UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

COMMUNICATIONS AND SYSTEMS ENGINEERING

Akintomide Ayodeji Akinsola

EVALUATION OF JSON-API-FORMAT REST API FOR BUSINESS-TO-BUSINESS INTEGRATION SCENARIOS

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 14 October, 2018.

Supervisor	Mohammed Salem Elmusrati
Instructor	Petri Kortelainen

TABLE OF CONTENTS PAGE

TITLE PAGE
TABLE OF CONTENTS i
LIST OF ABBREVIATIONS v
LIST OF FIGURES vi
ABSTRACT vii

1. INTRODUCTION
1.1. Representative State Transfer (REST) 2
1.2. Application Programming Interface (API) 3
1.3. Objective of the Thesis 3
1.4. Motivation for the Thesis 4
1.5. The Scope of the Thesis 5
1.6. The Structure of the Thesis 5
1.7. Related Work 7

2. THEORY AND LITERATURE REVIEW
2.1. Client-Server Programming Paradigm 8
2.2. Client-Server Programming Paradigm Terminology 9
2.3. Application Programming Interface (API) 10
2.4. Representative State Transfer (REST) 12
2.5. Hypertext Transfer Protocol (HTTP) 13
2.5.1. HTTP Header 15
2.5.2. HTTP Status Codes 16
2.5.2.1. Informational (1XX) 16
2.5.2.1.1. 100 – Continue 17
2.5.2.1.2. 101 – Switching Protocol 17
2.5.2.2. Success (2XX) 17
2.5.2.2.1. 200 – OK 17

2.5.2.2.2. 201 – Created 18

2.5.2.2.3. 204 – No Content 18

2.5.2.3. Redirection (3XX) 18

2.5.2.3.1. 301 – Moved Permanently 18

2.5.2.3.2. 303 – See Other 19

2.5.2.3.3. 307 – Temporary Redirect 19

2.5.2.4. Client Error (4XX) 19

2.5.2.4.1. 400 – Bad Request 19

2.5.2.4.2. 403 – Forbidden 20

2.5.2.4.3. 404 – Not Found 20

2.5.2.5. Server Error (5XX) 20

2.5.2.5.1. 500 – Internal Server Error 20

2.5.2.5.2. 501 – Not Implemented 20

2.5.2.5.3. 503 – Service Unavailable 21

2.6. API Formats 21

2.6.1. JavaScript Object Notation (JSON) 22

2.6.2. Extensible Markup Language (XML) 22

2.7. Functionality and Usefulness 23

2.8. Qualities of a Good API Implementation 24

2.8.1. Data format 24

2.8.2. Project Planning 24

2.8.3. Functions and Function Calls 24

2.8.4. Scalability 24

2.8.5. Documentation 25

3. FATMAN FRAME

3.1. ASP.NET MVC 26

3.2. Model 27

3.3. Controller 28

3.4. View 29

3.5. FATMAN LTD 29

3.6. FATMAN Frame 30

3.6.1. Login Page 30

3.6.2. Dashboard 30

3.6.3. Real Estate Register 31

3.6.4. Service Requests 32

3.6.5. Observations 34

3.6.6. Maintenance 34

3.6.7. Consumption 36

3.6.8. Construction and Technology 37

3.6.9. Documents 38

3.6.10. Contact Info 38

3.6.11. Reports 38

3.6.12. Additional Tools 39

4. FATMAN API

4.1. Authorisation 43

4.2. API Requests 43

4.2.1. Authorisation Request 44

4.2.2. GET 45

4.2.3. POST 45

4.2.4. PATCH 46

4.2.5. DELETE 46

5. DESCRIPTION AND EVALUATION OF FATMAN API'S BUSINESS GOALS 47

6. DESCRIPTION AND EVALUATION OF FATMAN API'S DESIGN GOALS
6.1. Testability 52
6.2. Flexibility 54
6.3. Allowance for incremental exposure of resources and fields 55
6.4. Compatibility with legacy database schema 55
6.5. Measurable Performance 56
6.6. Pricing for the API 56
6.6.1. Data Collection 56
6.6.2. Product Adoption 57
6.6.3. Developer Usage 57
6.6.3.1. Overage Model 57
6.6.3.2. Fixed Quota Model 58
6.6.3.3. Pay-As-You-Go Model 58

7. CONCLUSION AND RECOMMENDATION
REFERENCES 60
APPENDIX 1. FATMAN API Documentation 63

LIST OF ABBREVIATIONS

Platform as a Service	**PaaS**
Infrastructure as a Service	**IaaS**
Software as a Service	**SaaS**
Application Programming Interface	**API**
Representative State Transfer	**REST**
Internet of Things	**IoT**
Universal Resource Locator	**URL**
Hypertext Transfer Protocol	**HTTP**
Request for Comments	**RFC**
Operating System	**OS**
Internet Engineering Task Force	**IETF**
Extensible Markup Language	**XML**
JavaScript Object Notation	**JSON**
Standard Generalized Markup Language	**SGML**
Active Directory	**AD**
Entity Framework	**EF**
Identity Provider	**IDP**
JSON Web Token	**JWT**
Service-Level Agreement	**SLA**
Single Responsibility Principle	**SRP**
Simple Mail Transfer Protocol	**SMTP**
File Transfer Protocol	**FTP**

LIST OF FIGURES PAGE

Figure 1. Thesis Structure 6

Figure 2. Client-Server Architecture 8

Figure 3. API History 10

Figure 4. REST API 12

Figure 5. ASP.NET MVC 25

Figure 6. Building Model in FATMAN API (Partial) 26

Figure 7. Building Controller in FATMAN API (Partial) 27

Figure 8. Fatman API in context 41

Figure 9. Authorisation Request and 200 Status Code on response 43

Figure 10. GET Request showing Authorisation Token, 200 Status Code, and Data in JSON format 44

UNIVERSITY OF VAASA

Faculty of Technology	
Author:	Akintomide Ayodeji Akinsola
Topic of the Thesis:	Evaluation of JSON-API-Format REST API for Business-To-Business Integration Scenarios
Supervisor:	Mohammed Salem Elmusrati
Instructor:	Petri Kortelainen
Degree Programme:	Master of Science in Technology
Degree Programme:	Degree Programme in Communications and Systems Engineering
Major:	Communications and Systems Engineering
Admission Year:	2015
Graduation Year:	2018

ABSTRACT
API technology has brought about changes in the technology industry, especially in the ways that applications interact. It has created a different way for developers to interact with web services, applications, and resources over the internet. This way of interacting with data has spawned many applications and standards that businesses and companies have made use of to improve their products and services. REST is an architectural design that prescribes a set of constraints for interacting with applications and web services. REST is widely used in the API industry and applications that support REST architecture are called RESTful services.

For communication over the internet with web services and applications using REST, information has to be formatted and presented using formats that are efficient, concise, widely-used, and adaptable. JSON format fulfils these criteria. REST API in JSON format is a common technology, and has been deployed to create the API technology, using Entity Framework Core, evaluated in this thesis document.

Fatman Ltd. is a company that produces Facility Technical Management software products. One of the main products is called Fatman Frame. As a response to the demands of the modern industry and customers, an API was created for Fatman Frame. The main subject of this thesis is the evaluation of Fatman Frame API product. This evaluation is done against the business demands and goals of Fatman Ltd., and the customers of the company. The API product is also evaluated against its own design goals. Pricing for the API product is also explored using the pricing systems that are common in the API industry. It is shown in this thesis that the API fulfils its basic design goals, and business demands.

KEYWORDS: REST, API, JSON Format, Fatman Frame, HTTP, ASP.NET MVC, Technology

1. INTRODUCTION

With the rapid advancements in Information Technology, the technological needs of the society have increased. The society has become increasingly dependent on technology for the fulfilment of many aspects of work and living. Moreover, the basic communication need is no longer restricted to human-to-machine communication, we now have machine-to-machine communication. It has been shown that machine-to-machine communication is less prone to error due to parallax, and fatigue.

Many characteristics of machine-to-machine communication have been adapted to make work easier, quicker, and more reliable for the population. The capacity of a machine to interact seamlessly with another machine has also extended the boundaries of technology. The extension of the boundaries of technology comes with the extension of the usefulness of technology. Technology is applied readily to daily, weekly, and annual tasks more efficiently than before. The field of business and financial transactions amongst people, and corporations, has been positively affected by this technology.

Business processes have been improved by the application of technology over the years, from the stock market to word and journal processing. The development and application of technology in business has become an attractive field of research for engineers. The application of software technology to business is of particular interest to engineers, as it assists in the creation and understanding of agile technological systems, and the interaction and adaptation of these systems to the society. In the business world today, many crucial transactions are handled and executed by computer systems that are controlled by software created for those purposes.

The advancement of agile and flexible software technology is therefore of interest to the business world. Also, business transactions in today's highly interconnected and global economy are both location-critical, and time-critical. As such, it is business-critical for the computer systems that are employed in the business world to be efficient, fast, reliable, flexible, and scalable.

1.1. Representative State Transfer (REST)

Representative State Transfer (REST) is an architectural design that establishes a set of properties and constraints for developing web services based on the Hypertext Transfer Protocol (HTTP). Not all web services adhere to this design, but those that do are termed RESTful web services. These web services ensure interoperability of computer systems across the internet. Web services that are compliant with REST allow requesting services to access and manipulate web resources by using a set of pre-defined and stateless actions.

Web resources are defined as distinct entities that can be identified, addressed, renamed, and processed in some way or the other on the internet. These entities on the World Wide Web (www) that can be accessed via a Universal Resource Locator (URL). The term Representative State Transfer (REST) was invented by Roy Fielding in his PhD dissertation of 2000, at the University of California, Irvine. The title of the dissertation was "Architectural Styles and the Design of Network-based Software Architectures". Concerning the PhD dissertation, Fielding said:

> "Throughout the HTTP standardization process, I was called on to defend the design choices of the Web. That is an extremely difficult thing to do within a process that accepts proposals from anyone on a topic that was rapidly becoming the centre of an entire industry. I had comments from well over 500 developers, many of whom were distinguished engineers with decades of experience, and I had to explain everything from the most abstract notions of Web interaction to the finest details of HTTP syntax. That process honed my model down to a core set of principles, properties, and constraints that are now called REST".

There are many advantages to using REST in this way, and one of the them is that it is technically based on HTTP; which is a technology that has been around for a long time and therefore matured. This leads to the technical possibility and technical freedom of choice in determining what technology to use in the implementation of REST. In other words, REST is not bound to any particular technology, but it allows for the freedom of engineers to select whatever technology they are conversant with and implement REST in that technology. As such, REST has become popular amongst software engineers, software developers and software scientists as a means of achieving their technical goals.

1.2. Application Programming Interface (API)

Application Programming Interface (API) is a standard set of agreed-upon protocols, rules, routines, and tools for creating, and building software applications. The API defines the modes and rules of interaction between different software applications and/or components of the same application. Companies and organisations that own software products that can be integrated into other software applications over the internet invest in creating APIs that define the rules of communication between their own software and the foreign software application that wishes to communicate with it. Without this standard, software developers will encounter problems when creating the software applications that wish to communicate with the host software because of a lack of congruency.

API implementation can also be deployed locally in a company. It can be used to allow interaction between different parts of an internal software system. Developers can design an API system that reduces the manual work necessary for the internal functioning of a local system. A good API provides the building blocks that the software developer uses to create a functional integrated software for the host.

1.3. Objective of the Thesis

This thesis aims to document the evaluation of the functionality of an API in an integration scenario from the perspective of two businesses using a REST API approach. It seeks to evaluate the creation process of a standard API for the software called Framework by Fatman Ltd. It follows the procedure for creating and implementing the API, further deployment for the pilot company, and it seeks to assess the performance along defined lines. The defined lines for assessment include: how well the REST API approach solves the business goals of both the company and its customers, and how well the implementation of the API fulfils its own design goals.

This evaluation will be significant in documenting the process of creating and deploying an API. Additionally, it will be useful in instructing the developers of the API, and the customer companies, on what the best practices for this particular software are. Besides, this thesis wishes to define the design goals of the API. These design goals will provide a framework upon which the API will be built. After the API is built and deployed, the product will be tested against these design goals in a bid to investigate whether these goals were fulfilled or not, and to what extent.

Defining the design goals is critical to the process of creating software as it governs the quality of the finished product. In addition, a documentation for the API will be produced alongside this thesis to instruct software developers of client companies on the structure, resources, and process of using the API. The documentation is business-critical for both companies because without it, the task of creating interconnecting software is an arduous one.

1.4. Motivation for the Thesis

The interconnectivity of the modern economy requires that many businesses depend on other businesses or service providers to achieve their business goals. This has led to the need to create robust platforms through which this level of interconnectivity can occur seamlessly. Technology has grown and developed over the year in tandem to fill this role. Many businesses today, as opposed to traditional businesses, rely on technology to provide their clients with the quality of service that they wish to.

As a result of this interconnectivity of businesses and dependency on technology, the need to create platforms to facilitate speedy, seamless, and uninterrupted communication amongst businesses, and their clients has been the driving force of business-side technology. A study of how computer systems are utilised in the business world in the age of a global economy is of importance to computer engineers. Studies like this extend the understanding of computer systems in real life working environment as a consequence of computer systems in laboratory research.

For the businesses that utilise these computer systems, it is also of crucial business importance to have a firm understanding and control over what these systems do, and are capable of doing.

For many businesses, confidentiality and reliability are business-critical selling points, as such, these businesses require systems that support such a position. There are technological problems that become solved as a result of the requirements of businesses. For a software company such as Fatman Ltd., it is critical to monitor the usage of the software that is produced. For many of the customers of Fatman Ltd, there is a need to hire subcontractors who also require access to the software.

There is a need to integrate the different systems already in use by many of the customers of the company seamlessly. One of the ways to ensure this integration is the creation of an API that is capable of facilitating this seamless integration of many systems into Fatman Ltd.'s software. Fatman Ltd. has decided to embark on the creation of this API using one of the latest technologies available. Additionally, an up-to-date documentation is required for the API, to assist the developers on the side of the customers in the development of systems that can integrate with Fatman Ltd.'s software.

1.5. The Scope of the Thesis

This thesis seeks to evaluate the functionality of a particular API implementation in a business-to-business integration scenario. It also seeks to evaluate how well the API approach to integrating many systems solves the business goals of both the host company and the client companies. Furthermore, it intends to evaluate how well the implemented design fulfils its design goals. The design goals are stated later in this thesis. This thesis also seeks to explore the field of API technology and what the technology has achieved and what the future of the technology is.

1.6. The Structure of the Thesis

In this thesis, there are seven chapters. The first one is an introduction to the concept of Representative State Transfer (REST), and the Application Programming Interface (API). It also contains the objective, motivation, and scope of the thesis. The second chapter contains the literature review and background for the thesis. It also contains the theoretical background, explanation of key concepts, and the exploration of technical concepts. This will assist in providing the needed information for the work described in chapter three.

In chapter three, this thesis presents the software product of Fatman Ltd., and the essential aspects of it that the thesis explores. This thesis will only present the aspects of the software that are related to the thesis itself. In chapter four, the implementation of the API is discussed and explored exhaustively. Chapter five evaluates the API product based on the business goals and objectives of the company and its customers. Chapter six evaluates the API product based on its own design goals. These goals and objectives will also be stated in these chapters. Chapter seven presents the conclusion of the work done and recommendation for future research.

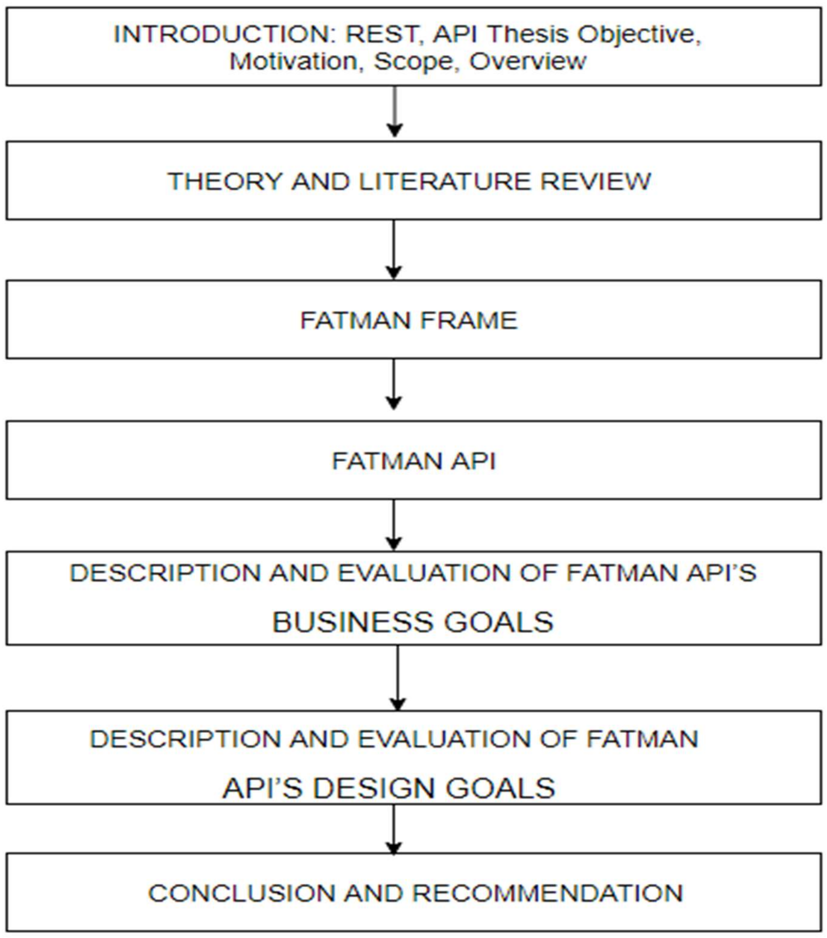

INTRODUCTION: REST, API Thesis Objective, Motivation, Scope, Overview
THEORY AND LITERATURE REVIEW
FATMAN FRAME
FATMAN API
DESCRIPTION AND EVALUATION OF FATMAN API'S BUSINESS GOALS
DESCRIPTION AND EVALUATION OF FATMAN API'S DESIGN GOALS
CONCLUSION AND RECOMMENDATION

Figure 1. Thesis Structure.

