Web API protocol and security analysis

Web API protokoll- och säkerhetsanalys

CRISTIAN ARAYA
MANJINDER SINGH
Web API protocol and security analysis

Web API protokoll- och säkerhetsanalys

Cristian Araya and Manjinder Singh

Degree project in
Computer science
First level, 15hp
Supervisor from KTH: Reine Bergström
Examiner: Ibrahim Orhan
TRITA-STH 2017:34
KTH
The School of Technology and Health
141 52 Flemingsberg, Sweden
Abstract

There is problem that every company has its own customer portal. This problem can be solved by creating a platform that gathers all customers’ portals in one place. For such platform, it is required a web API protocol that is fast, secure and has capacity for many users. Consequently, a survey of various web API protocols has been made by testing their performance and security.

The task was to find out which web API protocol offered high security as well as high performance in terms of response time both at low and high load. This included an investigation of previous work to find out if certain protocols could be ruled out. During the work, the platform’s backend was also developed, which needed to implement chosen web API protocols that would later be tested. The performed tests measured the APIs’ connection time and their response time with and without load. The results were analyzed and showed that the protocols had both pros and cons. Finally, a protocol was chosen that was suitable for the platform because it offered high security and fast connection. In addition, the server was not affected negatively by the number of connections. Reactive REST was the web API protocol chosen for this platform.

Keywords
Web API protocol, customer portal, performance test, REST, reactive REST, websockets
Sammanfattning


Nyckelord
Web API protokoll, kundportal, prestandatest, REST, reactive REST, websockets
Acknowledgements

This paper presents the thesis work done by two students from KTH and includes an analysis made on web API protocols and the task given by Linkle AB of developing a web platform.

During the thesis work, we had the supervisor Reine Bergström from KTH, whom we would like to thank for the help he provided.

We also want to thank Linkle AB, Filip Szymanski and Albert Olofsson for the opportunity they gave us as well as their support.
Glossary

