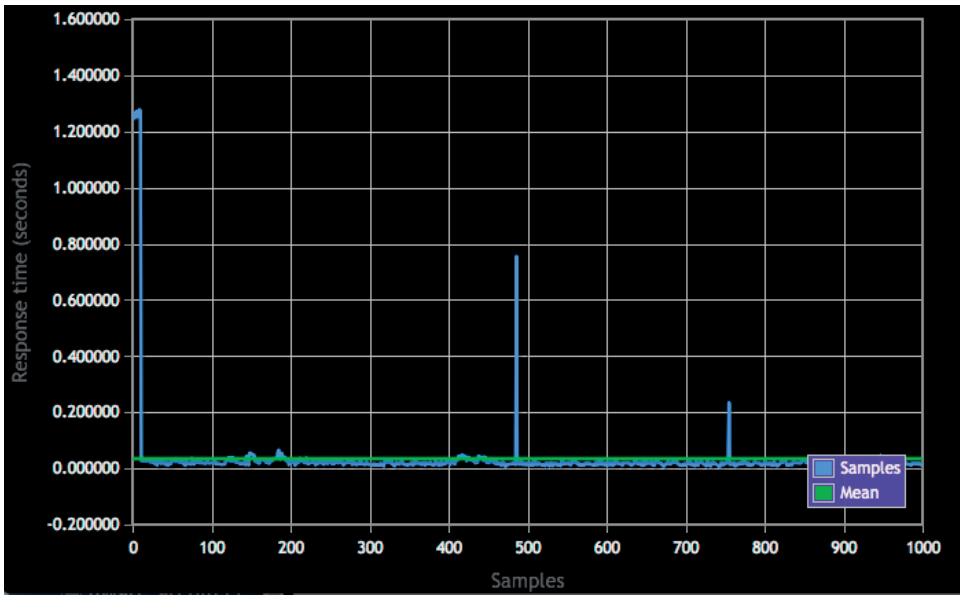

Figure 4.5: Api-benchmark test results for the Usage Management API

API	Mean	Standard Deviation
Billing Management	0.038812	0.150908
Catalog Management	0.040074	0.073614
Customer Management	0.034464	0.108654
Order Management	0.083557	0.223235
Usage Management	0.034197	0.126161

Table 4.1: API test results

These results indicates that the initial calls have high response times, which puts the mean average on a higher value as well as the standard deviation which can be seen in Table 4.1. However, when testing against other APIs, their values were much higher than the BAE API, most likely due to the fact that the BAE API was hosted on our own computers. The two tested APIs can be located in [29] and [23] and the results are presented in Table 4.2

API	Mean	Ping	Standard Deviation
BAE Catalog Management API	0.040074	N/A	0.073614
https://swapi.co/	0.556637	0.041749	0.315425
https://www.pokeapi.co/	0.409747	0.036550	0.593648

Table 4.2: API test results

4.1.3 LoadUI

Table 4.3 shows the avarage response time when load testing the API with 1, 10 and 20 requests/s in 5 minutes.

Requests/s	Average response time	Standard Deviation
1	22 ms	11.180339 ms
10	25 ms	46.238512 ms
25	82 ms	373.284074 ms

Table 4.3: API load test results

Figure 4.6 to 4.8 shows the average response time over time. It also shows number of failed requests.

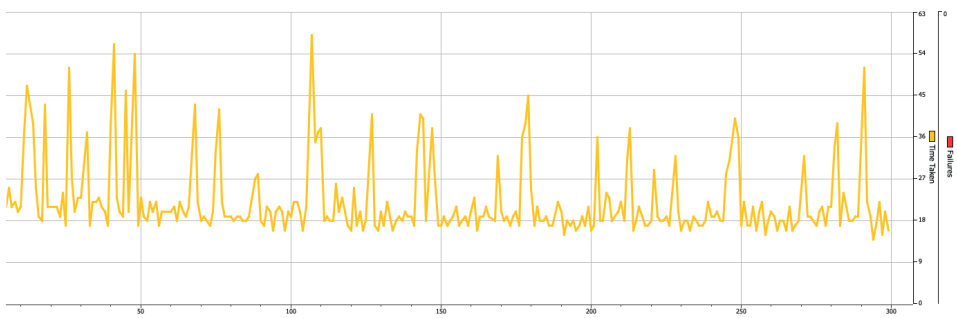

Figure 4.6: Load test results with 1 request/s

The first load test shows a rather good average response time with quite large standard deviation. The API can be considered stable with no large delays. The