1.7. Related Work

In today's era of high-speed interconnectivity and the global economy, it has become critical for many businesses to have properly designed and implemented APIs to participate in the global economy. Suffice it to say that this is not the first time in history that any company has decided to create an API product for their software. In reality, this is the usual practice for software companies in the business of selling manufactured software to willing customers. The APIs of many companies are publicly available for business reasons.

2. THEORY AND LITERATURE REVIEW

As a result of the technological advancement of the past decades, the importance of instant or near-instant connectivity has become a leading topic in the field of communication. The speed at which data is processed, and the ease of data access, are crucial to the daily operations of companies, organisations, and individuals, as discussed in the introduction. There are companies today that were started to research and solve the business-critical issues of data access and information processing. Businesses, organisations, and individuals have also embraced cloud services as a means to solve these problems. Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Software as a Service (SaaS) are some examples of how companies have moved to cloud-based services to solve these issues. For some companies that host customer data, a third-party solution is not optimal, for legal reasons or otherwise. (Williams 1990: 40-45).

Technological advancements in Mobile telephony, spurred by the advancement in the capabilities of smartphones, has been a catalyst for the advancement of API technologies. The need of smartphones to get data to and from databases and backend devices has increased as the usage of mobile devices has become more sophisticated. The advent of Internet of Things (IoT), mobile apps, and the improvements in responsive web applications are altogether another reason that responsive APIs are becoming necessary for companies, individuals, and organisations who wish to tap into the internet economy. Over time, APIs have been consistent in providing a simple and relatively easy-to-implement conduit for data and information. In their implementation, APIs have a lightweight data/information exchange format. (Williams 1990: 40-45).

2.1. Client-Server Programming Paradigm

The Client-Server paradigm or model is the distributed application architecture whereby the job of completing a request is split into two: between the computer systems dedicated to making a request called “clients” and the computer systems dedicated to responding to such requests called “servers”. A client and a server may be located in the same network and need to communicate over the intranet or they may be separated by a large distance and may therefore need to communicate over the internet. Clients typically initiate the request for a service or function from the server and wait for the response of the server. Usually a service is an abstraction of computer resources and the workings of the server to fulfil the requests of the client is opaque to the client. Examples of systems that utilise this paradigm include: online banking, email, online gaming, and web services. (Deitel & Deitel 2010: 1144 – 1178).

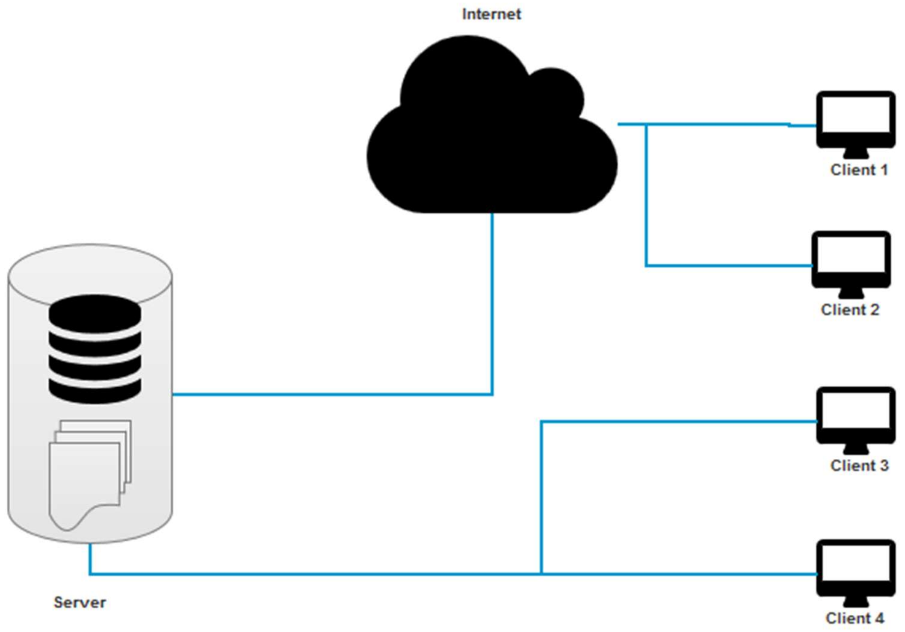


Figure 2. Client-Server Architecture

2.2. Client-Server Programming Paradigm Terminology

1. Broadcasting means sending a request to all the computers on a given network. This may occur when the server wishes to find out all the clients connected to it, it may send a broadcast and receive information from all the clients that accept the broadcast.

2. A connection is a network link between two machines; a typical connection allows the flow of data in both directions. The close end of the connection is called local, whilst the far end is termed remote.

3. A socket is any of the two ends of a connection. Sockets have unique ID numbers. Sockets are decided and assigned by the Operating System (OS). Each socket has two ports, an input port for reading what is received from the network/server, and an output port for writing what is needed to be transmitted to the network/server. Whatever is written to a socket's output port will be sent through the connection to the remote socket's input port.

4. A request is sent over the network in order to cause an action and/or receive a response. Requests have source, destination, command, and data fields.

5. A callback is a procedure that is triggered in response to an event. For instance, when a client's port connected to a specific server receives data, it calls the request-handler procedure.

(Berson 1992).

2.3. Application Programming Interface (API)

An Application Programming Interface (API) is a set of functions or procedures used by computer programs to access operating system services, software libraries, or other systems. It can also be defined as a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service. An API is tasked with the function of facilitating the exchange of data/information between two applications or web services across the internet. It is a set of functions and procedures that offers a library so other software can use it as an abstraction layer; an area to access and exchange additional information. Thus, all clients have access to common data and information without compromising common independence. (Chen, Annadata & Chan 2001).

The API is a versatile technology because the design is not restricted to any particular programming language or technology; it can be done in one of the many programming languages available. APIs include specifications for data, routine structures, classes and attributes of objects/variables; this is the backbone for the usage of the provided API interface. Additionally, the developers of an API platform provide comprehensive, efficient, and useful documentation for the specific API that details the components of the API, and how it must be used. This is important to developers who wish to create an app, software, or any such platform that will consume the functionality of the API. The ease with which other developers can create software for an API is also a measure of its efficiency, as well as a pointer to the limits of its acceptance in the development landscape. The easier it is to create a software for an API, the more developers will be willing to do so. The ease of use of an API is of important business interest and advantage to the company providing such an API. (Chen et al. 2001).

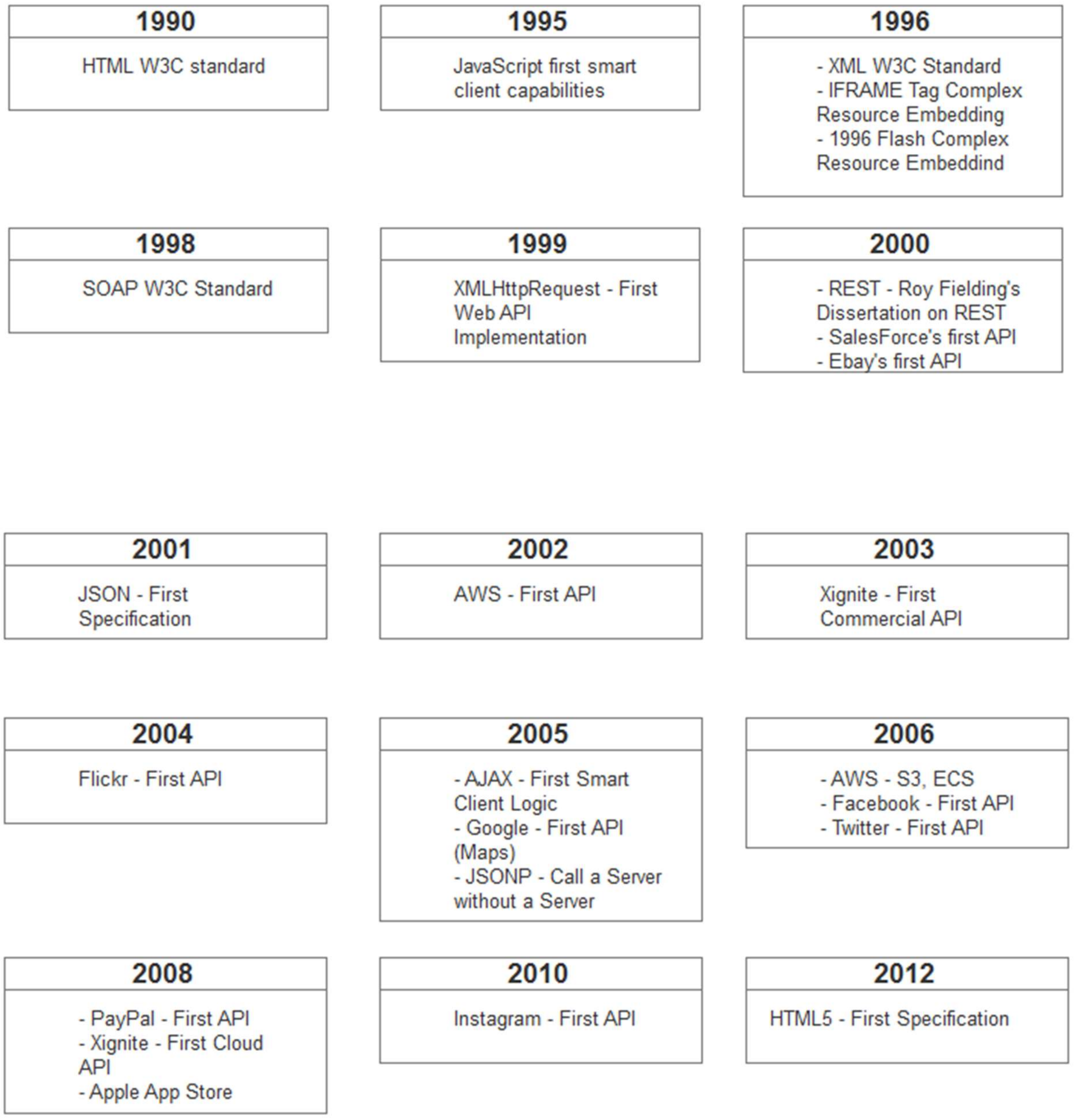

1990
HTML W3C standard
1995
JavaScript first smart client capabilities
1996
- XML W3C Standard
- IFRAME Tag Complex Resource Embedding
- 1996 Flash Complex Resource Embeddind
1998
SOAP W3C Standard
1999
XMLHttpRequest - First Web API Implementation
2000
- REST - Roy Fielding's Dissertation on REST
- SalesForce's first API
- Ebay's first API
2001
JSON - First Specification
2002
AWS - First API
2003
Xignite - First Commercial API
2004
Flickr - First API
2005
- AJAX - First Smart Client Logic
- Google - First API (Maps)
- JSONP - Call a Server without a Server
2006
- AWS - S3, ECS
- Facebook - First API
- Twitter - First API
2008
- PayPal - First API
- Xignite - First Cloud API
- Apple App Store
2010
Instagram - First API
2012
HTML5 - First Specification

Figure 3. API History.

For an API to exist, there has to be a software, service, or application that has some functionalities that need to be exposed via an API implementation. The API must be created with a thorough understanding of the parent software and its functionalities. This leads to better decisions on the aspects of the parent software that need to be exposed, in what manner they should be exposed, and which ones do not need to be exposed via the API at all. Additionally, the functionality that is being exposed through the API must be modelled in such a way that it addresses all real-life use cases. Typically, the application/software that an API is to be created for contains an application state. An application state is the data repository available to all classes in an application; it refers to information available to all the users and sessions of the application at any given time. Application state is stored in memory on the server and is faster than storing and retrieving information from a database.

2.4. Representative State Transfer (REST)

Representative State Transfer (REST) is a software architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system. (Kholod 2017). It can otherwise be defined as an architectural design used in software development that outlines a set of constraints for the creation of web services. REST was created to standardise the rules that different software systems use to interact with each other over the internet to improve interoperability. In the implementation of REST, the specifics of the protocol's syntax and component implementation are neglected so that component roles, the constraints on their interaction with other components in the same system or another system, and the interpretation of data elements can be focussed on. (Massé 2012: 5-6).

REST uses a technique that is similar to what browsers use, in the sense that resources are accessed via addresses similar to URLs. A Universal Resource Locator (URL) is the pointer to the location of a certain web resource on a computer network and the method for fetching it. A URL is otherwise called a web address. With REST, resources are located by different programs by their URL; web resources usually have a web address by which they can be located and interacted with. Typically, a computer system or software sends information to a particular address that contains a web resource using the protocol called HTTP. In this exchange of information, the initiator system/software is allowed to either get data from the resource or send data to the resource using methods that are defined in HTTP. An API implementation that follows the REST design is called a RESTful API. (Massé 2012: 5-6).

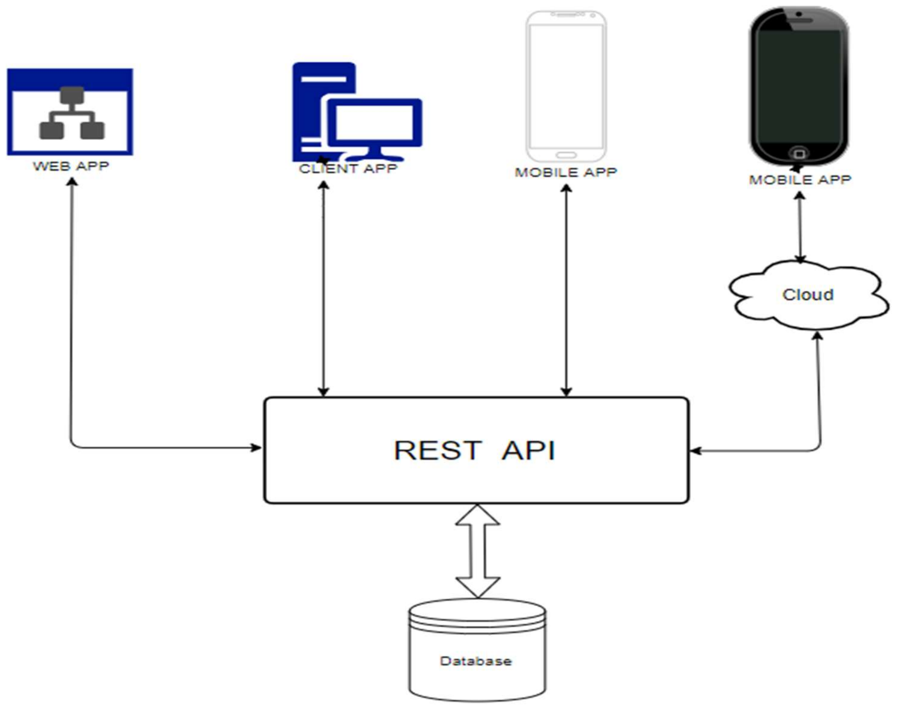


Figure 4. REST API.

2.5. Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems. Data communication over the internet is based on HTTP. Hypertext is defined as text structured in such a way that it contains logical links called hyperlinks amongst nodes that contain text. In essence HTTP is used in the transfer/exchange of hypertext. The first definition of HTTP to gain popular usage was finalised in 1997 in RFC 2068. Request for Comments (RFC) is a publication from the engineering community in memorandum form that describes the behaviours, methods, research, and innovations that can be applied to the internet itself and appurtenant systems. HTTP defines methods that are used to interact with resources over the internet. Amongst these methods are a few that are used in this thesis: GET, POST, DELETE, PATCH. (Totty, Gourley, Sayer, Aggarwal & Reddy 2009: 7-10).

The HTTP GET method is the main method that contains the mechanism to retrieve information from a target web resource. This method is used to fetch the current state/representation of the indicated properties of a targeted web resource. A client program can edit the properties of the targeted web resource according to its implementation or necessity. This means that the GET method is not a static method, it can be edited to take parameters so that the fetched data is tailored to the client's specification, and that specification can change with time or situation. This modification to the requested data can be done in the "range" header field. The "range" header field is used to set the range of the request – this is the range of data required by the client program at a given time. The data fetched by the GET method can be cached reused as needed, unless otherwise specified, by the Cache-Control Header field. (Fielding & Reschke 2014: 24-25).

The HTTP POST method is used to send data to a target web resource with the request that the resource processes it in an indicated manner or in accordance with the resource's own specific semantics. This method can be used to supply a range of data, like the filled fields of an HTML webform, to the data-handling process. It can also be used to create a new web resource that the origin server does not yet have or to append data to an existing web resource. The response to a POST request is indicated by the server by selecting and sending the appropriate status code that the specification of the server has set for such a situation. The result of a POST method can also be cached. (Fielding & Reschke 2014: 25-26).