**Web API** - Application Programming Interface over the Web. Defines how communication between systems interact.

**REST** - Representational state transfer, also known as RESTful web services. Connectionless web API protocol.

**RxREST** - Reactive REST, a REST variant implemented with reactive programming.

**SOAP** - Simple Object Access Protocol, protocol specification for web communication.

**RPC** - Remote Procedure Call, remote method invocation.

**WS** - websockets, connection oriented web API protocol.

**WSS** - websocket secure, WS with integrated security.

**HTTP** - Hypertext Transfer Protocol, communication-protocol for the web.

**HTTPS** - Hypertext Transfer Protocol Secure, protocol for encrypted communication over HTTP.

**JSON** - JavaScript Object Notation, text based format for data exchange.
### Contents

1. Introduction .................................................................................................................. 1  
   1.1 Problem definition ................................................................................................. 1  
   1.2 Objective ............................................................................................................... 1  
   1.3 Delimitations ....................................................................................................... 2  
2. Theory and background ............................................................................................. 3  
   2.1 History ................................................................................................................ 3  
   2.2 API ..................................................................................................................... 4  
   2.3 Web API ............................................................................................................. 5  
   2.4 Web services ...................................................................................................... 5  
      2.4.1 REST ........................................................................................................... 5  
      2.4.2 SOAP ........................................................................................................ 6  
      2.4.3 RPC ........................................................................................................... 6  
   2.5 Websocket .......................................................................................................... 7  
   2.6 Security .............................................................................................................. 8  
      2.6.1 REST security ............................................................................................. 8  
      2.6.2 SOAP security .......................................................................................... 9  
      2.6.3 RPC security ............................................................................................ 9  
      2.6.4 Websocket security ................................................................................ 10  
   2.7 Related work ...................................................................................................... 11  
   2.8 Performance test ............................................................................................... 12  
      2.8.1 Load impact .............................................................................................. 12  
      2.8.2 Apache JMeter ......................................................................................... 12  
      2.8.3 Neoload .................................................................................................... 13  
3. Methods .................................................................................................................... 15  
   3.1 Comparing APIs ................................................................................................. 15  
      3.1.1 Web services ............................................................................................ 16  
      3.1.2 Websockets .............................................................................................. 16  
      3.1.3 Selection of APIs ..................................................................................... 16  
   3.2 Application components ..................................................................................... 17  
      3.2.1 Apache Tomcat .......................................................................................... 17  
      3.2.2 Backend ................................................................................................ 17  
      3.2.3 Database .................................................................................................. 18
3.2.4 Security ........................................................................................................ 19
3.3 Testing ............................................................................................................. 20
  3.3.1 Selection of testing tools .......................................................................... 20
  3.3.2 Performance tests ...................................................................................... 21
  3.3.3 Test environment ....................................................................................... 22
4. Result ................................................................................................................. 23
  4.1 Response time ............................................................................................... 23
  4.2 Connection time ............................................................................................ 24
  4.3 Load test ........................................................................................................ 25
  4.4 Total response time on multiple requests ..................................................... 26
5. Analysis and discussion .................................................................................... 29
  5.1 Analyzing the method ................................................................................... 29
    5.1.1 Analyzing the test tool ........................................................................... 29
    5.1.2 Analyzing the tests ................................................................................ 29
    5.1.3 Analyzing the API implementations ...................................................... 30
    5.1.4 Analyzing the security implementations ................................................. 30
  5.2 Analyzing the results ..................................................................................... 30
    5.2.1 Comparing the result .............................................................................. 30
    5.2.2 Result discussion ..................................................................................... 31
  5.3 Possible performance improvements ............................................................ 33
  5.4 Potential impacts ........................................................................................... 34
  5.5 Social and economic aspects ........................................................................ 35
6. Conclusion .......................................................................................................... 37
  6.1 Future work .................................................................................................... 37
References ............................................................................................................. 39
Appendix ............................................................................................................... 45
1. Introduction

This chapter gives an introduction to the thesis and describes the existing problem as well as the objectives that were set. It also included the delimitations that were set from start.

1.1 Problem definition

Many companies offer a portal for their customers where they can look over their subscriptions and conditions. Using individual platforms for every customer portal on each company can be very frustrating and time consuming. Linkle is a start-up company that aims to come up with a solution to this problem. Linkle’s goal is to develop a web platform which will gather customer portals from every onboarding company and present them in one place.

The purpose of this work is set to find an optimal protocol for the application that will be developed. The goals will be achieved by analyzing different web API protocols and testing their performance against each other. The API needs to be able to handle several simultaneous users, for that reason the selected API must have a good response rate and be able to scale if the need arises. Security is also an important aspect within this platform because personal data will be handled in API requests and saved in a database.

1.2 Objective

The goal with this work is to analyze different forms of web API protocols and find a suitable one to implement on Linkle’s requested web-platform. To fulfill this goal, the work is divided into the following subtasks.

1. Research and document different APIs for web development:
   - commonly used API protocols used for web development
   - security approaches of selected APIs
   - speed and scalability of selected APIs.

2. Test of selected APIs
   - test the response time of selected APIs
   - test the request capacity and scalability of selected APIs when multiple requests are sent to the server
   - test the connection time of selected APIs.
3. Analyze test results and determine the best suited API protocol for the platform.

1.3 Delimitations

To be able to complete the objectives within the time set there had to be some delimitations.

- No dedicated front-end application will be designed.
- Backend does not need to provide all functionality (add, change, terminate subscriptions) as it will be a proof of concept for Linkle.
- Certificates for secure connection will be self-signed.
2. Theory and background

This chapter introduces the history of APIs, web APIs are presented in detail as well as the different types of web API protocols available in Java. The diverse protocols are described, how they function, strengths and weaknesses and how well they perform from a security point of view. There is also information of different kinds of tests that can be used to measure performance of each web API implementation. The chapter also reviews related work that has been done previously on APIs, especially web APIs.

2.1 History

Inter-process communication (IPC) has been in constant development as the need for interactive applications increases. Initial forms of inter-process communication consisted on local processes communicating with each other by sharing resources in local storage or in memory or by means of internal endpoints such as sockets or pipes [1]. Although these concern solutions within operating systems, software solutions have also been developed to fulfill the requirements of IPC on every platform i.e. RPC solutions.