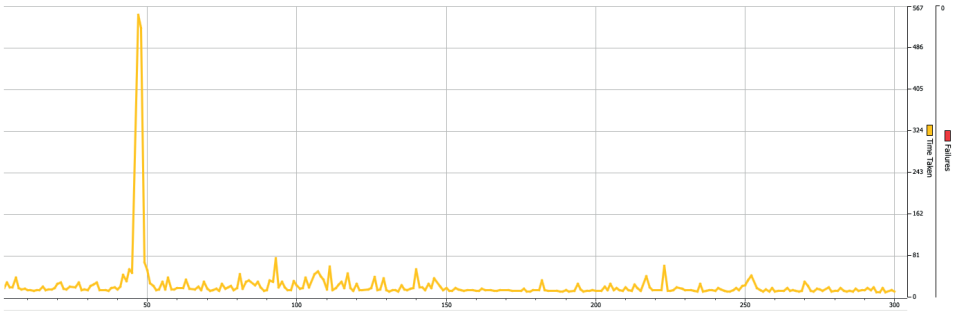

Figure 4.7: Load test results with 10 request/s

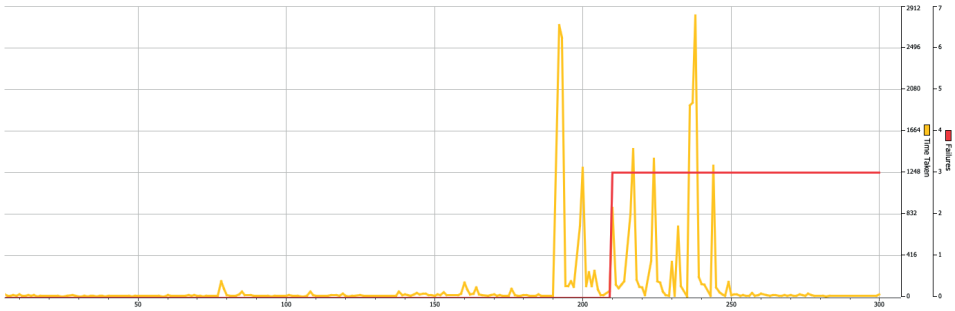

Figure 4.8: Load test results with 25 request/s

second load test follows the same pattern except for one deviation where two points reach a time of 560 ms. This result in a very large standard deviation. The third load test has a period of many large deviations which also results in three faults. This test results in a very large standard deviation and a relatively large average response time so the system might be considered unstable at this much load.

4.2 BAE Tests

Table 4.4 shows the results of running the test scripts for each component in the BAE.

Component	# tests	# errors	run time (s)
Logic Proxy	681	0	21.151
RSS	149	0	21.023

Table 4.4: BAE test scripts results

The Charging Backend test script failed to execute properly and as such gave no results.

Table 4.4 shows the test coverage of the Logic Proxy.

Statements	Branches	Functions	Lines
94.43%	91.59%	96.31%	94.55%

Table 4.5: Logic Proxy test coverage

Chapter 5
Discussion & Conclusion

In this chapter we first apply the ISO/IEC 25010 standard to evaluate the BAE. Secondly, we discuss enhancements, limitations and other relevant topics. We end with a conclusion of our findings.

5.1 BAE quality evaluation

A common way to evaluate software quality is by using the ISO/IEC 25010 standard [15]. It specifies a quality model that determines which quality characteristics will be taken into account when evaluating the properties of a software product. We used this standard to evaluate the quality of the BAE. Definitions of each characteristic can be found on the ISO25000 website referenced above.

5.1.1 Functional suitability

Functional completeness

The FIWARE BAE does have high functional completeness, if one takes plugins into consideration. If the opposite is considered the BAE does not support the selling of data inherently, and does not allow for URL access grants without the use of plugins, and as such does not have the functional completeness one would expect from a marketplace for data and services.

The BAE does support the largest payment option in the world (PayPal), and this is in most cases enough for many marketplaces. But because of the implication that this platform would be used for smart cities, one would expect an easier way to implement new payment options. For instance, in Sweden, BankID would be the preferred payment option for most customers.