The HTTP PATCH method is used to apply partial modifications/changes to a web resource. With PATCH, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version. The PATCH method affects the resource identified by the Request URI, and it may also have side effects on other resources; i.e., new resources may be created, or existing ones modified, by the application of a PATCH. The PUT method is used to update a web resource by replacing the instance of that resource with a new one defined in the PUT request. The difference between the PUT method and the PATCH method is that the PATCH method is used to only update specific parts of a web resource and not the entire resource. This is advantageous over the PUT method in many ways e.g. it saves bandwidth and it is less cumbersome for the server to handle. (Dusseault & Snell 2010: 2-4).

The HTTP DELETE method is used to make request to the server to discard the association between the target web resource and the its present functionality. The complete removal of the resource by the DELETE method from the server depends on the type of resource it is and the implementation of the method on the server. The server will only implement the DELETE action if it is designed to do so. This is a rarely implemented method as it requires the removal of a resource, therefore the client is usually advised to not use this method unless it is necessary so to do. Depending on the particular implementation, the DELETE request may return the appropriate status code to the client. A payload within a DELETE request message has no defined semantics; sending a payload body on a DELETE request might cause some existing implementations to reject the request. Responses to the DELETE request cannot be cached. (Fielding & Reschke 2014: 29-30).

2.5.1. HTTP Header

The HTTP messages in the requests and responses that are transmitted between a server and a client are enclosed in the HTTP header. A completed cycle of request and response between a client and a server is called a transaction. In a HTTP transaction, the header defines the operating parameter for that specific transaction. In a HTTP header, there are various fields in which different types of useful information relating to the transaction at hand are transmitted. These fields are called HTTP Header Fields. In an HTTP transaction, the header fields are sent after the request line – in the request HTTP message – or the response line – in the response HTTP message – which is the first line of a message. Header fields are colon-separated key-value pairs in clear-text string format, terminated by a carriage return and line feed character sequence. The end of the header section is indicated by an empty field/line resulting in the transmission of two consecutive carriage return-line feed pairs. (Massé 2012: 35-38).

Field names in HTTP headers are decided by the Internet Engineering Task Force (IETF). There are core fields that are mandatory, and these are set by IETF (in RFC 7230, 7231, 7232, 7233, 7234, and 7235), there are also optional fields that can be utilised according to the requirements of the application. There is the option of adding comments to some fields by design, and these comments are designed to be ignored by both the server and the client during a transaction. Although the standard does not mandate size limits for the header field names, value, or the total number of fields allowed, for security and practical reasons, clients, proxy software, and servers mandate a limit. For instance, the Apache 2.3 server by default limits the size of each field to 8,190 bytes, and there can be 100 header fields at most in a single request. (Massé 2012: 35-38).

2.5.2. HTTP Status Codes

HTTP Status Codes are shorthand messages sent from a server to a client with every response that the server transmits. These messages contain useful information concerning the response the server is sending that may be of benefit to the client, and also to the user of the client – especially a developer. Typically, these messages are transparent to the user if there is no special need to view them. For the developer of an API, or the user of the API, understanding Status Codes is crucial to the proper development of an API implementation. The messages contained in these codes assist in explaining precisely what is going on between the server and the client in terms of requests and the responses to those requests. Without these codes, it becomes difficult to trace and track problems when they occur. HTTP Status Codes are sent to the client in the HTTP header.

In total, there are over 40 distinct server status codes. HTTP status codes are classified into 5 distinct categories. These category identification tags supply information to the developer at a glance. REST APIs use the status line field of the HTTP response message to pass information to the client concerning the result of the request being responded to. These categories are listed as follows:

2.5.2.1. Informational (1XX)

This category is used to communicate information on the transfer protocol level. It is used to state to the client (and user) that the information that was sent was received, understood, and that further processing of such information is in progress. This is also partly an indication to the client to wait for the final result of a request. This message is made up only of elective header fields, and the status line. Some instances of this category are as follows.

2.5.2.1.1. 100 – Continue

This status code is sent as an indication to the client that the server has received and accepted the request header. This tells the client to send the request body. The splitting of this request into header and body is to ascertain that the request header fulfils the requirements of the server before the request body is sent. This occurs in the case of the HTTP POST. In the scenario of an HTTP POST, the server receives the HTTP header first and confirms that the parameters are acceptable before the huge request body is sent by the client. This increases the efficiency of the system by reducing the traffic per transaction.

2.5.2.1.2. 101 – Switching Protocol

This status code indicates to the client that the server as received and accepted the request to switch application protocols made to it by the client using the Upgrade header field. On sending this status code, the server generates an Upgrade header field that indicates the new application protocol switched to in response to the client.

2.5.2.2. Success (2XX)

This category of status codes indicates the successful completion of a particular task. The server returns these codes to indicate to the client that the request it sent to the server has been received, accepted, understood, and processed successfully. A few instances of this category are described below.

2.5.2.2.1. 200 – OK

This is the status code returned by the server when the request is successful. Status code 200 always returns with a payload, unless the request was a request to connect. The server is at liberty to generate a payload of zero length if there is no data to return. Status code 200 is cached by default, unless explicitly otherwise indicated by cache controls.

2.5.2.2.2. 201 – Created

This status code is returned to indicate that the request has been completed and that one or more resources have been created as a result. An indication of the location of this newly created resource is also sent with the payload. The payload returned is dependent on the type of request itself – GET, POST, PATH, or DELETE. It could be a URI or in the Location header field.

2.5.2.2.3. 204 – No Content

This status code is returned in the event that the server has accepted, understood, processed, and fulfilled the requirements in the request but there is no additional data to return to the client in the payload body.

2.5.2.3. Redirection (3XX)

When this category of status codes is sent, it is an indication that the client is required to take further action before the request can be completed. In the instance that the Location header field is supplied, the server may choose to redirect the client to the URI in the Location header field. This redirection must necessarily be done with specific security measures for methods that are not safe. A few instances of this category are highlighted below.

2.5.2.3.1. 301 – Moved Permanently

This is sent to indicate to the client that the resource being requested for has been assigned/moved to a new permanent URI. This also indicates that future references to that specific resource should be made to the new URI. A 301-response status code can be cached by default, unless otherwise explicitly indicated by cache controls, or the method definition.

2.5.2.3.2. 303 – See Other

This is the status code returned when the server redirects the client to a different resource from the one indicated in the request. This intends to provide an indirect response to the original request. A 303 response to a GET request indicates that the origin server does not have a representation of the target resource that can be transferred by the server using HTTP.

2.5.2.3.3. 307 – Temporary Redirect

The server returns this code to indicate to the client that the resource being targeted resides temporarily at a different URI. The new URI can be indicated in the Location field of the response. In some instances, the temporary address of the targeted resource may change with time. In such cases, the client is advised to continue to use the original address of the resource so that the server can always redirect to the proper temporary address that changes with time.

2.5.2.4. Client Error (4XX)

In this category of error, the server indicates to the client that an error has occurred in the request which is the fault of the client. In the response from the server, an explanation of the error is also sent, and information concerning whether it is temporary or not. The client should display any included representation to the user. Additionally, the client is expected to make modifications to the request before resending it.

2.5.2.4.1. 400 – Bad Request

This response is returned to the client to indicate that the server will not or cannot process the request as a result of an error in the request. The error might be as a result incorrect request syntax, invalid framing of the request message, or wrong request routing, amongst other examples. Listed below are a few examples of the error codes in this category:

2.5.2.4.2. 403 – Forbidden

This status code is returned if the server gets a request, understands it, but cannot authorise it. The server is allowed to indicate the reason for not authorising the request in the response payload sent to the client if there are any.

2.5.2.4.3. 404 – Not Found

This status code indicates that the server was unable to find the present representation of the target resource. This could also mean that, for some reason, the server is unwilling or unable to disclose that a current representation of the target resource exists. This status code is cacheable by default.

2.5.2.5. Server Error (5XX)

In this category of error, the server indicates to the client that an error has occurred in the request which is the fault of the server. This is also sent when the server is incapable of performing the requested action for any reason. The server has the responsibility of sending the explanation for the situation to the client, and whether it is temporary or not, unless it is a HEAD request. This status code is universal, i.e. it can be used for any method.

2.5.2.5.1. 500 – Internal Server Error

This indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

2.5.2.5.2. 501 – Not Implemented

This indicates to the user that the server is technically incapable of processing the request. The reason for this technical incapability can be that the functionality required to fulfil the request is not supported by the server. This is also the status code returned if the server does not recognise the request method and therefore cannot support it for any of its resources.

2.5.2.5.3. 503 – Service Unavailable

This is the status code returned to the client if the server is presently unable to process the request due to a temporary situation. This situation can be scheduled maintenance, temporary server overload, etc. This situation typically improves after a short time and the server can continue to handle requests normally. The server does not automatically send this code in these situations, some servers simply refuse the connection if they are in any of the situations stated above.
(Fielding & Reschke 2014: 50-64).

2.6. API Formats

As discussed in the preceding paragraphs, the API works in such a way that a client sends requests over a network to a server, which in turn replies to the request with either a confirmation or a denial given with shorthand called status codes. This request could be DELETE, POST, GET, or PATCH. These requests are for actions to be performed on resources located on the remote server. The question about what the client sends and what the server receives has been answered by the preceding paragraphs. What is left to answer is how the client sends the messages. In other words, what format is used in sending and receiving messages between the client and the server. In API technology, many different formats have been created over the years to standardise the use of APIs across networks and platforms. When designing an API, an important factor to pay attention to is what the format must be. Data formats are very critical to the performance of any API implementation. The data format describes how the handling of the interaction between data generation in the server and data request in the client is done. (Massé 2012: 47-69).

Since an API is essentially a technological instrument used in connecting two or more disparate services, users, and sources across the internet, it is critical that all the users are capable of understanding each other explicitly. As such, these systems are required to synchronise the template of their messages. The format of the API is independent of the type of service that the interacting systems are geared towards. The type of connection between any two users or interacting parties must be designed in such a way that it is usable to the requesting client and makes recognising the data being presented as easy as possible. The choice of format for any API implementation determines the effectiveness of the API, it affects the throughput of routine and specific calls, use rates, adoption of the API in the long term, and the curve of retention of the lifecycle of the implementation.

2.6.1. JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format. It was derived from the ECMAScript Programming Language Standard. JSON defines a small set of formatting rules for the portable representation of structured data. JSON is designed to represent all four primitive data types: integer, Boolean, null, and string. It is also designed to represent the structured types: array, and object. By definition, a string is a sequence of one or more Unicode characters. An object is an unordered collection of zero or more name-value pairs, where a name is a string and a value is a string, number, Boolean, null, object, or array. An array is an ordered sequence of zero or more values. (Bray 2017: 4-5).

JSON API format is specified based on the JavaScript programming language, as such, it is best suited to API implementations is JavaScript or such languages. It is also an open-standard file format. The JSON media type takes the form "application/json". Moreover, JSON is used to transmit array data type data objects, or data objects that have attribute-value pairs. The attribute (also known as key) is a property of the object being described. For example, if the object is a car, it has an attribute/key called manufacturer, and the corresponding value in this instance might be BMW. Browsers that support the 5th edition of the ECMAScript are capable of parsing JSON. JSON also supports Associative Arrays – this is a situation where the value of a key is a nested key-value pair. (Bray 2017: 4-5).

2.6.2. Extensible Markup Language (XML)

Extensible Markup Language (XML) is text format that is designed to reduce the complexity and challenge of electronic publishing on a large scale. It is now used to exchange wide variety of information in the form of data, especially on the worldwide web (www). This particular format is flexible and simple to use. It is also both machine-readable and human-readable. XML is based on the Standard Generalised Markup Language (SGML) therefore the XML formatted documents are by nature compliant with the SGML standard. SGML is the standard that is used to create document structures for documents that are intended for use on the internet. XML is a concise subset of the SGML. XML can be used to represent customised data structures even though its design focuses on documents. Additionally, XML supports Unicode for many languages. (Bray, Paoli, Sperberg-McQueen, Maler & Yergeau 2013)

XML documents are made up of storage units called “entities” which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure. Some important constructs of XML are defined thus: Characters are the basis of XML, and they are Unicode compliant. The XML Processor is utilised in parsing the XML documents and in the provision of access to the content and structure therein. The XML Processor transmits this structured data to the Application. Tags are included in XML as contructs that enclose the information in an organised manner, beginning with opening symbol “<” and closing symbol “>”. Elements are the characters that are enclosed within an opening tag and a closing tag. (Bray et al. 2013; Nurseitov, Paulson, Reynolds & Izurieta).

2.7. Functionality and Usefulness

The functionality of an API is determined based on whether it fulfils its design goals or not. Functionality can be defined as the assurance that an API implementation will deliver data when requested, and in the predetermined usable format. A functional API system can thus be defined as a system that operates as designed, and is usable by the system making the request. The usefulness of an API implementation is determined by the systems that utilise it. For an API to be useful, it is required to be convenient for the developers that will implement services that utilise such an API system. Long function names, poor documentation, and parameter names that are difficult to understand go against the usefulness of the API system. A useful API system should also have a built-in capacity to evolve. Since today’s business environment is such that evolves rapidly, this should be considered during the design of an API system. Scalability and extensibility are key concepts to be considered and built into the system from the beginning to avoid creating a rigid business environment. (Sandoval 2015).

2.8. Qualities of a Good API Implementation

For an API implementation to pass the quality assurance test, it must possess the following attributes:

2.8.1. Data format

The format that data and information will be transmitted in is critical to the performance of the API Implementation. Data organisation in a way that's clear and can be understood by other systems increases the usefulness of the API and enables extensibility.

2.8.2. Project Planning

Proper planning of the stages of the API development from the outset is very important. The developers who will use the API implementation in the future must have a comprehensive understanding of how the API progresses and grows. Proper planning connects the development into logical and easy-to-understand units which eventually grow into a whole.

2.8.3. Functions and Function Calls

Functions should be written in the simplest terms possible for easy comprehension. Function names, parameter names, and function calls should be explicit. This makes future development and accessibility easier for both the API developer and the developers utilising the functionalities of the API.

2.8.4. Scalability

The API must be implemented in such a way as to make future scaling possible and simple. The uses of the API in the future must be put into consideration from the start to allow for the implementation of the best practices. As an instance, if the API deals with a large amount of slow-processing data but may need to implement features in the future that require high security, SOAP is the better option than REST.

2.8.5. Documentation

The documentation of the API is just as important as the implementation itself. The documentation must be explicit, comprehensive, and current.
(Sandoval 2015).

3. FATMAN FRAME

Fatman Frame is designed based on Microsoft's ASP.NET MVC technology. A short introduction to the technology is described in the following sections.

3.1. ASP.NET MVC

ASP.NET MVC is a Web development framework from Microsoft that combines the effectiveness and tidiness of the model-view-controller (MVC) architecture, the most modern techniques and ideas from Agile Development, and the best parts of the existing ASP.NET platform. ASP.NET MVC framework was announced by Microsoft in 2007, it was intended to be a web development platform built on the existing and mature ASP.NET. MVC is an established architectural pattern, it was introduced in the Smalltalk project from XEROX PARC in 1978. ASP.NET MVC implements the most modern techniques in the MVC architecture. It was created by Microsoft in response to such technologies as Ruby on Rails, and Node.js. Additionally, since modern development implements the Agile Software Development techniques, it was imperative to introduce a framework that is compatible with these techniques. (Adam Freeman 2013).