A web application’s performance is considered to be its ability to scale and its response time. A web application is defined as scalable if by adding more hardware, the application is able to handle more requests than before. Scalability is hardly dependent of the software architecture, because if the architecture is not able to use additional resources to increase performance the application will not be scalable. [2]

Response time is the time it takes for the application to process a request and return a response. It is important to be aware of existing delays in network communication as they can interfere with the results of any response time tests.

It is possible to improve the experienced performance of a web application by implementing web caching. Web caching is a technological solution that aims to reduce the transmission of redundant network traffic, this can be done by saving copies of already retrieved data closer to the end user and allowing fast access to the same data in the future. Caching is beneficial to the majority of internet users, especially those still depending on slow
internet connections. The main reasons one would like to implement web caching are:

- to increase the bandwidth availability by curbing the transmission of redundant data
- for reducing network congestion
- for improving response times
- for achieving savings in terms of cost (for e.g. cost of bandwidth).

Many companies are opting for a reactive solution when it comes to high demand applications, in the meaning that the application should strive for asynchronous calls and non-blocking architecture in order to minimize the use of resources and blocking time of threads.

Asynchronous processing means that a servlet container’s thread delegates processing to another worker thread from a thread pool while the main thread can continue listening for incoming requests. If a non-blocking paradigm is used then threads are returned to the thread pool so they can be used to handle other processes instead of being blocked while waiting for a result to arrive.

### 2.2 API

The technical definition of an API is: a way for two software application to communicate with each other. API is a set of tools, routines and protocols to develop an application, which specifies how different components should interact with each other. An API can be used for several different purposes such as an interface for libraries, frameworks or even for an operating system. By using an API, the quality of a program raises and it can even speed up the development. The developer can hide the complexity of a program by using an API.

There are some characteristics that APIs need to be of quality. One of the aspects of a quality API is obvious and it is comprehensibility. Understanding the API and its architecture to be able to use it efficiently. A more concrete aspect is that an API should be consistent to the extent that it uses the same structure in all places. It should also be easy to maintain an
API so it can easily be updated by deleting, modifying or adding new methods. [7]

2.3 Web API
Application to application interaction over the network is usually established by using a web API as an extensible and a systematic strategy to communicate. Web APIs are the opposite of static linked API such as interface to a library. Web API is a way to integrate different services into applications. The web API is a predefined message system with a request-response system. Requests are sent using the web API protocol and are handled by the remote system which will process the request and respond with the result. [6]

There are some different web API protocols with the same purpose but diverse strategy. Among these, there are two, called web services and websockets. Both these two work as a web API but differ from each other on how they communicate and what characterizes them. [8]

2.4 Web services
A web service is a software system designed to handle request from remote systems, enabling what is defined as machine-to-machine interaction. Communication between these systems is normally done over the World Wide Web. [9]

There are many web service architectures; among those REST, SOAP, XML-RPC and JSON-RPC.

2.4.1 REST
REST is an architectural style that defines the architectural quality attributes of the World Wide Web. REST has become very popular over the years for its simplicity and its stateless and decoupled qualities. Its architectural style derives from a set of well-known styles such as layered and client/server architecture. REST offers high performance and scalability by facilitating replication on the server side and on intermediary layers.

The main characteristic that differs in REST compared to other approaches is the definition of a uniform interface to be shared among all architectural
components. It requires that web resources are identified by global and unique addresses, called Uniform Resource Identifiers (URIs). Client and servers communicate by exchanging request and response messages, which contain both data and metadata. Formats of representation can be negotiated and vary according to the client context, capabilities and interests. This ensures the way data representation is parsed, processed and understood.

Messages follow a network protocol that indicates how they should be handled. For instance, while using the HTTP protocol, methods GET, DELETE, POST among others are used to access or modify web resources. [10]

Recent development have altered the way REST implementations work and have begun to adapt reactive programming leading to the start of reactive REST (RxREST). To take the full advantage of RxREST services, it is necessary to use non-blocking and asynchronous connections to files, other services and databases. [11]

2.4.2 SOAP
SOAP allows messages to be exchanged between systems using object method invocation. SOAP is a more sophisticated protocol compared to XML-RPC as message composition is constrained by the XML schema and message validation can be enforced. SOAP requests are embedded in HTTP request and response messages. [12]