There is no way of specifying who made a product. In our opinion it would be preferred if you knew what user or which company is offering/made the product. The only option now is to specify a category (only creatable by the administrator) or catalog, which anyone can specify and add products to. This could be done to make a more reliable platform, since if you know that Stockholms Stad is offering the product, then you are more inclined to trust that the data being sold is genuine.

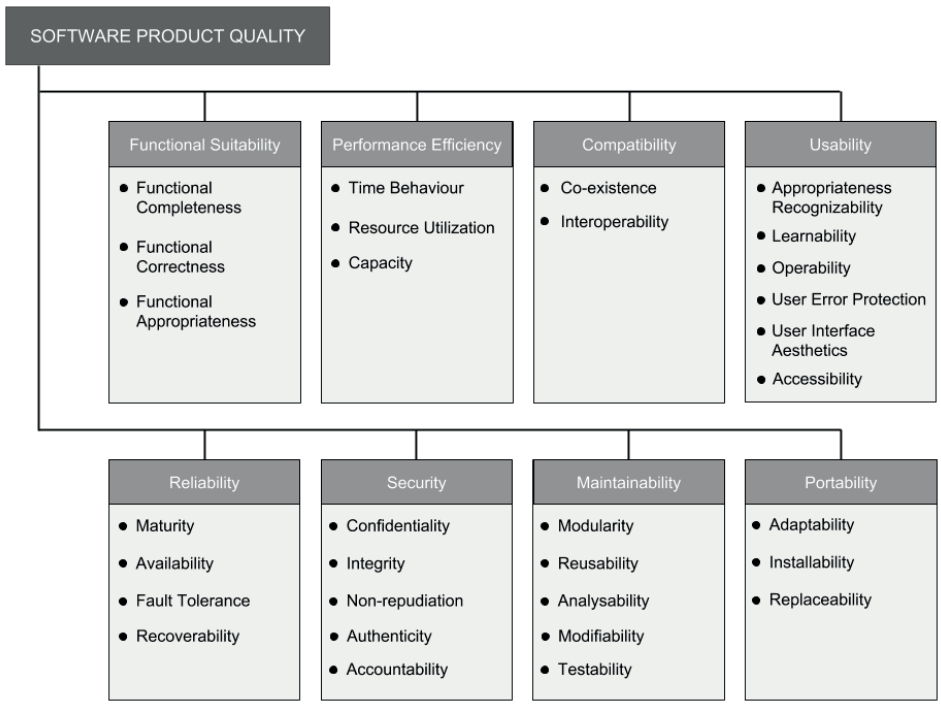

Figure 5.1: The ISO/IEC 25010 standard

One feature that is absent from the BAE is that of *related/suggested products*. This is a common feature in web stores and is useful for sellers to increase sales as well as for customers to find relevant products.

Functional correctness

We have observed a few error messages, mostly because of incorrect installments of the BAE, but during the completion of a purchase, the transaction-API-call was made twice and resulted in an error message being thrown. The transaction still went through (or will go through, as the transactions will not be made until the end of the month). This was tested on version 6.4.0 and the developers have released a new version (7.4.0) of the software since, which might have fixed this bug.

The shopping cart on the platform allows you to add any product that you wish to purchase. This does not, however, mean that you are allowed to purchase all products, since you cannot acquire your own products.

Functional appropriateness

There are a number of complications to consider when a new user is introduced to the BAE. Firstly, the new user is not assigned the seller role, and cannot add himself to become a seller without having access to the FIWARE IdM BAE application. As such, only administrators can administrate the roles of the users. This is of course intended to be able to vet the potential sellers, but it complicates the process of selling your data, and could discourage the user altogether.

On the platform, you must specify what the product you are selling is. This is certainly not strange, but when coupled with the fact that you are just creating a specification of the product, and not an actual acquirable object, it becomes confusing. To be able to sell a product, the user must create an offering, relating to a product specification. This would make sense if we got some sort of indication or tutorial on how and why this must happen, but it is up to the user to discover and explore everything. The reason for this, is to be able to create bundles of offerings in one offering. This is not, however, specified in any way, which makes the relationship between the product specification and the offering slightly confusing.