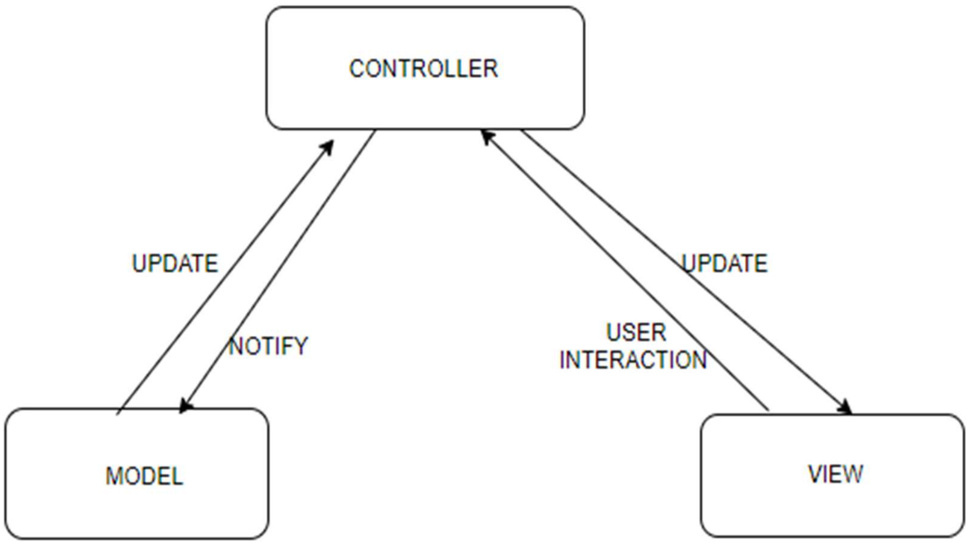
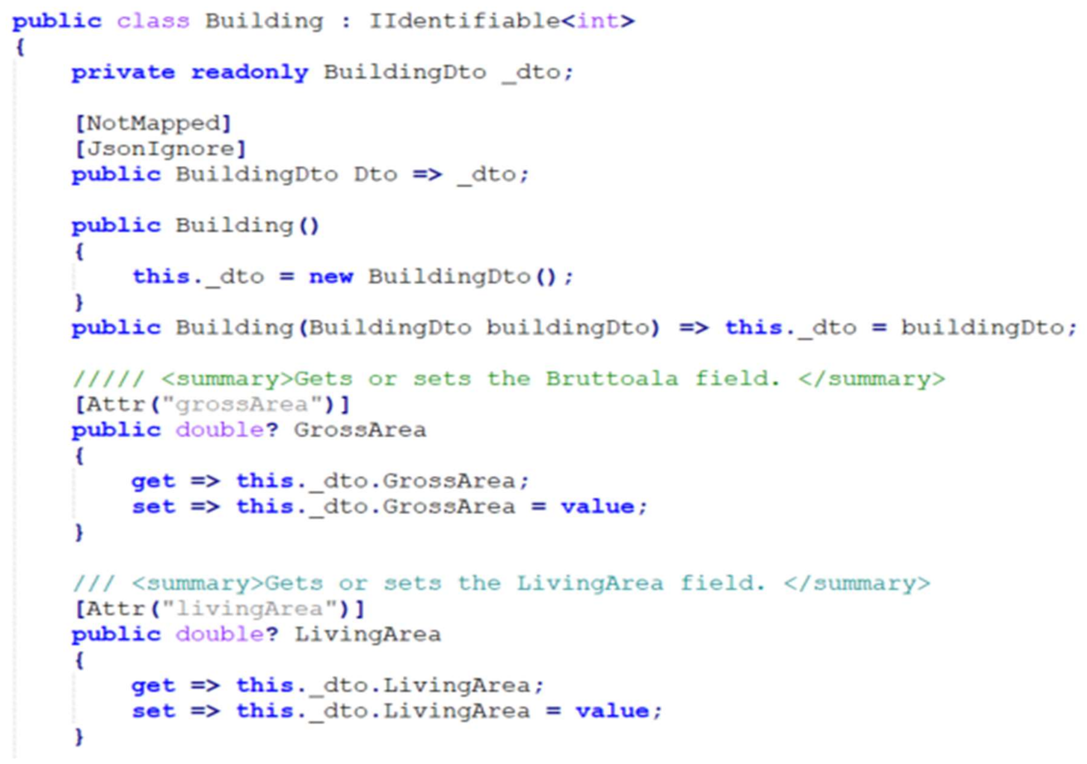


Figure 5. ASP.NET MVC

3.2. Model

In the model-view-controller architecture, the Model represents the data as it is stored in the database. The Model is a standalone aspect of the architecture that does not depend on the other two to function. It is independent, and static. Typically, the model represents an object as it is in real life. It is only dynamic in that the state, parameters, values, of the object may change, but not the object itself. It can also be used to represent a stage in a serious of actions that may be called a task. A single task in a series of tasks can be represented with a model. In this way, the necessary parameters that qualify as a task are coded into the model. A model is a person, a request, a device, or any other real-world object, depending on the needs of the application being made. Both the view and the controller in the model-view-controller architecture depend on the model to manipulate data and present it to the user. In model-view-controller architecture, the model naming convention is important, and the file usually ends in ".cs" in ASP.NET framework. (Adam Freeman 2013).

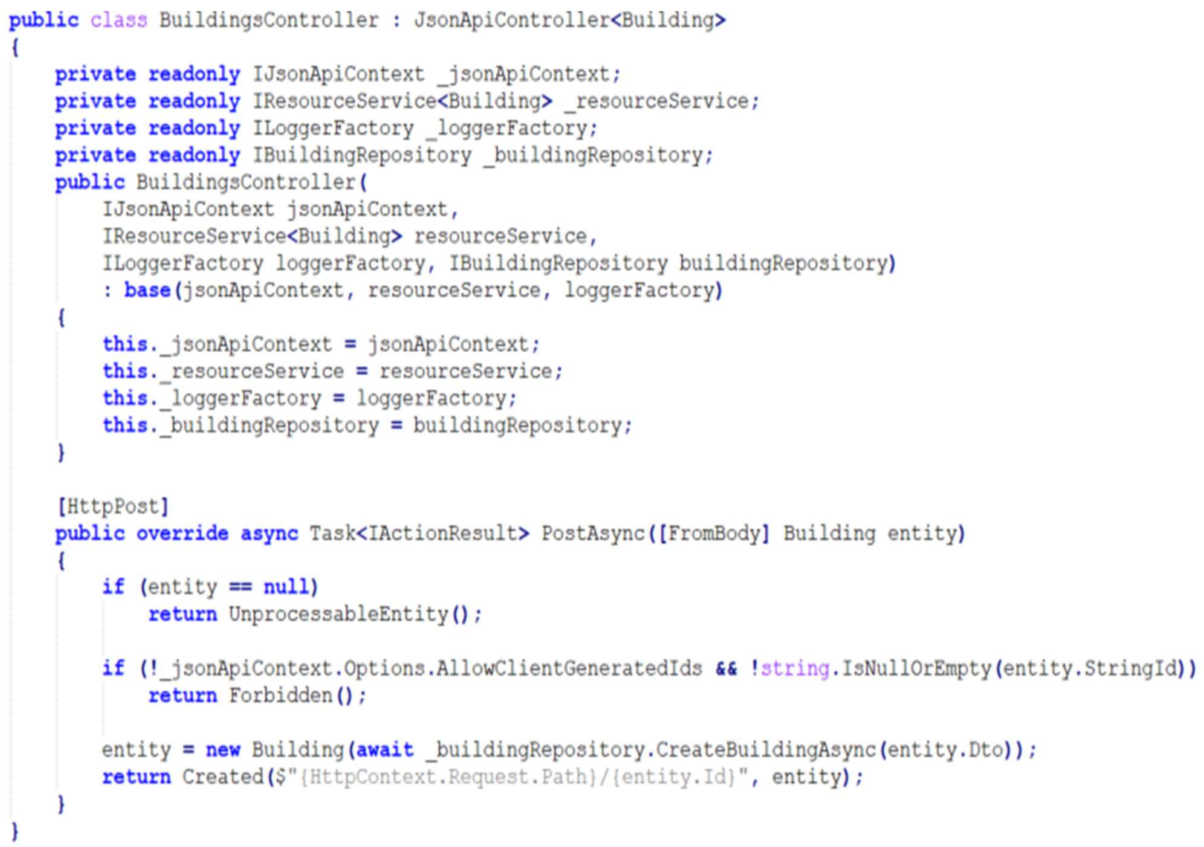


```
public class Building : IIdentifiable<int>
{
    private readonly BuildingDto _dto;

    [NotMapped]
    [JsonIgnore]
    public BuildingDto Dto => _dto;

    public Building()
    {
        this._dto = new BuildingDto();
    }
    public Building(BuildingDto buildingDto) => this._dto = buildingDto;

    ///// <summary>Gets or sets the Bruttoala field. </summary>
    [Attr("grossArea")]
    public double? GrossArea
    {
        get => this._dto.GrossArea;
        set => this._dto.GrossArea = value;
    }

    /// <summary>Gets or sets the LivingArea field. </summary>
    [Attr("livingArea")]
    public double? LivingArea
    {
        get => this._dto.LivingArea;
        set => this._dto.LivingArea = value;
    }
```

Figure 6. Building Model in FATMAN API (Partial)

3.3. Controller

The controller in the model-view-controller architecture is the stage responsible for data reception, manipulation, and extraction, from the user and the database. The controller is also responsible for presenting data to the view. The controller responds to promptings from the user in terms of events handling, such as button clicks. The controller is not independent in this architecture, it depends on the view and the model for data representation and data presentation. The controller is versatile and can be deployed for data and user validation; this can be designed and coded to occur in the controller. The controller is also responsible for arranging the view and pointing to the right view to display the appropriate information. The controller is the intermediary between the model and the view, is such a way that both cannot perform any tasks without those tasks being explicitly stated in the controller. In ASP.NET, the controllers are explicitly named "controller" in the codebase, and they are also ".cs" type files. (Adam Freeman 2013).



```
public class BuildingsController : JsonApiController<Building>
{
    private readonly IJsonApiContext _jsonApiContext;
    private readonly IResourceService<Building> _resourceService;
    private readonly ILoggerFactory _loggerFactory;
    private readonly IBuildingRepository _buildingRepository;
    public BuildingsController(
        IJsonApiContext jsonApiContext,
        IResourceService<Building> resourceService,
        ILoggerFactory loggerFactory, IBuildingRepository buildingRepository)
        : base(jsonApiContext, resourceService, loggerFactory)
    {
        this._jsonApiContext = jsonApiContext;
        this._resourceService = resourceService;
        this._loggerFactory = loggerFactory;
        this._buildingRepository = buildingRepository;
    }

    [HttpPost]
    public override async Task<IActionResult> PostAsync([FromBody] Building entity)
    {
        if (entity == null)
            return UnprocessableEntity();

        if (!_jsonApiContext.Options.AllowClientGeneratedIds && !string.IsNullOrEmpty(entity.StringId))
            return Forbidden();

        entity = new Building(await _buildingRepository.CreateBuildingAsync(entity.Dto));
        return Created($"{HttpContext.Request.Path}/{entity.Id}", entity);
    }
}
```

Figure 7. Building Controller in FATMAN API (Partial).

3.4. View

The view in model-view-controller architecture is responsible for present information of just data to the user. It serves as the user interface component of the architecture. The view gets information from the model using the controller and displays it to the user according to the design present in the view. It can also be used to update the model with data received from the user. The view depends on the controller and the model. It has knowledge of the model and is designed to use specific controllers explicitly. Multiple views can be displayed on the same screen, and in multiple formats, including diagrams, tables, pictures, and text. In ASP.NET the views are ".cshtml" extension files, and they can be written in HTML and C# syntax. They are also CSS and JavaScript-compliant. (Adam Freeman 2013).

3.5. FATMAN LTD

According to Fatman Ltd.'s website, the company provides modern, easy-to-use, and mobile-optimised applications to increase intelligence in life cycle management. With the help of our products, our clients have enhanced their businesses for over 25 years. We work with our clients to increase the profitability and value of their assets. At Fatman Ltd., we present many different solutions to the challenges facing our clients in the field of real estate management. These solutions include Fatman InfoMaster, Fatman Contract Management, Fatman eCenter, and Fatman Frame – which is the focus of this thesis. Fatman Ltd. was founded by Kari Hein. Kari Hein served as a pioneer in the development of the Finnish Property Management business. He gained experience in, among others, the North American real estate market. During the years spent in North America, Hein remembers the ideas of new, innovative and pioneering real estate management tools and operating models. It could be said that electronic real estate management solutions landed in Finland on the same plane as Mr. Hein himself.

The foundation of our operations today is the development of the most advanced technology, and the most knowledge-intensive software in the industry. Since our foundation, we have been committed to being a reliable expert and responsible partner to our customers, not just a tool vendor. That is why the precise needs of our customers are at the heart of our product and service development. We create and offer solutions that give our customers tangible benefits, and increase their business productivity. The name of our company, FaTMan, is an abbreviation of the term Facility Technical Management which explains the business that we are involved in as a company at a glance.

3.6. FATMAN Frame

As the focus of the thesis is Fatman Ltd.'s Fatman Frame - hereafter referred to as “Frame” - the product shall be described in relevant detail in the following sections and subsections. Fatman Frame is designed using the ASP.NET MVC architecture and the latest Microsoft technologies in C#. Since Frame is a facility management software, it is designed to be based on the basic structure of an architectural building. As such, Frame can only work if a building has been created beforehand. If the user is new, Frame prompts the user to create a building during the first log-in. This building is then preselected for Frame to work, by design.

3.6.1. Login Page

Frame is a web-based software; therefore, it can be accessed by using certain customised URLs. For the production site, the URL is origo.fatman.fi. This URL leads immediately to the login page where the user can only advance by entering the appropriate and verifiable credentials. The front page also allows users to enter their Windows Active Directory (AD) credentials by supplying a button that prompts the user to enter those credentials.

3.6.2. Dashboard

After logging in, the software displays the dashboard pre-set for the specific user. The dashboard is designed to have a navigation bar on the left of the screen. This navigation bar contains a list of all the modules that the user has access to, and they are listed in no particular order. As stated above, it is necessary that a building is selected when the user logs in, and Frame does this building selection automatically; Frame typically selects the last building operated on by the user which is stored in browser cookies. Additionally, the dashboard, apart from the navigation bar, contains such useful features as widgets. These are applications that are designed to display information about a specific aspect of the software to the user at a glance. The dashboard widgets are customisable, and include graphs, maps, tables, charts, notifications, etc., depending on the setup of the user and the choice of widgets.

On the upper part of the dashboard, Frame displays the building selection and typically also the facility selection. A facility as a construct in Frame is a collection of buildings that have something in common, such as an address. There are different pictorial depictions for a building and a facility which makes it easy to identify if the selection is a building or a facility. There is also a quick search functionality that allows the user search for and select a particular building or facility. In this part also, Frame displays the username of the current user. A user can change the software's language by clicking on the username and selecting a different language. This part of the dashboard also provides a shortcut to the user's notifications page and Frame's logo which is programmed to always return the user to the dashboard when clicked on.

3.6.3. Real Estate Register

One of the central features of Frame is the module called Real Estate Register. Modules in Frame typically consist of submodules that perform targeted functions or provide specific functionality. In the Real Estate Register module, there are a list of submodules including, but not limited to, Real Estate Information, Plot Information, Building Information, Portfolios, and Tenant Search. By opening the Real Estate Information submodule, Frame displays a page that contains many tabs. These tabs are used to store different kinds of information depending on the needs of the customer. The tabs in Real Estate Information include Real Estate Information, Features, Contact Information, Notifications, and Subcontractors.

The Plot Information submodule is used to hold information about the plot on which the Real Estate is situated. Also, it is possible to add a new land lot to the database by using the "Add Land Lot" button on this page. This makes it possible for customers to include additional plots to their business and have such plots represented in Frame. The Building Information submodule is used to hold information about individual buildings belonging to the customer. In this module, there are tabs that divide the information is sections such as Estate's Information, Parking Spaces, Area of Responsibility, and Areas and Volumes. Here, there is the functionality to add a new building. Another submodule in the Real Estate Register is the

Rooms List. This is the submodule that displays the list of rooms in a selected building. This submodule also contains information about each of the rooms in sections/tabs such as Basic Information, Keys, Notification, and User/Inhabitants.

Additionally, the Real Estate Register has a Tenant Search submodule. This submodule provides the functionality to search for an individual tenant in a selected building. The search returns a table containing the name of the tenant, and some other useful data such as the Building, Room, Phone Number, and Property. In the Environment submodule, the environmental qualities of the building are stored. This information includes groundwater area, environment permit, and attachments. Portfolios is a submodule in Frame the performs a special function for each user of Frame. A user can group any number of properties or buildings together on any basis such as district, area for which the user is responsible, etc., and give this group a name for quick reference. This sort of group is called a portfolio.

3.6.4. Service Requests

Another module that is of critical importance to Frame software is the Service Request module. In Frame, a Service Request is a representation of an actual work order for operation on physical devices or maintenance of such apparatuses. This module contains submodules such as New Service Request, Service Requests, and Settings. The New Service Request submodule provides the functionality for creating a new Service Request concerning a building, room, an apparatus in a building, or room. A Service Request is always attached to a building. There are also some fields that must contain some value so that the Service Request can be sufficiently descriptive. In creating a Service Request, there also is the option of adding an attachment to the order. Once a particular Service Request is created, such fields as the building/property cannot be changed.

The Service Request submodule displays Service Requests from all buildings/properties. Existing Service Requests are displayed in a tabular form. This enhances the ease of interaction with these items. It is possible to limit this view through various measures, for example, user roles can determine which types of service requests users will see and where to find them. Besides, user roles can be defined to limit the number of buildings/properties from where a specific user can view Service Requests. A supervisor role will see a large number of service requests from different buildings/properties. Each Service Request exists with a status. A status is a stage in the processing of a Service Request. The statuses are "Not Started", "Received", "In Maintenance", "Wait", and "Ready". Each Service Requests progresses through each of these statuses except Wait. Wait is the status that shows that the Service is Request is on hold for one reason or another.

When the Service Request is initially created, it is in the "Not Started" status and when it is completed, it is moved to the "Ready" status. In some cases, a Service Request cannot be completed because another task – that was discovered after the Service Request was created – has to be done before it can be completed. In such cases Frame provides a functionality called Sub-Service Request. A Sub-Service Request is a Service Request that is spawned from a parent Service Request, it has all the functionalities of a normal Service Request, with the exception that it is attached to another Service Request. For any parent Service Request that has any number of Sub-Service Requests, the parent Service Request cannot be moved to "Ready" until all the Sub-Service Requests are completed and moved to "Ready". A parent Service Request is moved to "Wait" whilst the Sub-Service Request is being attended to.

In the table for Service Request, there is an option to open an individual Service Request. By opening a Service Request, the user is shown a page that contains tabs including Basic Information, Processing, Resources, and Log. The Basic Information page allows the user to view the details of the Service Request, and change some fields. As stated above, some fields such as Building/Property cannot be changed once the Service Request has been created. Additionally, this page allows the user to create a Sub-Service Request, and to view all the Sub-Service Requests attached to the specific Service Request. The Processing tab displays information about the status of the Service Request and allows the user to change this status. Changing Service Request status can also be done from the Service Requests table in the Service Requests submodules main page. The Logs tab displays a table of event that have been carried out on the Service Request and the username of the user who initiated the events.

On the Service Requests main page, a robust search functionality is implemented. This helps to narrow down the number of Service Requests visible on this page, as the number of Service Requests may grow large. Additionally, Portfolios can be used to sort Service Requests for better clarity. The Settings submodule contains basic settings for the Service Requests module. These settings are sorted into separate tabs such as Automatic Forwarding, Messages for

Service Requests, Service Requests List Managers, Service Request View Manager, Service-Level Agreement, and Service Request Statuses. These pages allow the user to specify specific behaviours for Service Requests.