SOAP is a more complex and heavyweight protocol as data is sent in XML format and this is known to contain more overhead if compared to how data is sent for example in REST. Benefits of SOAP are that it has built-in support for security, atomic transactions and reliable messaging. [12]

2.4.3 RPC
A remote procedure call (RPC) is when a computer creates a procedure to be executed on a remote system. The method call is coded similarly as if the requested method was located in the same system. The simplicity lies in that the programmer does not need to code the remote invocation part. There are several variants to RPC but only two will be explained in more detail, XML-RPC and JSON-RPC. [13]
XML-RPC is a remote procedure call protocol adapted to the Internet. In this protocol, HTTP POST requests are used to send XML-formatted messages across systems. Messages exchanged may contain procedure invocation, a return value, or an error notification. Parameters supported in these messages are of data types: integer, boolean, string, double, base64, struct or array. If a procedure call fails, the returned message will contain one or more error codes. [13]

JSON-RPC is a stateless, light-weight remote call procedure protocol. It uses JSON as its data format, hence JSON-RPC. JSON-RPC is transport agnostic which means that it can be used within the same process or in different message passing environments. Messages are of type request or response, both containing an id that identifies different calls from each other. [13]

2.5 Websocket

Websocket is a stateful protocol that supports full-duplex communication and is working on the application layer. Websocket is a protocol based on the TCP protocol and uses HTTP for the handshake in the beginning of a connection. Handshake is done by the client sending a request to upgrade the HTTP protocol to websocket. The meaning of full-duplex is that the communication can take place in both directions simultaneously, much like today’s phone communication. Websockets have a callback function that can be called when receiving a message and work as a response method. This callback function can either be blocking or non-blocking depending on the implementation. [14]

Establishing a connection using websockets is done as said previously; by a handshake. First step is to set up a TCP connection, then to establish the websocket connection, a GET request is sent to the HTTP server. The server receives the request from the client and responds to the handshake. When the response from the server is received by the client a connection has been established. Now both parts of the connection are listening for incoming data and simultaneously able to send data. The connection is persisted until one part of the connection decides to close the websocket or in case of an error occurs in the communication channel. When one part closes the connection the other part will automatically close the connection and the session will end. [14]
Websocket communicates through a single socket, it does not need to continuously add HTTP-headers as in HTTP requests. By discarding this extra header the amount of data sent over the web for each request is reduced, which leads to less overhead. This leads to faster communication and more effective applications. [15]

2.6 Security

When it comes to the web and the Internet, security is an issue of importance. Sensitive data such as personal information and bank credentials, to name a few, is stored and transferred on the Internet. This information must be kept safe and away from unauthorized access. This can be achieved by developing a secure application, where all the information that is send over the network goes through a secured channel. Sensitive data should be stored securely for example in databases where only authorized personnel have access. Another security approach is to encrypt data sent over the internet to make it unreadable, this can be achieved by different methods. [16]

2.6.1 REST security

REST services can be secured by using different security methods, one of them is by making every REST call to attach an API key unique for the service that functions like a password enforcing authentication. Another solution is to create session tokens when a service is initially requested, the session token may then be used like an API key to authenticate every request. It is possible to store API keys and tokens as cookies in a web browser and set an expiration date. [17]

As mentioned earlier in the thesis, REST uses the HTTP protocol for communication. It is possible to secure HTTP protocol by applying HTTPS on top of it. HTTPS enables secure data transmission by using encryption, where a key used for encryption is agreed upon connection establishment. Encrypted information is then only readable for those who have access to the key, even if the transmission between two ends is eavesdropped the information acquired will not be readable unless the right decryption key is used to decrypt the message. [17]
2.6.2 SOAP security

SOAP protocol has some standards that have been built to mitigate the security problem. These standards are made to bring a basic level of security to SOAP web service, such as message integrity. [18]

Transport Layer Security (TLS) is one of the methods to make SOAP more secure. It is a protocol that uses cryptography for communication over a channel, which creates a secure path between the two parts involved. TLS works on a lower level in the OSI model than SOAP, which means it can only give limited security. TLS provides point-to-point authentication and security in the transport. [18]