5.1.2 Performance efficiency

Time behaviour

The results of the Api-benchmark testing shows that the average response time ranges between 30 ms to 40 ms except for the Order Management API which had a time of 83 ms. The standard deviation are rather high. This is probably due to the long response time of the first request of every test. The load test results with 1 request/s shows an average response time of 22 ms. Why this is lower than the Api-benchmark results is probably due to that they were run on a faster computer.

Resource utilization

Since we were unable to fully run the program with Valgrind, the resource utilization analysis remains incomplete. With the before mentioned results we can say that the software should be analyzed to implement solutions to the lost bytes of memory when starting the BAE, but a check on memory utilization in the Activity Monitor (Resource Monitor in Windows) results in no drastic changes while using the software during four days. Our analysis of the memory utilization is, however, not a precise analysis, and the functionality used was not process demanding. This could have effected the results to be skewed.

Capacity

The capacity was tested by the load tests. The results are discussed in Section 4.1.3. To conclude; the load test indicates that the API is stable under low loads and becomes unstable at high loads.

5.1.3 Compatibility

Co-existence

The possibility of having two versions of the BAE running on the same system is hypothetically possible, but you would have to create new databases, change port settings and possibly other settings. The possibilities depend on what you would want to do with the two systems. For instance, if you want to list the same products and use the same instance of the Identity Manager, then you would have to change the port numbers, but would not have to create new databases.

This characteristic is however out of scope of our investigation, and will as such not be investigated further.

Interoperability

Since a few of the APIs do not require an OAUTH-check, it is possible to exchange and make use of some information. You can, for example, list the product specifications on another platform by calling the API, but you cannot make any transactions or update products. The most important, as in security related importance, API calls are protected by OAUTH, which means that no calls that can impact the software negatively can be made.

5.1.4 Usability

The usability evaluation are restricted to the front-end of the BAE since this is what most users interact with.

Appropriateness recognizability

It is fairly easy to recognize the front-end as being a web shop as it uses a lot of the same visual components as other webshops and app stores. The products are

shown as "cards" which helps to indicate that they are separate entities, similar to products in a physical world store. The buy buttons and the checkout button are green and have shopping cart icons which are common components in webshops.

Learnability

We consider the learnability of the BAE to be rather low. This is mostly due to confusing models and the relationship between them. In order to create a product a seller has to create a product specification and then an offering in which the product specification is included. The fact that you have to create two different entities in order to sell one product is not very obvious. Also the difference between catalogs and categories can be confusing. However there is a user guide in the documentation that helps with the learning process.

Operability

While a large amount of the platform's operability can be considered good, there are sections that are unsatisfactory. For instance, the sub-menus for *My inventory*, *My stock*, and *Revenue sharing* are placed beneath the grouping of menu items, which leaves the user wondering if it is connected to the menu item he chose, or not. Sub-menus should relate to their parent by either being exposed beneath the parent, or by being accessible in the window corresponding to the item.

User error protection

When submitting a form on the BAE the user will be getting feedback on inputs that are not correct. Most of the time this feedback is constructive but sometimes it only shows arbitrary server error messages which makes it difficult for the user to correct the error. Many of the form inputs could use a description on the expected format. For example, when entering the version number of a new product specification it would be beneficial to have a helping text; "e.g. 1.0". It should not be possible to create two product specifications with the same name, at least not also with the same version number, which in its current state it is. When searching for an offering the BAE shows no suggestions, which will lead to customers not finding what they seek if they do not know the exact name or if they misspell. When a seller wants to disable a product specification or offering he/she can put it into the state of *retired* or *obsolete*. However, it is not possible to revert the action if the user made a mistake.

User interface aesthetics

The aesthetics of the front-end is only okay. The color scheme is of bright colors and there is not much clutter which makes for a calm and focused experience. The fact that products are shown as cards instead of in a list makes it less technical and more fun. Many elements cast a drop shadow which creates a sense of depth. Both card layout and drop shadow are part of the popular design language Material Design by Google [14]. However, the general design of the BAE looks a bit outdated and the project would benefit from introducing a more modern design.

Accessibility

BAE only supports two languages, English and Spanish. This is obviously a problem for people not speaking those languages. The design doesn't seem to be very reliant on colors as there are no components that only use color to communicate information, which makes the website accessible for colorblind users.