3.6.5. Observations

The Observation module, as the name suggests, is used to record observations made during a routine check-up of a Building/Property. These Observations are divided into many categories including observations about Electricity, Ventilation, Heating, Alarms, and Water and Sewage. This module is used to record such observations for future use or for appropriate action. By opening these modules, the user is shown a table of observations already made about the Building/Property that the user has selected. Each observation in the table can be open and explored in finer detail. By opening an observation, the user navigates to a page that contains tabs such as Basic Information, Attachments, and Machines and Devices.

The Basic Information tab contains information about the observation made such as the Building/Property, the Type of observation i.e. Electricity, Heating, etc., and the Description of the observation. The identity of the user who made the observation is also recorded and displayed in the tab. In the Attachments tab, the user is allowed to submit attachments that are related to the observation. Instructions concerning the type and size of the attachments are also written in this tab. In the Machines and Devices tab, the user is allowed to add a machine or device to the observation. This is important especially if the observation is related to a specific machine of device. Additionally, all the machines/devices in the selected Building/Property for which observations have already been made are listed in this tab in a tabular format.

3.6.6. Maintenance

The Maintenance module is used to manage maintenance tasks for each Building/Property belonging to a customer in Frame. Maintenance tasks can be such tasks that are done on a regular basis according to a schedule, or the ones that are created spontaneously as a result of a fault. The module consists of four main submodules, namely: Defaults, Scheduling, Maintenance Tasks, and Yearly Calendar. These submodules are restricted according to the role assigned to the user who is logged in to Frame. Furthermore, there are two submodules called Maintenance Task Import, and Maintenance Task Template. These two submodules assist the customer in uploading predetermined maintenance tasks to Frame in a separate format. This is useful in that Frame automatically processes hundreds of maintenance task in one step instead of manually adding these tasks; which is time-consuming.

In the Defaults submodule, Frame displays a list of maintenance tasks in a table. Additionally, this page provides a robust filtering functionality that returns a list of maintenance tasks by title. It also provides the functionality to add a new maintenance task. Each maintenance task in the table can be edited by clicking the edit button which display a page with Basic Information, and Attachments tabs. The information on the individual maintenance task can then be modified, and attachments supplied, if necessary. Scheduling submodule displays a list of scheduled maintenance tasks i.e. maintenance tasks that have a particular date attached to them. In maintenance task scheduling, a company can add recurring maintenance tasks with dates. Also, these schedules can be changed or ended. In a colour-coded manner, Frame displays the different maintenance task schedule in the submodule, and provides search functionality.

Maintenance Task submodule provide a view of the different types of maintenance task that the customer has. The submodule is divided into four tabs, namely: Periodic, Daily, Monthly, Made When Necessary. The Periodic tab displays a list of maintenance tasks that are done periodically, with the period specified by the user. The Daily tab shows daily maintenance task, the Monthly tab displays monthly maintenance tasks, whilst the Made When Necessary tab display the maintenance task that do not fall into any of the other categories. All of these tabs provide advanced search functionality that can be parameterised. The Yearly Calendar submodule provides a calendar that displays all the maintenance tasks scheduled for the selected year. It is an overview of all the tasks that are to be done or that have already been done distributed into months. This submodule also provides an advanced search functionality.

3.6.7. Consumption

The Consumption module is designed to take data about a Property’s consumption in terms of water, heat, and electricity, and present that information to the user. This data is gathered from the meters present in the Building/Property and their readings over a period of time. The submodules in Consumption include Tracking, Readings, Meter Management, and Settings. The Tracking submodule provides a graph that displays a Property’s consumption on all three fronts side by side. It also provides a comparison functionality that allows the user to compare the consumption of many different Properties for different years on the same graph. There is a search functionality to assist in specifying what needs to be viewed by the user. The different metrics are colour-coded to provide clarity, and the contain detailed information during observation.

The Readings submodule is designed to provide, in a tabular manner, a list of meter-specific meter readings of all the meters in a Building/Property over a specified period of time. The table provides columns that detail the readings such that a lot of information is presented at a glance. Each meter reading in the table can be opened to inspect the meter in more detail. Clicking the open button displays more information about the meter over the space of the specified year. The page also draws a graph called Consumption Chart based on this information to show the user quick information about anomalies that may have occurred. This page also allows the user to input reading related to this device, and also to edit previous readings.

The Meter Management submodule is where the details of all the meter relating to a Building/Property are specified and stored. These meters are divided in three groups, namely: Electricity, Heat, and Water. Meter information can be edited and changed, new meters can be created, and old meters can be removed/deleted in this submodule. Additionally, all the changes made to a particular meter are stored in a log and displayed in the tab called Change History. Each category of meter can hold an unlimited number of meters and these meters are automatically attached to the Building/Property selected in the top margin. The Setting submodule is used to create and save settings for each of the three categories of meters. These settings include Unit Used, Form of Consumption, and Name.

3.6.8. Construction and Technology

The Construction and Technology module is implemented to contain the machines and devices that are attached to a specific Building/Property. Machines and devices include such items as Lifts, Pumps, Compressors, Sprinklers, and Transformers. This module contains submodules including General Description, Machines and Devices, Defaults, and Device Copying. The General Description submodule contains a list of devices, presented in a tabular form, and their descriptions. There is also a simple search functionality that allows the user to search for a specific device from the table. The user can also add a device from the list of devices linked to the Building/Property and add a description for it. Device descriptions can also be edited and/or deleted from this table.

Machines and Devices submodule contains a comprehensive list of all the machine and devices in the selected Building/Property. When a device is selected from the list of devices, the page displays tabs that allow for different operations on this device. These tabs include Basic Information, Attachments, Features, and Device Events. In the Basic Information tab, the basic information – such as the name, type of device, and building – about the device is displayed. Any attachments to this specific device can be viewed in the Attachments tab. There is the option to add a new attachment to this device as well. the Features tab displays a list of attributes of this device, such as unit of measurement, type of device, etc. A new feature can also be added here.

In the Device Events tab, there are three different tables that display information concerning this device. The Service Requests table shows all the Service Requests that have been created that are connected to this device. All the Service Request in this view operate like a normal service request from the Service Requests module. The Observations table shows all the observations that have been made about this device. The Maintenance Tasks table shows all the maintenance task that have been order for this device. In the Defaults submodule, the user can modify the properties of devices according to device groups. Device Copying submodule allows the user to copy devices from one Building/Property to up to 250 multiple Buildings/Properties at a time. This automatic copying makes it convenient for the user to replicated devices with ease, and removes the manual work necessary to include new devices to new buildings.

3.6.9. Documents

One of the other modules in Frame is the Documents module. This module is responsible for storing and displaying all documents that the user may wish to save to Frame. This module takes documents in many different formats, and it also allows the user to create folders so as to group the documents according to the discretion of the user. This module also provides a simple search functionality to make locating specific documents easy.

3.6.10. Contact Info

The Contact Info module is a repository module. It contains information about different entities connected to the specific customer. The entities may be companies, such as subsidiary companies, and subcontractors, or persons. To this effect, the Contact Info module is divided into Companies, and Persons submodules. The Companies submodule contains a list of companies linked to the customer, and the list is presented in a tabular form. There is a filter function that allows the user to filter the list of companies by predefined groups. Such groups are defined based on functionality, and type of business, for example. Additionally, this page provides the functionality to add a new company to a group and the list.

In the Person's submodule, a user can add persons to the list of persons who perform certain functions for the customer. This submodule also presents a list of persons that are already stored in the database to the user in a table. The information of each person on this list can be edited. The edit page provides four tabs: Basic Information, Properties, Premises, and Suppliers. The Basic Information page contains the person's basic information such as name, address, email, etc. The Properties tab contains the list of properties that the particular person is attached to. The Premises tab also contains the list of premises that the person is connected to. The suppliers tab contains the list of supplies connected to the specific person.

3.6.11. Reports

This is the last main Module in Frame. It contains a list of links to all the reports or reporting functionality that the customer subscribes to. The Reports module contains some form of reporting for most of the modules in Frame. Most of the modules have some form of report that can be created from them for the benefit of the customer. Reports include Real Estate Information from the Real Estate module, Service Request List from the Service Request module, Observation List from the Observation module, Maintenance Task Specification from the Maintenance module, Consumption Comparison Report from the Consumption module, Device List from the Construction and Technology module, and many more. This module is essential to the customer because it presents an overview of the present condition of the company to the customer in portable form.

The reports in this module are highly customisable and offer a wide range of options to the user concerning what can be presented on the report. The reports, after presentation can also be saved, exported, downloaded, or printed in many different formats including PDF, Word, CSV, and XML. This module is very helpful for end-of-year reports and permutations for Frame customers. Additionally, the reports are based on subscription, so each customer can select which reports are included in their instance of Frame. Additionally, access to reporting can be filtered based on the roles of the user, i.e. not all users have access to all the reports that the customer subscribes to. The final reports are designed to be easy to read and concise in information presentation.

3.6.12. Additional Tools

In Frame, there is a subsection called Additional Tools. This subsection contains modules that are useful but are not essential to the primary use of the software. These modules include Key Management, Quality Management, Product Register, and Tools. These modules perform different useful functions, Key Management is directed at customers who issue different keys to different users and associates of the customer. Key Management allows the customer to understand, without too much effort, the association between people and specific keys. It makes it easy to trace and track keys and how they are being used. Key Management contains two submodules namely: Key Management, and Defaults. Key Management submodule presents a list of keys that the user has access to. Each key has properties such as Name, Model, Status, etc., and the person to whom the key is presently distributed. Default submodule offers the user a way to group keys into different groups.

Quality Management module offers the user a means by which to carry out some inspections on Buildings/Facilities. In this module, there are submodules including Inspections, Templates, and Email Configurations. Inspections are routinely carried out on the customer's Infrastructure, to this end, a central place to store all the inspections and their results is essential for many customers of Frame. For many inspections there are templates that present a list of things to do to the technician. These templates are stored in the Templates submodules and the they can be edited to suit the requirements of each inspection. Before an inspection can be created, a template must be selected. If the inspection is such that doesn't have a template yet, one has to be created. Email Configuration submodule contains email configurations for users who receive immediate reports about the inspection by email.

For customers of Frame who deal with certain products, there is a Product Group module. This module allows these customers to store their products and retrieve them with ease. There are two submodules in the module namely: Product Group, and Product. Each product is listed under a product group. A product group is a collection of products that have something in common. This grouping makes it even easier to store, and retrieve information about product quickly. A product must belong to a product group by design. The Product submodule display s list of products in a tabular manner, with their specific attributes such as product group, description, unit price, etc.

In the Tools module, there are submodules such as Users, Subcontractors, Filters, Defaults, and Role Requirements. In the Users submodule Frame lists all the users of Frame that belong to the specific customer with some basic information such as name, login, email, and role. Each user's data report can also be ordered from this table. User Data Report presents the user's activity in Frame over a selected period of time. In the Subcontractor submodule, all subcontractors associated with the particular Frame customer are listed in a tabular form. Basic information about each subcontractor such as name, business Id, and address, is also displayed. All rows in this table can be edited. Filters submodule contains a list of all defined roles in Frame and their corresponding filters.

As stated earlier, different defined roles in Frame have specific filters that determine how much access to information each has. In this submodule, this access restriction is defined by filters. For each role there is a row in the table displayed and the columns are named according to the modules in Frame. A specific filter can be created for each column in the row thereby determining how many Buildings/Facilities a specific role has access to. In Role Requirement submodule, is similar to the Filters submodule to the extent that grants read/write access to different modules for different roles in Frame.

In this chapter, the modules and aspects of the Fatman Frame software that are relevant to this thesis were explored. This is by no means an exhaustive exploration of Fatman Frame. There are many more modules and functionalities that are left out of this chapter.

4. FATMAN API

FATMAN's API is designed to be an HTTP-based RESTful API technology. It uses the Entity Framework Core of Microsoft ASP.NET technology. Entity Framework (EF) Core is a light-weight, extensible, and cross-platform version of the popular Entity Framework data access technology. The decision to utilise this technology with the JSON-API-.NET was taken with consideration for our business goals. These goals are stated in chapter 5 and this technology is an approach that solves most of those goals.

Using Entity Framework Core allows the developers using the .NET technology to skip many aspects of the critical data-access code that would otherwise have been necessary. This is because the data-access code is implemented differently in EF Core through the Model. This model is comprised of derived context and entity classes that represents a session with the database. This implementation allows the user to query the database for data, and to save data into the database. Using EF Core, it is possible to generate a database from a Model; using EF Migrations, make models that match an already existing database, or generate models from an existing database. In the case of FATMAN API, the models were created to match an already existing database. (Esposito 2018: 3-13).

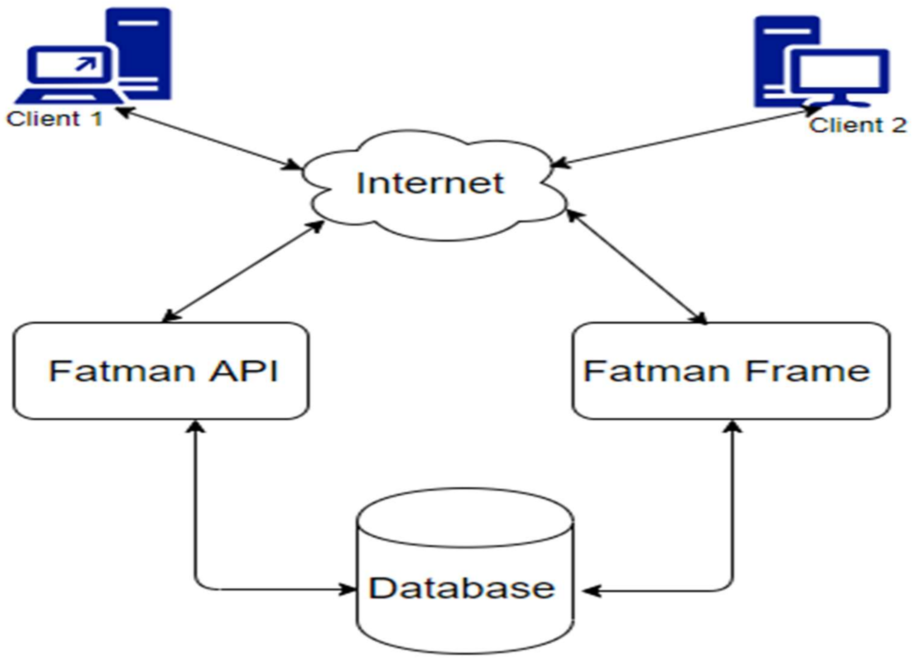


Figure 8. Fatman API in context

4.1. Authorisation

The OAuth 2.0 authorisation framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. An authorisation grant is a credential representing the resource owner's authorisation (to access its protected resources) used by the client to obtain an access token. The authorisation code is obtained by using an authorisation server as an intermediary between the client and resource owner. Instead of requesting authorisation directly from the resource owner, the client directs the resource owner to an authorisation server (via its user-agent), which in turn directs the resource owner back to the client with the authorisation code. (Hardt 2012: 8-9).

FATMAN API uses OAuth 2.0 Authorisation for all its transactions. The REST API uses the OAuth 2.0 protocol to authorise calls. OAuth is an open standard that is widely used by many companies for providing secure access to protected resources. To access Fatman's environments, a "Username" and "Password" is required. This is passed in the authorisation header of the HTTP POST call. Currently, to acquire the "Username" and "Password", a request has to be made to Fatman Ltd.'s helpdesk to have these created for, and supplied to, the specific customer. To obtain an access token for use in accessing the API, a request for a token must be made to Fatman's Identity Provider (IDP). The "Username" and "Password" must be supplied to the IDP in the request header. The IDP responds with an access token with which the user can access the API and begin to make HTTP requests.

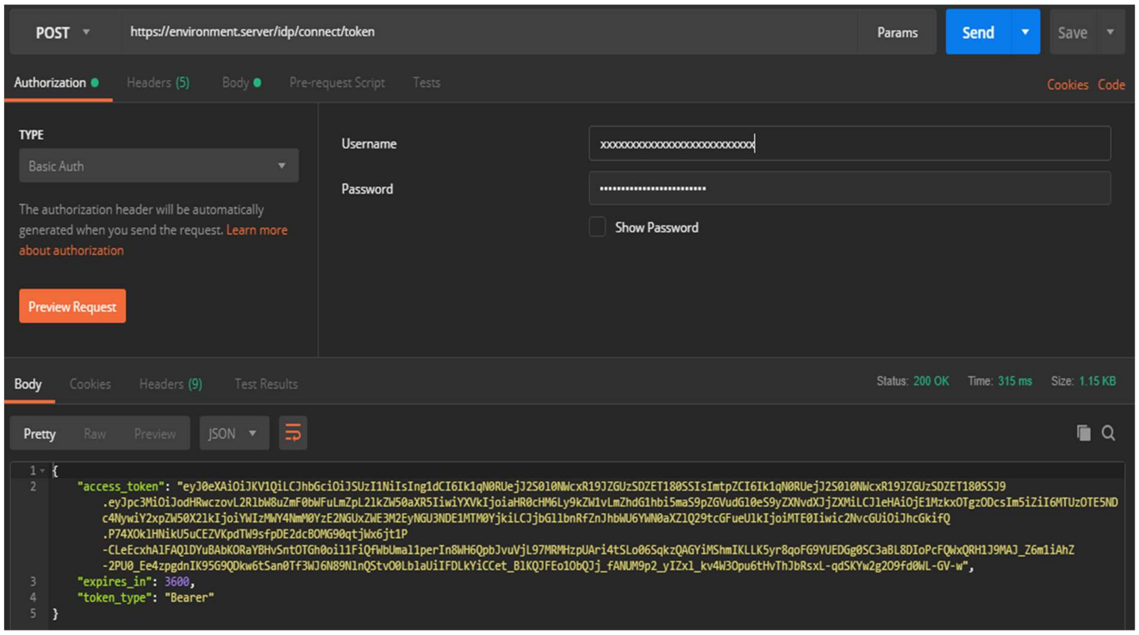
4.2. API Requests

As stated in Chapter 2, Fatman's API supports HTTP methods and calls. The POST call is used to retrieve an "access token" from the IDP. After the token has been received, the user/client is allowed to proceed with other types of HTTP requests. The four main HTTP requests supported by Fatman's API are GET, PATCH, POST, and DELETE. The API service is in different environments depending on the intended usage, the URL for each environment takes the form: https://environment.server/api/

The URL for a resource takes the form: https://environment.server/api/building/Id for the building with a specific Id.

4.2.1. Authorisation Request

As stated above, before the client or user can begin to use the API, the client must have a Username and a Password. To acquire these, the client must consult Fatman Ltd.'s Helpdesk so as to have this pair of elementary data created and supplied. Afterwards, the client can begin to make requests of the API. The first necessary request is the request for access token. This is a POST request that must contain the authentic Username, and Password. The server responds with a JWT access token. The client then has to include this access token in every future request sent to the API to gain access. This access token is only valid for a specified amount of time. The validity of the access token is given in the "expires_in" field of the response to the access token request.



POST https://environment.server/idp/connect/token

TYPE: Basic Auth

Username: xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Password: ••••••••••••••••••••••••••

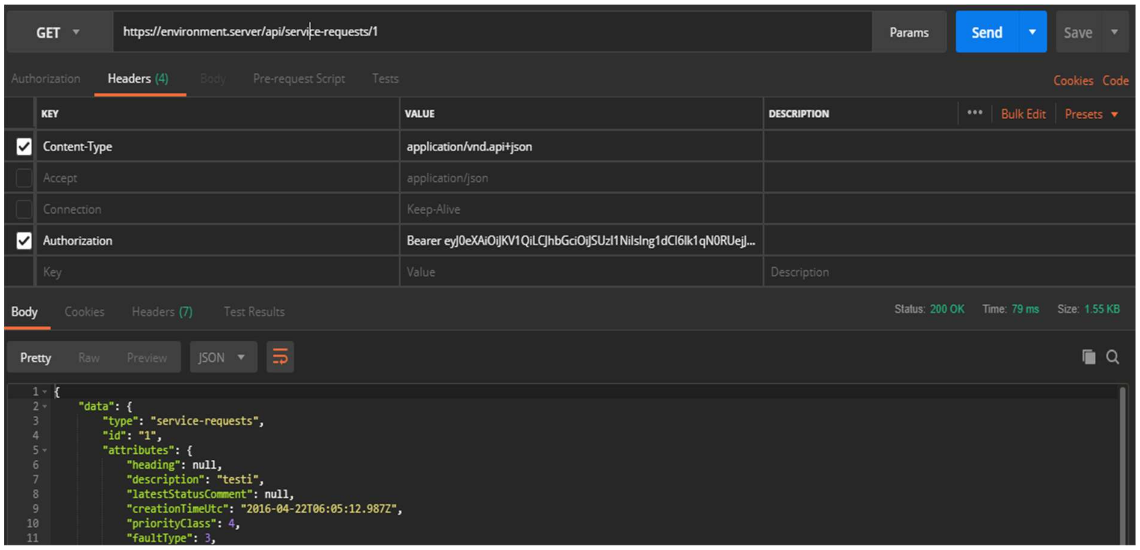
Status: 200 OK Time: 315 ms Size: 1.15 KB

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik1qN0RUejJ2S010NWcxR19JZGUzSDZET180SSIsImtpZCI6Ik1qN0RUejJ2S010NWcxR19JZGUzSDZET180SSJ9
        .eyJpc3MiOiJodHRwczovL2RlbW8uZmF0bWFuLmZpL2lkZW50aXR5IiwiYXVkIjoiaHR0cHM6Ly9kZW1vLmZhdG1hbi5maS9pZGVudGl0eS9yZXNvdXJjZXMiLCJleHAiOjE1MzkxOTgzODcsIm5iZiI6MTUzOTE5ND
        c4NywiY2xpZW50X2lkIjoiYWIzMWY4NmM0YzE2NGUxZWE3M2EyNGU3NDE1MTM0YjkiLCJjbGllbnRfZnJhbWU6YWN0aXZlQ29tcGFueUlkIjoiMTE0Iiwic2NvcGUiOiJhcGkifQ
        .P74XOklHNikU5uCEZVKpdTW9sfpDE2dcBOMG90qtjWx6jt1P
        -CLeEcxhAlFAQ1DYuBAbKORaYBHvSntOTGh0oil1FiQfWbUmal1perIn8WH6QpbJvuVjL97MRMHzpUAri4tSLo06SqkzQAGYiMShmIKLLK5yr8qoFG9YUEDGg0SC3aBL8DIoPcFQWxQRH1J9MAJ_Z6m1iAhZ
        -2PU0_Ee4zpgdnIK95G9QDkw6tSan0Tf3WJ6N89NlnQStvO0LblaUiIFDLkYiCCet_BlKQJFEo1ObQJj_fANUM9p2_yIZxl_kv4W3Opu6tHvThJbRsxL-qdSKYw2g2O9fd0WL-GV-w",
    "expires_in": 3600,
    "token_type": "Bearer"
}
```

Figure 8. Authorisation Request and 200 Status Code on response

4.2.2. GET

This is an HTTP request for data from the server. In the case of Fatman's API, the request must be made to the URL of the resource as indicated in API Requests section above. The request must contain the access token in the HTTP header. The response to the GET request from the server is formatted in JSON. If the GET request is correctly processed, the server returns the data in JSON to the user and the appropriate HTTP status code.



https://environment.server/api/service-requests/1

	KEY	VALUE	DESCRIPTION
✓	Content-Type	application/vnd.api+json	
	Accept	application/json	
	Connection	Keep-Alive	
✓	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik1qN0RUejJ...	
	Key	Value	Description

Status: 200 OK Time: 79 ms Size: 1.55 KB