5.1.5 Reliability

Maturity

Maturity is a characteristic that has a debatable definition. In [2] Wu puts an emphasis on maintenance in the software life cycle. In regards to the BAE, though there is not a high activity on the repository, maintenance is ongoing and bugs are being reported and handled. Others [20] state that maturity should be measured in Mean Time To Failure (MTTF) which is based on how many errors the user encounters in the software. In our experience, the maturity in this aspect is high, since the errors we have encountered were mainly a result of faulty installations.

Availability

The BAE should at any time be available as long as the server it is hosted on is fully functional. No apparent reasons have been discovered where the system is not fully operational.

Fault tolerance

With the presence of hardware faults, we cannot ensure that the BAE will be fully functional. If a software crashes, there will also be loss of funcitonality, although as long as the Logic Proxy does not crash, its error handling will handle the faulty operations.

Recoverability

There exists no automatic recovery. A maintainer must manually restart any crashed program. At least one error will result in an inconvenient error report as the error was shown despite the fact that the desired action was completed.

5.1.6 Security

Confidentiality

In the BAE confidentiality is assured by the use of the IdM. The IdM provides authentication and access control. Roles such as *seller* and *customer* are given to users by the administrator to manage access control.

Integrity

To ensure that data sent from the system to the user remain correct and unchanged encryption can be used. In the BAE the protocol HTTPS is supported. Enabling HTTPS in the BAE is easy and is done by changing a few lines in the Logic Proxy configuration file. However, there is no added security for integrity when the data is stored on the system. For that you have to rely on the security of the hosting server and the databases.

Non-repudiation

The products made in the BAE cannot be deleted, only be put into a "disabled" state, and purchases will be documented both in Paypal and in the form of billing receipts. Because of this the BAE does have some ability to prevent actions being repudiated later.

Accountability

Since accountability is the traceability of actions performed on a system to a specific system entity, and the IDM provides an ID to each entity, there is some form of accountability. It is possible to see what actions the ID performs, such as creating, acquiring, and selling products. Since there are no functions of the system that are abusable to legal extent, the accountability of the BAE is not vital. Although in the event of a user selling data that has been purchased on the BAE, the accountability of the BAE does provide some traceability.

Authenticity

OAUTH2 is mainly an authorization protocol but also provides pseudo-authentication. There is no way of ensuring the authenticity of offerings of data.

5.1.7 Maintainability

Modularity

The code is written to be modular, but as in any program, some changes will have larger impacts. The different components in the FIWARE BAE all have their own responsibility and the biggest connection between them is the Logic Proxy, since it contains the API orchestrator.

Some modules have closer relationships than others, which would mean that changes in these modules would have larger impacts on the modules they have closer relationships with.

Reusability

Since the program is written to be modular, there is reusability in the system. The APIs which have been created by TMForum are intended to be used in Data

Sharing Management, and this particular instance has been developed to accommodate FIWAREs needs. As such, the APIs can be used in other instances as well.

The other modules (RSS, Charging Backend and the Logic Proxy) would be harder to reuse, since they are directly dependent on the APIs mentioned earlier.

Analysability

It is not easy for an inexperienced developer to analyse the code and understand how and what to change without creating problems for other modules, especially since it is a large project, which has been developed over several years. As such, we have decided that it is out of our scope to examine the code enough to include analysability in our report.

Modifiability

As this section interconnects with analysability, we have decided that this is also out of scope for our report.

Testability

As described in the Chapter 3 (Methodology), when downloading the BAE, three test scripts are included. These scripts are made by FIWARE and tests the Charging Backend, the RSS and the Logic Proxy. As such, we ourselves have not established any test criteria, but FIWARE themselves have. The tests all passed without errors, and on the GitHub page FIWARE has included thier own Quality Assurance, which indicates that their only test criteria is whether their tests pass. They do have tags that indicate that Scalability, Performance and Reliability also are tested in many of the Generic Enablers in their Ecosystem, but they have not tested any of those criteria on the FIWARE BAE. It is also unclear what those criteria entail.