```
{
    "data": {
        "type": "service-requests",
        "id": "1",
        "attributes": {
            "heading": null,
            "description": "testi",
            "latestStatusComment": null,
            "creationTimeUtc": "2016-04-22T06:05:12.987Z",
            "priorityClass": 4,
            "faultType": 3,
```

Figure 9. GET Request showing Authorisation Token, 200 Status Code, and Data in JSON format.

4.2.3. POST

This is an HTTP request for sending data to the server. The data being sent in this request must be formatted in JSON for the server to be able to process it. In any other format, or if there are syntax errors in the request, the server rejects the request and returns an appropriate error and status code to the client. If the POST request is correctly processed, the server returns the appropriate HTTP status code to the client.

4.2.4. PATCH

This is an HTTP request to modify a resource in the server. This request sends a partial data to the server so as to modify/update an already existing resource. This request has to be formatted in JSON for the server to be able to process is successfully. In any other format, or if there are syntax errors in the request, the server rejects the request and returns an appropriate error and status code to the client. If the PATCH request is correctly processed, the server returns the appropriate HTTP status code to the client.

4.2.5. DELETE

This is an HTTP request to remove a resource from the server, or to make it no longer accessible over the internet. Each of these two approaches to deleting a resource is implemented in Frame depending on the resource. This request has to be formatted in JSON for the server to be able to process is successfully. In any other format, or if there are syntax errors in the request, the server rejects the request and returns an appropriate error and HTTP status code to the client. If the DELETE request is correctly processed, the server returns the appropriate HTTP status code to the client.

5. DESCRIPTION AND EVALUATION OF FATMAN API'S BUSINESS GOALS

Innovation in the field of technology has allowed businesses and companies to create new ways of representing and presenting customer data and information to the customer over the internet. Different types of technological innovations can be used to create customised functionalities for individual customers, or generic functionalities for all customers of a company. From the technical perspective, an API is a tool that allows a developer to interact with a company's data and/or call a functionality over the internet. From this perspective, there is an increase in the number of ways that a company can trade, offer, and exchange services and functionalities with itself across regions, with its partners, and with the general public. This allows for creativity in the ways in which the services and functionalities are utilised by the company's customers and partners.

For Fatman Ltd., receiving, manipulating, and storing customer data is an essential aspect of the services provided to customers. The company is entrusted with crucial customer information that the customer needs to update regularly. Information such as service requests, invoice, devices, and personnel data are generated frequently and accessed widely. Usually data in many formats are also used to create reports that the customer uses. These reports include financial reports, consumption reports, and audit reports. Fatman Frame, and all other products of Fatman Ltd. manipulate customer data and information to produce the results that the customer has requested. It is of importance to the business of Fatman Ltd. that the customer has access to data and information as and when needed. It is equally crucial to the customers of Fatman Ltd. to have unlimited and uninterrupted access to data stored with Fatman Ltd. in any of the many products of the company.

To provide access to data and information to customers, Fatman presently utilises separate, customised, and dedicated integration platforms for many customers. Integration platforms such as Mule Integration have played a crucial role in the business of Fatman Ltd. so far. Each integration is specially designed to suit the needs of a specific customer. Over time, this approach has spawned a long list of special integration platforms that require special knowledge and expertise to update, and maintain. The knowledge and resources required to monitor, maintain, and repair these integrations is a matter of disadvantage to the business of Fatman Ltd. These resources can be re-deployed to improve on the software packages instead of to maintain various integrations. It is a common occurrence that a certain integration for a customer experiences down time and resources, both human and material have to be redirected to solve this problem as quickly as possible.

Fatman API provides a technical solution to this problem. Fatman API is not a dedicated platform therefore it can be used by as many customers as choose to. In the future, all customers will have access to their data and information only via the API. This makes maintenance less resource-intensive. The technical knowhow for maintaining one solution can be more easily accessed and shared. Centralising access to data reduces the risk of data leaks, and also introduces a dedicated method for data security. In the technical solution where there are many dedicated integrations between different customers and the company's software packages, there is a greater risk of data leaks.

In addition, the provision of one solution to all customers rather than tailored solutions to each customer reduces the resources required to create the solution in the first place. This is to say that creating an API is a one-time event, the resources needed to create the system are utilised only once – once the system is set-up, the only resources required will be for maintenance and improvement. This is not the case in the scenario where separate, tailored, and dedicated integration platforms are in use. There are always new customers and each customer will require a new tailored integration platform. Overall, the centralisation of data storage and retrieval, and information access, reduces the quantity of resources needed to create, maintain, customise integration platforms for both Fatman Ltd. and the customers. This improvement and reduction in resource expenditure is of critical business importance to the customers and the company itself.

The Fatman API is created using Microsoft EF Core. This is a modern technology that gets updated regularly. This technology provides a variety of advantages over many other technologies, including Global Query Filters, Filed Mapping, DBContext Pooling, and Property Value Conversions. These advantages are essential to modern technological solutions for data access. The performance of this new technology has also been proven to be excellent. This modern technology is easily recognisable amongst professionals in terms of adaptation. The skills necessary to utilise this technology are widespread amongst professionals which makes it easier to hire developers who already know EF Core and can work with it. (Esposito 2018).

When the sales team of Fatman Ltd. have gone to sell Fatman Frame and have introduced the API to the prospective customer, the technical departments easily recognised the technology and it made the sale of the software package smoother. If the technology used to create the API were not so easily recognisable, sales would be more difficult as the customer would be required to hire specialised technicians to create an integration with the API. This is also of business importance to the company as the technology utilised to create the API makes it easier to sell the software package and close deals more quickly than when customised and dedicated integrations were needed to be discussed in tiny details.

When Fatman Ltd. had not invested resources in creating the API, the sales team reported that customers usually made enquiries about the technology in use for the company's integrations. Customers also routinely inquired about the company's API implementation. This feedback contributed to the need for the company to create an API implementation to meet customer demand. An API implementation is the modern approach to data access and manipulation for many technological companies. Companies such as Google, Amazon, and Facebook also use this approach. For any company, having an API implementation is a mark of modernity, and technological advancement. A company that is utilising modern and technologically advanced systems is good for sales and business success. (Forbes Technology Council 2018).

From the perspective of information and data security, a centralised system is more easily secured than a decentralised system. Resources that are geared towards the security of data and information can be aggregated and optimised if the repository has only one access route. In a decentralised system, security, especially for smaller companies, is spread thin over many different implementations, and that can have a detrimental effect on the overall security of data and information. If the security of a company is compromised, the business and reputation of such a company is affected severely. As security of information and data is of utmost importance to the business of Fatman Ltd., the API approach is an advancement in enhancement of data security and customer confidence. Resources required to secure this system can also be streamlined and optimised by this approach. (Lai & Chan 2008).

In summary, the introduction of the REST API implementation for data/information access to the customers of Fatman Ltd. solves the business goals that necessitated this implementation. The customers who have made a demand for this technology are satisfied with the implementation. Furthermore, this means that the sales of the software products of the company are made more efficient because the sales team includes the presence of the API in their sales pitches. For both the customers and the company, the implementation of the API is a solution to many business problems and an achievement of many business goals.

6. DESCRIPTION AND EVALUATION OF FATMAN API'S DESIGN GOALS

When embarking on a software project, the developers involved, besides the business requirements of the implementation, agree on the goals that the system to be created must fulfil in terms of technical design. These goals are either written down concretely or embodied by the team. These goals are important to the creation of the system because it informs the approach, both technically and functionally, to the development of the system. In modern technology, there are typically more than one way to create any software system; as such, it is critical that the goals of such a system are well-known and agreed upon by the team of developers and the business interests. This is important to the decision that concerns selecting the right technology, team, programs, and implementation. They also assist in breaking any deadlock in decision making if any occurs in the process of the project. The decisions that lead to the achievement of the design goals are determined and implemented by the team.

For the Fatman Frame API team, the preliminary design goals for the API implementation were determined and relied upon during the process of developing the system. These goals include the ones listed below, against which the system was tested after its initial implementation.

1. Testability: The design focuses on making sure that the API is unit-testable, integration-testable, and performance-testable.
2. Measurable Performance: The design must achieve industry-standard performance levels.
3. Compatibility with legacy database schema.
4. Allowance for incremental exposure of resources and fields.
5. Flexibility: Allowance for the implementation of complexity (when necessary).

These are some of the goals that assisted the team in the implementation of the API project. These goals were decided upon with an understanding of the business model of Fatman Ltd. and the present technological capabilities of the company's systems. In the business model and goals as described in chapter 5, one of the major aims of the API implementation is to serve as a generic mode of integration to Fatman Frame. This will remove the need for customised integrations for individual customers, as is the current situation. To this end, the longevity forecast for the technology is important. Moreover, the technology must also provide a solution that is general and can be easily deployed on the client side.

6.1. Testability

The quality and correctness of a software product can be improved practically and effectively by program testing. The benefits of program testing include cost effectiveness, immediate feedback, and simplicity. For a complete modern software product, the source code must contain test where and as necessary. This provides benefits and an insurance against the malfunctioning of the software in the future. It must be noted that tests must be maintained in synchrony with the source code so as to keep the tests up to date, and effective. If the code under review by a test code changes, the test code must also be changed to reflect such modifications to maintain high quality performance. This is an investment of resources that is vital to the proper functioning of the software in the present and in the future. (Cheon & Leavens 2002).

A unit test is a software testing level where single components of the source code of a software are tested to ensure that they are performing the function for which they were designed and written. (Beck & Gamma 2000). For Fatman API, an essential function is to receive data from the customer and store it in the database. Additionally, exposing data from the database to the customer is an essential function of the API. Unit tests can be written for these functions, but they are not necessary, and as such, the API team did not expend resources on implementing these unit tests. However, in the Service Requests resource, since there is some amount of business logic implemented in the API, unit tests are crucial. These tests have not been implemented in the API yet.

To ensure that the source code for the API allows for unit testability, non-deterministic behaviour is avoided. Non-deterministic behaviours are such that arise from utilising aspects of code that require a hidden input. Such aspects include the use of Datetime.Now to determine the time. Furthermore, tight coupling to a concrete data source is avoided in the design and implementation of the API. Tight coupling is the reliance of an aspect of code on specific data from a specific data source for proper functioning. For unit testability in design, classes, functions, and modules of a given software should follow the Single Responsibility Principle (SRP). The SRP is the style of programming where a unit of the program is designed to perform only a single function. This principle was followed in the design of the API, where possible, units of the code are written to perform only a single function such as get data, read data, and process data.

In the Service Request module of Fatman Frame, there are functionalities that make the module function as designed. These functionalities include sending emails to the right recipients when a Service Request is created, updated, and/or completed. For these functionalities, triggers have been implemented in the software. These functionalities must be unit-tested to ensure that the software sends emails according to the data received and that all emails are sent at the right time. Changing the status of a Service Request is another example of when testing the email trigger is necessary. These are of business-critical importance to both Fatman Ltd., and the customers as they affect the Service Level Agreement (SLA) rules.

Integration testing is the type of testing done in a software where composite units are integrated and tested as a group. This test is suited to the discovery of faults and inadequacies in the interfaces between two or more sections of a software's code. In Fatman Frame, many of the individual modules are connected to each other in terms of functionality. These include Service Requests connected to Reports, and Construction and Technology connected to Maintenance. To ascertain that these modules work together seamlessly, integration testing is important. This necessitates the testing of the entire flow of the process. One of such flows is receiving data from the user and storing it in the database, in the right format. Fatman API is designed to be integration testable, however, integration tests have not been implemented.

Integration testing is the stage of testing that directly follows and relies on unit testing. If unit testing is properly designed for, integration testing is accomplished more easily. Additionally, to ensure that the codebase is integration testable, dependency injection is implemented across the source code liberally. To test a flow that gets data and stores it in the database, for instance, the test code must be designed to avoid invoking the aspect of the code that actually hits the database. Hitting the database in a testing scenario has detrimental effects on the entire software, as it may modify the data stored there. The use of SOLID design principles also assists in making integration testing less troublesome.

k6 is a developer-centric load testing tool designed to help the developer incorporate load testing into the workflow, and ultimately get it into the automation flow. It aims to do so by:

1. Being easy to get started with; open-source, well documented and with simple command line usage.
2. Scaling with the developer from simple unit load tests (e.g. single API endpoint) to more complex scenario load tests (e.g. an application using a full API).
3. Allowing the developer to seamlessly go from local or on-premise use to the convenience of managed cloud use.

Performance testing is crucial to the process of creating a software. Any software must be designed with performance testability as a goal of the design team. In creating Fatman API, performance testability was a design goal. This goal has been achieved and performance tests have been carried out on the software using the k6 software. All according to the k6 documentation.

6.2. Flexibility

The major requirement of Fatman API is to collect data and insert the data into the database of the customer. The other key feature of the API is to present data to the customer in the appropriate format upon request. As a consideration, business logic and customer customisation are not included in the design of the API, it is designed to be as generic and universal as possible. This allows for it to be used by as many customers as possible without the added application of customisation. Fatman Ltd. has other integration systems that have existed for a long time, these systems include web services integrations with customers' systems. These integration scenarios also are of many formats, including Simple Mail Transfer Protocol (SMTP) and File Transfer Protocol (FTP). After the implementation of the API, these integrations will still exist.

The implementation of the API will not solve all the integration difficulties that need to be solved. This is not the goal of the API. As such, it was not designed to implement complex flexibility. It is designed to solve one of the most important needs of Fatman Ltd., and the customers that use Fatman Frame and the other software products of the company. The API is designed to be cost, technical, and human resource efficient. It is not designed to be overridden or to solve all the integration scenarios of Fatman Ltd.'s software products.

6.3. Allowance for incremental exposure of resources and fields

One of the goals of the API is to allow the developers to be able to include new fields and expose new features throughout the lifespan of the API. This was so as to monitor the usage of the product, and to increase the technical depth that it represents. Presently, the implementation supports the incremental exposure of resources. The resources that were included in the initial exposure included the key modules in Frame such as Service Requests, Buildings, Lots, and Rooms. Resource relations can also be incrementally created and exposed. Resource relations include HasMany; as in Facility/Building to Room and HasOne; as in Resources to Service Request. These resource relations were incrementally exposed via the API and didn't have to be built in from the start. This implementation has been mostly seamless except in the cases where the current database schema was unsuitable.

6.4. Compatibility with legacy database schema

Fatman Ltd. was founded about 26 years ago. For this reason, and as it is true for any company that was founded in that era, there is not only legacy software produced with old technology, there is also legacy database schema. Dealing with legacy software and technology is not a novel engagement, therefore, an understanding of this is critical for the developer especially when considering what technologies to utilise in a new implementation that relies on legacy database schemas, such as the Fatman API. The goal decided on by the Fatman API team from the onset was to build the API in such a way as to suit the legacy database schema without being restricted to it. This was a crucial goal to the business objectives of the company. It allows the company to reuse legacy technology and thereby save resources.

The implementation was designed and implemented to suit the legacy database schema unless it was detrimental to the other goals or the business needs of either the company or the customers. In some instances, the database schema did not support a particular technology. One of such instances was the use of composite key technology in the Entity Framework Core using Fluent API. For the resource Resource in Frame, a single resource can only be identified uniquely using two attributes; relationType and relationId. This necessitated the usage of a composite key relation to the Service Request attribute serviceRequestId. This method could not be implemented because the current database schema does not support such implementation. In summary, the design goal of fitting the API software to the legacy database schema was fulfilled, except in single instances where the database schema did not support the technology implicitly.

6.5. Measurable Performance

The Fatman API was implemented with consideration for using the k6 software (described above) tests to quantify and verify the performance of the API. Using the k6 test, some performance-related indices such as throughput, requests per second, CPU utilisation, concurrent users, error rate and peak response time were determined. The API team did not explicitly decide what the proposed-optimum performance was to be. In summary, the performance of the API implementation was adjudged satisfactory by the team, and plans are in place to improve on the performance wherever necessary in the future.

6.6. Pricing for the API

The monetisation of the REST API implementation of is importance to Fatman Ltd. The company is interested in what monetisation models are available and which ones suit the company's needs and profile. In the API industry, there are three established methods for API pricing, namely: product adoption, data collection, and developer usage.

6.6.1. Data Collection

This is the method of collecting data from the user, with data privacy in mind, to determine how the user sues the API and thereby charge the user on usage. Getting information about usage is crucial to the future development of the API itself. It allows the API team to understand the needs of the user and create better and faster means to fulfil these needs.

6.6.2. Product Adoption

This is a method in which developers invest in creating functionalities that make it difficult for the users of the API or software product to migrate to other providers of similar products. Automating commands and processes in the same software by the developers of the software makes it more difficult for users to migrate to other solutions.

6.6.3. Developer Usage

This is the type of monetisation where the API product is charged directly based on usage. API calls, requests, and data usage are some of the deductibles in this method. This method is the most typically used method in the industry. In the method, there are three distinct models of monetisation for the API, namely: overage model, fixed quotas, and the so-called pay-as-you-go model.

6.6.3.1. Overage Model

For the overage model, the company charges a certain amount for a specific quantity of data usage and a premium on overuse i.e. 350 downloads per month + £0.02 per overuse download. In this model, the company offers a certain number of requests to the customer at a certain price. Whenever this offered number is exceeded, the customer has be pay a premium on top of every extra request that is made to the API.

Advantages:

1. Predictable Revenue
2. Predictable Pricing
3. API is always up and running, no downtimes.

Disadvantage:

1. No objective way to determine overage pricing

6.6.3.2. Fixed Quota Model

For the fixed quota model, the total data usage, calls, requests to the API are capped at a certain threshold i.e. the customer is only allowed 700 calls per month. The customer can only make a limited number of calls to the API in a specified amount of time. This means that at a certain point, the customer can no longer use the API because the number of requests allowed has been reached.

Advantages:

1. Predictability of pricing
2. Predictability of revenue
3. Revenue is certain even from customers with low usage

Disadvantage:

1. Application is no longer available to the customer when the quota is fulfilled

6.6.3.3. Pay-As-You-Go Model

For the pay-as-you-go model, the customer is charged per call, request, download etc., without having a cap on the amount that can be used i.e. €0.04 per call. The customer is only charged for the value the is used. The more requests the customer makes to the API, the more the customer has to pay. The price increases linearly with the number of requests.

Advantages

1. Linear scaling depending on activity of the user
2. Price scaling

Disadvantages

1. Unpredictable revenue
2. Budgeting issues as revenue is not constant

7. CONCLUSION AND RECOMMENDATION

This is a three-part document that deals with the evaluation of the role that the Fatman API implementation has played in solving the business goals of Fatman Ltd. and the customers of the company in a business-to-business scenario, and the design goals of the API itself. The first part is an introduction to the fundamentals of web apps, and some technical background of the thesis. The second part provides an introduction to Fatman Ltd. and Fatman Frame as a software product and introduces some of the essential parts of Fatman Frame that are relevant to the topic of the thesis. The third part made an evaluation of the JSON-API-Format REST API in this business-to-business scenario. The third part also explored whether the REST API met its own design goals.

This thesis is a necessary aspect of the creation of this new solution to the business concern of the company and the customers of the company. It is also a step in the direction of modernisation that Fatman Ltd. aims for. During this thesis, a documentation document for the REST API that will serve as the first draft of what will be the final documentation was also created. The API is presently in the demo state and it will be deployed in the production environment in the near future.

For future development of the REST API, it is recommended that scalability of the system is considered and implemented. Presently, only one instance of the REST API is deployed on one server, and it is not dynamically scalable. It will be beneficial to be able to scale to more instances on different servers. The legacy database schema presently limits the REST API in some of the ways discussed in this thesis. Additionally, a functional sandbox should be provided for future client developers to test the functionality of the API beforehand. In the future, the database schema should be redesigned to meet modern standards and therefore create more technically robust implementations for the REST API. Azure API Management suite is recommended for the pricing of the REST API. It is optimised to utilise the Developer Usage model for API pricing, this is the most typically used model and it is highly recommended.

REFERENCES

Beck, Kent & Gamma, Erich (2000). Test Infected: Programmers Love Writing Test [online] [cited 9 August 2018], 1-15. Available from Internet: <URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.5961&rep=rep1&type=pdf >.

Berson, Alex (1992). Client-Server Architecture. New York: McGraw-Hill. ISBN 978-007-005076-1

Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format. *Internet Engineering Task Force (IETF) Request for Comments* [online] 8259 [cited 1 June 2018], 2-16. Available from the Internet: <URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc8259.txt.pdf>. ISSN 2070-1721.

Bray, Tim., Paoli, Jean., Sperberg-McQueen, C. M., Maler, Eve & Yergeau, Francois (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition) (2008). W3C Recommendation [online] dated after 7 February 2013 [cited 12.7.2018]. Available from World Wide Web: <URL: https://www.w3.org/TR/xml/>.

Chen, Mingste, Annadata, Anil K., Chan, Leon (2001). Adaptive Communication Application Programming Interface [online]. Oracle America Inc. [cited 20 Aug. 2018] Available from the World Wide Web <URL: https://patents.google.com/patent/US7581230B2/en>.

Cheon, Y. & Leavens, Gary T. (2002). A Simple and Practical Approach to Unit Testing: The JML and JUnit Way. Magnusson B. (eds) ECOOP 2002 — Object-Oriented Programming. ECOOP 2002. Lecture Notes in Computer Science, vol 2374. Springer, Berlin, Heidelberg. ISBN 978-3-54043759-8.

Deitel, Paul & Deitel, Harvey (2010). JAVA How To Program. 8th Ed. New Jersey: Pearson Education Inc. ISBN 978-0-13-136483-7.

Dusseault, L. & Snell, J. (2010). Patch Method for HTTP. *Internet Engineering Task Force (IETF) Request for Comments* [online] 5789. [cited 5 September 2018], 2-4. Available from the Internet: <URL: https://tools.ietf.org/html/rfc5789>. ISSN 2070-1721

Esposito, Dino (2018). Programming ASP.NET Core. 1st Ed. London: Pearson Education Inc. ISBN 978-1-50930441-7.

Fatman Ltd. (2018). About Us. Espoo: Fatman Ltd. [online] [cited 30 August 2018]. Available from the World Wide Web: <URL: https://fatman.fi/en/company/>.

Fielding, R. & Reschke, J. (2014). Hypertext Transfer Portocol (HTTP/1.1): Semantics and Content. *Internet Engineering Task Force (IETF) Request for Comments* [online] 7231:2817 [cited 3 September 2018], 24-64. Available from Internet: <URL: https://tools.ietf.org/html/rfc7231#section-4.3.1>. ISSN 2070-1721.

Forbes Technology Council 2018. Options for Boosting Your Tech. Business' Success. New Jersey: Forbes Media.

Freeman, Adam (2013). Pro ASP.NET MVC 5. 5th Ed. California: Apress Media. ISBN 978-1-43026530-6.

Hardt, D. (2012). The OAuth 2.0 Authorization Framework. *Internet Engineering Task Force (IETF) Request for Comments* [online] 6749 [cited 30 August 2018]. Available from the Internet: <URL: https://tools.ietf.org/html/rfc6749#section-1.3>. ISSN 2070-1721.

Jones, M., Bradley, J. & Sakimura, N. (2015). JSON Web Signature. *Internet Engineering Task Force (IETF) Request for Comments* [online] 7515 [cited 2 August 2018]. Available from Internet: <URL: https://tools.ietf.org/html/rfc7515.html>.

Lai, Ray Y. & Chan, Ka Fu (2008). Method and Apparatus for Secureky Invoking a REST API [online]. Intuit Inc. [cited 31 Aug. 2018] Available from the World Wide Web <URL: https://patents.google.com/patent/US8621598B2/en>.

Massé, Mark (2012). REST API Desing Rulebook. 2nd Ed. California: O'Reilly Media Inc. ISBN 978-144-931050-9

Nurseitov, Nurzhan, Paulson, Michael, Reynolds, Randall & Izureita, Clemente. Comparison of JSON and XML Data Interchange Formats: A Case Study [online]. Montana: Montana State University [cited 20 June 2018]. Available from the World Wide Web: <URL: https://pdfs.semanticscholar.org/8432/1e662b24363e032d680901627aa1bfd6088f.pdf>.

Sandoval, Kristopher (2015). Functional vs. Useful: What Makes A Useful API? Nordic APIs. [cited 25.7.2018]. Available from the World Wide Web: <URL: https://nordicapis.com/functional-vs-useful-what-makes-a-useful-api/>.

Stallings, William (1990). Business Data Communication. New Jersey: Prentice Hall. ISBN 978-002-415431-6.

Totty, Brian., Gourley, David., Sayer Marjorie., Aggarwal, Anshu & Reddy Sailu (2009). HTTP: The Definitive Guide. 9th Ed. California: O'Reilly Media Inc. ISBN 978-156-592509-0.

APPENDIX 1. FATMAN API Documentation

Get Started

The Fatman APIs are HTTP-based RESTful APIs that use OAuth 2.0 for authorization. API request and response bodies are formatted in JSON.

Authorisation

The Fatman REST API uses the OAuth 2.0 protocol to authorise calls. OAuth is an open standard that is widely used by many companies for providing secure access to protected resources. To access Fatman's environments, a "Username" and "Password" is required. This is passed in the authorisation header.

Currently, to acquire the "Username" and "Password", a request has to be made to Fatman Ltd.'s helpdesk to have these created for, and supplied to, the specific user. To obtain an access token for use in accessing the API, a request for a token must be made ot Fatman's Identity Provider (IDP). The "Username" and "Password" must be supplied to the IDP in the request header. The IDP responds with an access token with which the user can access the API.

Below is an example of the flow (Javascript):

To obtain an Access Token

```
var xmlhttprequest = new XMLHttpRequest();
xmlhttprequest.open("POST", "https://environment.server/idp/connect/token", false);
xmlhttprequest.setRequestHeader("Authorization", "Basic
YWIzMWY4NmM0YzE2NGUxZWE3M2EyNGU3NDE1MTM0Yjk6UnlTUFJKamgzaEZnS3dsYVY4Vjh1dTQ
4VQ==");
xmlhttprequest.setRequestHeader("Content-Type", "application/x-www-form-urlencoded")
xmlhttprequest.send("grant_type=client_credentials&scope=api")
xmlhttprequest.responseText
```

In the request header, the "Username" and "Password" are encoded with a base64 hash.

<u>The response from the IDP takes the form:</u>

```
{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ik1qN0RUejJ2S0l0NWcxR19JZGUzS
DZET180SSIsImtpZCI6Ik1qN0RUejJ2S0l0NWcxR19JZGUzSDZET180SSJ9.eyJpc3MiOiJodHRwczovL2Rlb
W8uZmF0bWFuLmZpL2lkZW50aXR5IiwiYXVkIjoiaHR0cHM6Ly9kZW1vLmZhdG1hbi5maS9pZGVudGl0
eS9yZXNvdXJjZXMiLCJleHAiOjE1MjQ0NjgzMzQsIm5iZiI6MTUyNDQ2NDczNCwiY2xpZW50X2lkIjoiYWIz
MWY4NmM0YzE2NGUxZWE3M2EyNGU3NDE1MTM0YjkiLCJjbGllbnRfZnJhbWU6YWN0aXZlQ29tcGFu
eUlkIjoiMTE0Iiwic2NvcGUiOiJhcGkifQ.Z-
l2BgTUbcLEcsTEwVOIGIDUuFzb62A6v7oxcxk55VxZ1TzCWD6itIr5AvMm81z4c5ZKLjdx7ty9K6pIc2BMX
APJlVcwTd-rIZH_hlGpIzGrHEP1kr9ihtb_WSvod-
r74XwK0Z6iUuZAA385zRgNFPH7bi4wAWOLUjuE4EfIq7xJUAEwrIguwZ8ATuufOO71XFDVt8k3dxpqRia
XVze6LERbgX-t-
vWfXlVEYUqePX1i2iCndVxu4W7lJc5h6TguVjwbMMslo27SUSOViKp7EZ1RZHju7eY9BccIj4TrWhtzE-
ZaaJHaPMhZGh_nOM1fD5XhxoUp0v5Wc3a_TqOPMA",

"expires_in":3600,

"token_type":"Bearer"}
```

The expiration time is indicated in the “expires_in” field.

The “access token” is created with the JWT open standard.

Currently Swagger doesn’t support JSON API dotnet Core.

<u>General Requests for the API</u>

All requests require Authorization header that contains the API access token. Access token can be retrieved from identity endpoint using oauth2 client credentials flow. All requests contain the JSON request body. The following elements are used to make requests of the REST API.

- GET: This request is used to obtain data from a resource.