5.1.8 Portability

Adaptability

Since we have developed our own front-end for the BAE, we have effectively proven that it is indeed possible to adapt the system for new technologies, at least for React. However, to run the program on newer versions of dependencies proved to be harder. First of all, we had to use Node 8.0.0, which also means that npm 5.0.0 had to be used (the current version of Node is 10.14.2 and for npm it is 6.4.1). Python3 also proved to be a hurdle, and we had to use 2.7. As such, the adaptability of the system is questionable. It might be as easy as upgrading a few dependencies, but without modifying code, and making a change assessment, we can not speculate on how efficient and effective the adaptability is.

Installability

With the documents provided by FIWARE on fiware-biz-ecosystem.readthedocs the installability is increased, but although they do specify some version requirements, these requirements are not always correct. For example, they have specified that Node 6.9.1 and above are required for the Logic Proxy to be installed, but versions over 8.0.0 do in fact not work with the Logic Proxy. Uninstallment of the software is left up to the user entirely, and the documents do not offer any kind of description of this step. So we cannot know if we are deleting all of the software when cleaning up our directories.

It is, however, easier if the user is using docker. Since docker puts up containers which contain all the necessary dependencies and the like, both the installment and uninstallment are facilitated significantly. This does require that you have a fair bit of knowledge about how Docker works, since you will have to connect the containers with other software, such as CKAN and the IDM.

Replaceability

This is the only instance of a store that sells both services and data that we have come across. There does exist a service that sells data called DataBrokerDAO, although this is not software that any company can acquire and use as their own. As such, the replaceability of this software is arguably zero.

5.2 Enhancements, limitations and general discussion

5.2.1 API testing

When evaluating APIs in relation to other APIs, especially when testing a local API, there are limitations of how accurate the evaluation can be. First of all, the processing required to do different tasks is varied, so to compare two APIs requires similar functions. The second problem is the network latency. Thirdly, there are a number of different technical issues which we cannot take into account. As such, the requests that we made were small ones, so as to not put too much processing pressure on the system. We also negated the network latency by making ping-calls to the APIs which were hosted by other parties which requires so little processing that we can almost see this as the network latency. This method is by no means exact, but by subtracting the ping values from the average response times we considered the network latency to be negated. In this case, the BAE APIs have values that can be considered very good, compared to APIs not made by corporations.

The fact that the variance differed so much in the load testing, versus the api-benchmark testing was because of the choice to present the values in seconds and milliseconds between the two tools. As such, the variance being low in the api-benchmark testing showed that the response time of each call was diverse, while this indication was shown by the variance being high in the load testing.

5.2.2 Real time IoT Data

The BAE itself does not support any way of uploading data from sensors in real time, which in our opinion, is the purpose of an IoT Data Marketplace. Instead, it facilitates the monetization process by having subscriptions and pay-per-use management. As such, to integrate the sensor flow of data, one would have to use either the GEs of FIWARE to route the sensors data via MQTT as described in [18], or push data to the CKAN datastore in regular intervals and in this way simulate the real time data flow of a sensor.

5.2.3 Popularity and activity of the BAE

A factor that can not be neglected when choosing whether to begin to use a software is how popular it is. It gives an indication of how mature, bug-free and maintained the software is. Also the chances of the software being discontinued in the future is lower if it has many active users. While we researched who are using the BAE today we only came across two large companies, which is not very much, although one of the companies (Synchronicity) has a EU funded project, where several cities will use the BAE. BAE's Github repository shows a rather low activity with only 43 commits last year (2018).

5.2.4 Legal aspects

The BAE is only an accommodator of products and as such the owner of it does not take responsibility for the products. It is the seller of the product that has the responsibility towards its customers. The customer must single handedly check what laws apply in the country where the seller resides. The BAE does however give the option for sellers to supply terms and conditions when creating an offering which the customer must accept upon purchase.

In the store itself, there is nothing hindering the users from selling the same data, which would mean that users can buy some data and sell it at a lower price, effectively hindering the original seller from selling his/her data. What can be done in situations like this is having a Terms and Conditions that the buyer has to accept when acquiring the product (which is possible in the BAE). This would protect the seller, but could also turn into a legal situation. There should exist some form of protection from this type of problem. Although the BAE is, as mentioned, only an accommodator and would not have any legal obligations to interfere.