```
var accessToken =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6InRrNklWNHhQSVVGeVd3dUxCRmI2WWJ0
eVBwcyIsImtpZCI6InRrNklWNHhQSVVGeVd3dUxCRmI2WWJ0eVBwcyJ9.eyJpc3MiOiJodHRwc
zovL2Rldi1sYXRlc3QuZmF0bWFuLmZpL2lkZW50aXR5IiwiYXVkIjoiaHR0cHM6Ly9kZXYtbGF0ZX
N0LmZhdG1hbi5maS9pZGVudGl0eS9yZXNvdXJjZXMiLCJleHAiOjE1MTE5NTQwOTEsIm5iZiI6M
TUxMTk1MDQ5MSwiY2xpZW50X2lkIjoiYWIzMWY4NmM0YzE2NGUxZWE3M2EyNGU3NDE1
MTM0YjkiLCJjbGllbnRfZnJhbWU6YWN0aXZlQ29tcGFueUlkIjoiMTE0Iiwic2NvcGUiOiJhcGkifQ.f
ofL7xm_Xk3hF1uLmRZnuFfGkx8sg2FcKd0FmfTFB5rVKbmxx4IiU0ptx4Ug7GXi4z7ttBDpcx83M
5msinHRr3g-BLSsOpZZTILH8Ynd7dSUtIiohoDXhy4DwNLaZ2-4PF3BJEhb-
7SQ4brH2Zqur1ZdxCYnhnlg0yAP1qiEVP9Ws93RzrnyWJz7LVU-
iJVEXRhBqcVKC2bKhvUnVBJrm0PR4wZ1d9KSLMnYCYvVKlfl6uLyiqSLhk1nz5jNaMVVfJTJlzLD8
MEC8C3yVHdhNEvF76qwoAQbp904TNS5ck_Gke7rCFWhQ-
L9jT6VvnVAbjhxk1D33Z3mSpkcyrnK2A";

xmlhttprequest.open("GET", "https://environment.server/api/buildings")

xmlhttprequest.setRequestHeader("Authorization", "Bearer {your token}")

xmlhttprequest.send()
```

- PATCH: This request partially updates a resource.

```
var data = {
        "data": {
        "type": "buildings",
         "id": "13",
        "attributes": {
         "Antennijarjestelma": "To TDD or not"
                }
        }
}
xmlhttprequest.open("PATCH", "https://environment.server/api/buildings/13")
xmlhttprequest.setRequestHeader("Authorization", "Bearer {your token}")
xmlhttprequest.setRequestHeader("Accept", "application/vnd.api+json")
xmlhttprequest.setRequestHeader("Content-Type", "application/vnd.api+json")
xmlhttprequest.send(JSON.stringify(data))
```

- DELETE: This request is used to delete a resource.

```
var buildingIdToDelete=2137;
xmlhttprequest.open("DELETE","https://environment.server/api/buildings/2137");
xmlhttprequest.setRequestHeader("Authorization", "Bearer {your token}");
xmlhttprequest.setRequestHeader("Accept", "application/vnd.api+json");
xmlhttprequest.setRequestHeader("Content-Type", "application/vnd.api+json");
xmlhttprequest.send();
```

- POST: Provides data to a resource for further processing.

```
xmlhttprequest.open("POST", "https://environment.server/api/buildings");
xmlhttprequest.setRequestHeader("Authorization", "Bearer {your token}");
xmlhttprequest.setRequestHeader("Accept", "application/vnd.api+json");
xmlhttprequest.setRequestHeader("Content-Type", "application/vnd.api+json");
xmlhttprequest.send(JSON.stringify(newBuilding));
```

The API service is in different environments depending on the intended usage. The URI to the resource takes the form:

- https://environment.server/api/buildings/

List of Resources and Attributes

Below is a list of Resources exposed via the API in no particular order:

- **Buildings**

  Attributes:

  - grossArea (double? GrossArea): Gross area in metre square(m^2).
  - livingArea (double? LivingArea): Living area in metre square(m^2).
  - id (int Id): Index Id of the building.
  - address (System.String Address): Address of the building.
  - floorArea (double? FloorArea): Floor area in metre square(m^2).
  - floors (System.String Floors): Number of floors in the building.
  - propertyId (System.Double PropertyId): Property Id of the building:
  - postalCode (System.String PostalCode): Postal code in text.
  - city (System.String City): City in which the building resides.
  - buildingName (System.String BuildingName): the name of the building.

- constructionYear (System.String ConstructionYear): The year the building was constructed.
- buildingId (System.Double BuidlingId): Unique id of the building.
- heatedCapcity (double? HeatedCapacity): Heated capacity of the building.
- lotId (System.Double LotId): Building lot.

Resource relations:

- rooms (List<Room> Rooms): list of the Rooms in a building. HasMany relationship.
- lot (Lot Lot): Lot related to the building. HasOne relationship.

- **Deleted Service Requests**

  Attributes:

  - id (int Id) filterable: id of the deleted Service Request.
  - tyomnro (int Tyomnro): work number of deleted Service Request.
  - timeDeletedUtc (DateTime TimeDeletedUtc): time that the Service Request was deleted.
  - user (string User): name of the user who deleted the Service Request.

- **External Systems**

  Attributes:

  - id (int Id) filterable: Id of the External System.
  - lookupKey (string LookupKey) filterable: Lookup key for the External System.
  - name (string Name): the name of the External System.

  Resource relations:

  - service-request-external-system-ids (List<ServiceRequestExternalSystemId> ServiceRequestExternalSystemIds): HasMany relationship.

- **Fat Notifications (Beta)**

  Attributes:

  - propertyId (int PropertyId): the id of the property related to the Fat Notification.
  - roomId (int? roomId): the id of the room related to the Fat Notification.
  - startDate (DateTime StartDate): the start date of the Fat Notification.
  - endDate (DateTime EndDate): the end date of the Fat Notification.
  - priority (int Priority): the figure that indicates the priority of the Fat Notification.
  - message (string Message): Message attached to the Fat Notification.
  - id (int Id): the id of the Fat Notification.
  - dateModifiedUtc (DateTime? DateModifiedUtc) mutable: date the Fat Notification was modified.

- **Lots**

  Attributes:

  - propertyId (double propertyId) filterable: Property identifier.
  - area (double? Area): Area of Lot.
  - lotId (double LotId): Lot identifier.
  - lotNumber (string LotNumber): Number of the Lot.
  - id (Int32 Id): Id of the Lot.

Resource relations:

- buildings (List<Building> Buildings): list of Buildings in the Lot. HasMany relationship.

- **Properties**

  Attributes:

  - region (string Region): the Region of the Property.
  - address1 (string Address): the first Address of the Property.
  - address2 (string Address2): The second Address of the Property.
  - propertyId (int PropertyId): Property Identifier.
  - propertyName (string PropertyName): name of the Property.
  - propertyType (string ProperyType): type of the Property.
  - registrationId (string RegistrationId): Registration Identifier.
  - businessId (string BusinessId): business identifier for the Property.
  - notes (string Notes): extra comments as necessary.
  - postalCode (string PostalCode): Postal Code of the Property.
  - countryCode (int? CountryCode): Country Code for the Property.
  - city (string City): City in which the Property resides.
  - group (string Group): assigned Group of the Property.
  - identification (string Identification): Identification assigned to the Property.
  - id (int Id): Identifier for the Property.
  - customer (int? Customer): Customer attached to the Property.
  - customerID (int? CustomerId): Identifier for the attached Customer.
  - businessArea (int? BusinessArea): Business Area of the Property.

    - integrationSystem (string IntegrationSystem): Integration System

  Resource relations:

    - subContractorPropertyLinks(List<SubContractorPropertyLinks> SubContractorPropertyLink): List of the Subcontractors associated to this Property. HasMany relationship.

- **Purchase Order Items(BETA)**

  Attributes:

    - active (bool Active): indicates the state of the Item.
    - amount (double? Amount):
    - name (string Name): Name of the Item.
    - price (double? Price): Price of the Item.
    - purchaseItemsId (int PurchaseItemsId): Identifier for the Item.
    - purchaseOrdersId (int PurchaseOrdersId): Identifier for the related Purchase Order.
    - unit (string Unit):
    - vat (double? Vat): The value added tax.
    - id (int Id): Identifier for the Purchase Order Item.

  Resource relations:

    - purchase-order (PurchaseOrder PurchaseOrder): The Purchase Order associated with the Purchase Order Item. HasOne relationship.

- **Purchase Orders(BETA)**

  Attributes:

    - active (bool Active): indicates the state of the Purchase Order.

- buildingId (int BuildingId): Identifier for the Building associated with this Purchase Order.
- createdByUserId (int CreatedByUserId): Identifier for the User who created this Purchase Order.
- createdTime (DateTime CreatedTime): Time Purchase Order was created.
- deliveryAddress (string Address): Delivery Address associated to the Purchase Order.
- deliveryCity (string DeliveryCity): Delivery City associated with the Purchase Order.
- deliveryDate (Nullable<DateTime> DeliveryDate): Delivery Date associated with the Purchase Order.
- deliveryPostalCode (string DeliveryPostalCode): Delivery Postal Code associated with the Purchase Order.
- description (string Description): Description of the Purchase Order.
- dnPurchaseOrderStatusComment (string DnPurchaseOrderStatusComment): Comment about the Purchase Order Status.
- dnPurchaseOrderStatusId (int DnPurchaseOrderStatusId): Identifier for the Purchase Order Status.
- dnPurchaseOrderStatusUpdaterId (int DnPurchaseOrderStatusUpdaterId):
- dnPurchaseOrderStatusUpdaterType (int DnPurchaseOrderStatusUpdaterType):
- frameworkMasterDataCompaniesId (int FrameworkMasterDataCompaniesId):
- frameworkMasterDataPartnersId (int? FrameworkMasterDataPartnersId):
- handlerUserId (int? HandlerUserId): Identifier for the handler of the Purchase Order.
- title (string Title): The Title of the Purchase Order.
- id (int Id): Identifier for the Purchase Order.

Resource relations:

- purchase-order-item (List<PurchaseOrderItem> PurchaseOrder-Items): List of Purchase Order Items associated with this Purchase Order.

- **Rooms**

  Attributes:

  - roomNumber (string RoomNumber): The number of the Room.
  - intendedUse (Nullable<int> IntendedUse): The Room's intended use.
  - floor (double? Floor): The Room's floor number.
  - propertyId (double PropertyId): Identifier for the Property to which the Room is associated.
  - notes (string Notes): extra comments as necessary.
  - roomName (string RoomName): The Name of the Room.
  - area (double Area): The Area of the Room.
  - buildingId (double BuildingId): Identifier for the Building associated with the Room.
  - lotId (double LotId): Identifier for the Lot associated with the Room.
  - roomType (string RoomType): Type of the Room.
  - id (int Id): Identifier for the Room.
  - integrationRoomId (int? IntegrationRoomId): Integration Room Identifier.

  Resource relations:

  - building (Building Building): The Building associated with the Room. HasOne relationship.

- **Service Requests**

  Attributes:

  - heading (string Heading): Heading for the Service Request.
  - description (string Description): Description of the Service Request.
  - latestStatusComment (string LatestStatusComment): latest comment on the Service Request.
  - creationTimeUtc (DateTime? CreationTimeUtc): Time the Service Request was created.
  - priorityClass (double? PriorityClass): Priority Class of the Service Request.
  - faultType (int? FaultType): Fault Type of the Service Request.
  - dueDateUtc (DateTime? DueDateUtc): Due Date of the Service Request.
  - manualDueDate (bool ManualDueDate): Indicates whether the Due Date was manually set.
  - propertyId (double? PropertyId): Identifier for the Property associated with the Service Request.
  - lotId (double? LotId): Identifier for the Lot associated with the Service Request.
  - buildingId (double? BuildingId): Identifier for the Building associated with the Service Request.
  - statusId (int? StatusId): The status identifier for the Service Request.
  - parentId (int? ParentId): The identifier for the parent of the Service Request if it is a SubService Request.
  - incompleteSubServiceRequest (int? IncompleteSubServiceRequest): Indicator for an incomplete SubService Request
  - petsInRoom (bool PetsInRoom): indicator for the presence of pets in the room associated with the Service Request.
  - reporterEmail (string ReporterEmail): Email for the reporter of the Service Request.

- reporterName (string ReporterName): Name of the reporter of the Service Request.
- reporterPhoneNumber (string ReporterPhoneNumber): Phone number of the reporter of the Service Request.
- agreeOnVisitTime (bool AgreeOnVisitTime): Indicates whether an agreement about visiting time as been made.
- allowMasterKeyUsage (bool AllowMasterKeyUsage): Indicates whether consent to use the master key has been given.
- plannedStartDateUtc (DateTime? PlannedStartDateUtc): The planned Starting Date of the Service Request.
- plannedEndDateUtc (DateTime? PlannedEndDateUtc): The planned Ending Date of the Service Request.
- active (bool Active): indicates if the Service Request is in an active state.
- id (int Id): Identifier for the Service Request.
- roomId (int? RoomId): Identifier for the Room associated with the Service Request.
- dateModifiedUtc (DateTime? DateModifiedUtc) mutable: Date that the Service Request was modified.
- enableReporterEmailNotifications (bool? EnableEmailNotificationsForReporter): Indicates whether the Email Notification for Reporter is enabled.

Resource relations:

- service-request-external-system-ids (List<ServiceRequestExternalSystemId> ServiceRequestExternalSystemIds): HasMany relationship.

- **Service Requests External Systems ID**

  Attributes:

  - Id (int Id) filterable: Identifier for the Service Requests External System
  - serviceRequestId (int ServiceRequestId) filterable: Identifier for the associated Service Request.

    - externalSystemId (int ExternalSystemId) filterable: Identifier for the associated External System.
    - externalId (string ExternalId): External Id

  Resource relations:

    - external-system (ExternalSystem ExternalSystem): Represents the navigator which is mapped onto the association 'PurchaseOrderItem.PurchaseOrder - PurchaseOrder.PurchaseOrderItems. HasOne relationship.
    - service-request (ServiceRequest ServiceRequest): HasOne relationship.

- **Subcontractor Link Details (BETA)**

  Attributes:

    - id (int Id) filterable: Identifier for the Subcontractor Link Detail
    - area-of-responsibility (string AreaOfResponsibility): Subcontractor's area of responsibility
    - subcontractor (string Subcontractor): Name of Subcontractor
    - service-description (string ServiceDescription): Description of the service that the subcontractor will provide.
    - comment (string Comment): Comments
    - phone (string Phone): Phone number of Subcontractor
    - email (string Email): Email of Subcontractor
    - department-description (string DepartmentDescription): Description of the Department.

  Resource relations:

    - subContractorPropertyLinks (List<SubContractorPropertyLinks> SubContractorPropertyLink): HasMany relationship

- **Subcontractor Property Link (BETA)**

  Attributes:

  - propertyId (int PropertyId): Subcontractor's link to Property.
  - subContractorsId (int SubContractorsId): Subcontractor's Identifier
  - subContractorLinkDetailsId (int? SubContractorLinkDetailsId): Link detail id

  Resource relations:

  - property (Property Property): HasOne relationship.
  - Subcontractors (SubContractors SubContractors): HasOne relationship.
  - subContractorLinkDetails (SubContractorLinkDetails SubContractorLinkDetails): HasOne relationship.

- **Subcontractors (BETA)**

  Attributes:

  - id (int Id) filterable: Identifier for the Subcontractor.
  - name (string Name): Name of the Subcontractor.
  - business id(string BusinessID): Business identifier for the Subcontractor.
  - address (string Address): Subcontractor's Address.
  - postalCode (string PostalCode): The Postal Code of the Subcontractor.
  - city (string City): The city associated with the Subcontractor.
  - active (System.Boolean Active): Indicates whether the Subcontractor is active.
  - email (string Email): The Email associated with the Subcontractor.
  - emailLanguage (string emailLanguage): The preferred language in which the Subcontractor wishes to receive emails.

- externalPartner (System.Boolean ExternalPartner): Indicates that the Subcontractor is an External Partner.

- sendServiceRequestAssignmentEmail (string sendServiceRequestAssignmentEmail): Indicates that the Subcontractor wishes to receive Service Request Assignment emails.

- propertyId (int PropertyId): Property associated with Subcontractor.

- subContractorsId (int SubContractorsId): Identifier for the Subcontractor.

Resource relations:

- subcontractor-external-system-ids (List<SubcontractorExternalSystemIds> SubcontractorExternalSystemIds): List of the External Systems Ids. HasMany relationship.

- subcontractor-property-links (List<SubcontractorPropertyLinks> SubcontractorPropertyLin): List of Subcontractor Property Links. HasMany relationship.

- **Subcontractor Link Detail External System Ids**

  Attributes:
  - id (int Id): Identifier for the Subcontractor Link Detail External System Ids
  - subcontractorLinkDetailId (int SubcontractorLinkDetailId): Subcontractor Link Detail Identifier.
  - externalId(string ExternalId): External identifier.

  Resource relations:
  - subcontractor-link-detail (SubcontractorLinkDetails SubcontractorLinkDetail): Related to Subcontractor Link Detail. HasOne relationship.
  - external-system (ExternalSystem ExternalSystem): Related to External System. HasOne relationship.

- **Subcontractor External System Ids**

  Attributes
  - id (int Id): Subcontractor External System Ids identifier.
  - subcontractorId (int SubcontractorId): Identifier for the associated Subcontractor.
  - externalId (string ExternalId): Identifier for the External Id.

- externalSystemId (string ExternalSystemId): Identifier for the External System Id.

Resource relations:

- subcontractor (Subcontractors Subcontractor): Related to Subcontractor. HasOne relationship.
- external-system (ExternalSystem ExternalSystem): Related to External System. HasOne relationship.