5.2.5 Alternatives to the BAE

As of this moment, there are no open source IoT data markets that are able to sell both data and services except for the BAE. There are however many data marketplaces which allow you to sell your sensor data, both in real time and in bigger volumes. The main difference between these marketplaces and the BAE is the fact that you can use the BAE as your own platform, while the competitors are already established data markets.

5.3 Conclusion

FIWARE's Business API Ecosystem is a viable choice when looking for an online IoT focused marketplace. It includes most of the features you could ask for in a standard web store. The design of the out-of-the-box web front-end is relatively good and its usability is sufficient for its intended target audience. However, being a marketplace that is focused on IoT, it leaves a lot to ask for. A fresh install of the BAE does not provide many features that go beyond what a regular web store offers. It is only with plugins that the IoT-related functionality is added, such as being able to sell digital products and data. As data is a big part of IoT you might expect the monetization of data would be built-in feature in a IoT marketplace. Instead the BAE uses the external program CKAN, which also requires a plugin. However, this separation of functionality can also be seen as positive as it makes for a more modular program where you pick and choose the features you need.

The BAE is officially only supported on Linux operating systems, but as we found out, also works on Mac OS X. Installing the BAE is rather complex and requires quite a lot of configuration. This should not, however, be a big problem for the target audience that will install and maintain the program. The program seems to be stable with no faults occurring, although more tests would be beneficial. The performance is relatively good with the exception of the APIs showing some bad performance at higher loads.

The BAE is a relatively small project in terms of number of people developing it, the number of people using it and its activity on its Github repository. This might be a reason to not start using the BAE as it is an indication of low maturity and the risk of the project being discontinued in the future.

Despite its flaws, the BAE makes for a viable choice due to the low number of alternatives. As of now it is the only open source IoT marketplace that supports monetization of both services and data. It is also one of the only open source projects in this area, with only one other open source IoT marketplace found in our research.

References

[1] BDEX, datamarket, https://www.bdex.com/ [8 February, 2019]

[2] B. H. WU, 2012. Modeling software maturity: A software life cycle management approach, 2012 IEEE International Conference on Information Science and Technology 2012, pp. 716-720.

[3] CKAN, About - ckan. Available: https://ckan.org/about [28 November, 2018].

[4] Data, datamarket http://www.data.com/ [8 Feruary, 2019]

[5] DE LA VEGA, F., Biz-Ecosystem documentation. Available: https://business-api-ecosystem.readthedocs.io/en/latest/index.html [18 September, 2018].

[6] DE LA VEGA, F., FIWARE TMF Business API Ecosystem - Apiary. Available: https://fiwaretmfbizecosystem.docs.apiary.io/# [10 December, 2018].

[7] DE LA VEGA, F., Plugins Guide. Available: https://business-api-ecosystem.readthedocs.io/en/latest/plugins-guide.html\#ckan-dataset-and-ckan-api-dataset [28 November, 2018].

[8] FIGUS, M., Api-benchmark Repository. Available: https://github.com/matteofigus/api-benchmark [10 December, 2018].

[9] FIWARE, 17 September, Application Mashup - Wirecloud. Available: https://catalogue-server.fiware.org/enablers/application-mashup-wirecloud [30 November, 2018].

[10] FIWARE, Developers - FIWARE. Available: https://www.fiware.org/developers [17 September, 2018].

[11] FIWARE, FIWARE Readthedocs. Available: https://tourguidemigration.readthedocs.io/en/latest/05_open_data/How%20to%20Publish%20Context%20Information%20as%20(Open)%20Data%20in%20CKAN/ [15 December, 2018].

[12] FIWARE, Identity Manager - Keyrock - Fiware-IdM. Available: https://fiware-idm.readthedocs.io/en/latest/index.html [14 September, 2018].

[13] FRIEDMAN, T., LANEY, D., HARE, J., 2016. Prepare to monetize data from the Internet of Things

[14] GOOGLE, Design - Material Design. Available: `https://material.io/design/` [17 December, 2018].

[15] ISO/IEC, ISO 25010. Available: `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010` [4 December, 2018].

[16] J. P. BROGAN and C. THUEMMLER, 2014. Specification for Generic Enablers as Software, 2014 11th International Conference on Information Technology: New Generations 2014, pp. 129-136.

[17] K. MIŠURA and M. ŽAGAR, 2016. Data marketplace for Internet of Things, 2016 International Conference on Smart Systems and Technologies (SST) 2016, pp. 255-260.

[18] KRISHNAMACHARI, B., POWER, J., SHAHABI, C. and HO KIM, S., 2017. IoT Marketplace: A data and API market for IoT devices.

[19] MCKINSEY, 2018. The Internet of Things: How to capture the value of IoT.

[20] LOSAVIO, F., CHIRINOS, L., LÉVY, N. and RAMDANE-CHERIF, A., 2003. Quality characteristics for software architecture. Journal of object Technology, 2(2), pp. 133-150.

[21] NG, I.C.L. and WAKENSHAW, S.Y.L., 2017. The Internet-of-Things: Review and research directions. International Journal of Research in Marketing, 34.

[22] PERERA, C., ZASLAVSKY, A., CHRISTEN, P. and GEORGAKOPOULOS, D., 2013. Sensing as a Service Model for Smart Cities Supported by Internet of Things.

[23] Pokémon API. Available: `https://pokeapi.co/` [18 December, 2018].

[24] ROBERT, B., 2014. Towards the trillion sensors market. Sensor Review, (2), pp. 137.

[25] SCHOMM, F., STAHL, F. and VOSSEN, G., 2013. Marketplaces for data: an initial survey. SIGMOD Record, , pp. 15-26.

[26] STAHL, F., SCHOMM, F., VOMFELL, L. and VOSSEN, G., 2017. Marketplaces for Digital Data: Quo Vadis? Computer and Information Science, 10, pp. 22-37.

[27] STAHL, F., SCHOMM, F., VOMFELL, L. and VOSSEN, G., 2015. Marketplaces for digital data: Quo vadis? University of Münster, European Research Center for Information Systems (ERCIS), .

[28] STAHL, F., SCHOMM, F., VOSSEN, G. and VOMFELL, L., 2016. A classification framework for data marketplaces. Vietnam Journal of Computer Science, 3(3), pp. 137.

[29] Star Wars API. Available: `https://swapi.co/` [18 December, 2018].

[30] TIDELIUS, C., JIVUNG, M. https://gitlab.com/jivung/bae/blob/master/install.py.

Appendix A

API-benchmark

```
var apiBenchmark = require('api-benchmark');
var fs = require('fs');

var service = {
       server1: "http://df843add.ngrok.io"
};

var routes = {
               catalog: {
                       method: 'get',
                       route:
                       '/DSProductCatalog/api/catalogManagement
                       /v2/productOffering',
                       headers: {
                               'Accept': 'application/json'
                       }
               },
               charging: {
                       method: 'get',
                       route:
                       '/charging/api/assetManagement
                       /assetTypes',
                       headers: {
                               'Accept': 'application/json'
                       }
               },
               order: {
                       method: 'get',
                       route:
                       'DSProductOrdering/api/productOrdering
                       /v2/productOrder',
                       headers: {
                               'Accept': 'application/json'
                       }
```

```
                },
                custmanagement: {
                        method: 'get',
                        route:
                        '/DSCustomerManagement/api/customerManagement
                        /v2/customerAccount',
                        headers: {
                                'Accept': 'application/json'
                        }
                },
                billmanagement: {
                        method: 'get',
                        route:
                        '/DSBillingManagement/api/billingManagement
                        /v2/billingAccount',
                        headers: {
                                'Accept': 'application/json'
                        }
                },
                usagemanagement: {
                        method: 'get',
                        route:
                        '/DSUsageManagement/api/usageManagement
                        /v2/usageSpecification',
                        headers: {
                                'Accept': 'application/json'
                        }
                },
                rss: {
                        method: 'get',
                        route:
                        '/DSRevenueSharing/rss/algorithms',
                        headers: {
                                'Accept': 'application/json'
                        }
                }
        };

apiBenchmark.measure(service, routes, {
                        debug: false,
                        runMode: 'parallel',
                        maxConcurrentRequests: 10,
                        delay: 0,
                        maxTime: 100000,
                        minSamples: 1000,
                        stopOnError: false
                }, function(err, results){
```

```
        apiBenchmark.getHtml(results, function(error, html){
                fs.writeFileSync('benchmarks.html', html);
        });
});
```

Printed by Tryckeriet i E-huset, Lund 2019

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-687
http://www.eit.lth.se