Another method that can raise the level of security in SOAP is by implementing an XML signature, which is a standard by the World Wide Web Consortium (W3C). By using hash-code and cryptography, you can select parts of the XML document to provide integrity verification. XML signature provides security from the end of a node to another. There is another method involving XML document and consists in encrypting the document, which gives confidentiality to the message under the circumstances that the application does not send the decryption key. [18]

Another way to make SOAP more secure is to implement Web Service Security, which is a security standard. This method is a combination of XML encryption, XML signing and includes tokens. Web Service Security provides more security, in addition to the XML signing and XML encryption, it also provides identification of the sender. Web Service Security has also the ability of using timestamps on the message which defines when the message was created and when it expires. This information allows the receiver to be informed if the message’s security may have been compromised and can then decide to ignore the message. [18]

2.6.3 RPC security

RPC is fairly insecure because it lacks authorization and authentication. RPC traffic is not secure either, which means the traffic can be intercepted. However it is still possible to secure RPC traffic by implementing another protocol on top of RPC such as HTTPS, which enables secure traffic and adds authorization and authentication. [19]
2.6.4 Websocket security

Websocket is no different to web services or other technologies in the web regarding security issues. The majority of the security control that is available in today’s web is also available for websockets. The key factor to make websockets secure is to implement these security controls in an appropriate way. [20]

Websockets do not have any authentication control on handshakes during session establishment, hence clients cannot authenticate themselves to the server. However this can be managed by implementing a form-based authentication. This can be done by using HTTP, the server sets a cookie and passes a token that is associated with the metadata to the client which stores it on the web browser. Each time the client makes a request, it attaches the token and the server can then verify the client by checking if the token matches. [20]

Websockets have neither any security measures regarding authorization. This issue is mostly application-based, where unauthorized users can have access to data. This can occur when someone is eavesdropping the communication, also known as man-in-the-middle attack. By encrypting the message the access to data is reduced to only those in possession of the decryption key, even if users can access the resource. [20]

It is possible to raise the security level of websockets by disabling cross-domain requesting. Cross-domain requesting works by adding a header to the HTTP-request in the beginning of a handshake. This header is called origin and protects the websockets from cross-origin that does not have permission. By adding the origin header the server can see where the request came from and if the server refuses a connection from this origin, it will be rejected. [20]

Furthermore websockets need protection against input validation attacks, which can either attack the client or the server. Input validation attacks are when an attacker sends unexpected input to the application, which can alter the behavior of the application. The best way to prevent these types of attacks is to divide the message and remove malicious input. There are
frameworks and libraries that provide simple input validation functions which can handle these attacks. [20]

2.7 Related work
Empirical research has been done on the evolution of web APIs and different strategies to develop them. A case study has been made regarding providing current information that includes a various set of web APIs. The case study used new data sources for its analysis and its purpose was to see how different methods changed while APIs evolved. [21]

Web API protocols, SOAP and REST, have been compared with one another, deciding which web service is optimal. Comparing their strengths and weaknesses, a conclusion could be decided on which one to use. The conclusion of this work was that REST uses more standards and up-to-date solutions than SOAP, for example REST has support for more data formats. In conclusion REST was the more optimal choice [22]. This thesis will continue the study by comparing and examining if there are other options that could be considered as a better web API than REST and SOAP.

Security aspects among websockets, SOAP and REST have been analyzed as well as their pros and cons. There is also a study made on SOAP contra REST regarding their security. The result of this work was that SOAP had better security when it came to message-related security means. This work concluded that REST had some flaws in its security but future works would focus on solving this issue to get a more secure web service. [23]

Websockets security has been analyzed by performing security tests in previous work. This work included different types of security solutions which could be applied to websockets. Websockets have support for a high level of security but to achieve this, the implementation needs to be proper. The result of the security test concluded that it was possible to achieve a high level of security. [20]

An analysis made on performance of web services came to the conclusion that RESTful services are more suitable for mobile applications because they create faster responses due to its less overhead comparing to conventional SOAP web services. It was also analyzed how fast data is parsed in both XML
and JSON concluding that parsing JSON objects is much faster than parsing XML objects. [24]

2.8 Performance test

Performance tests are conducted to test the application’s quality. The goal with these tests is to find possible improvements in the application’s performance and potential bottlenecks. Applications that satisfy the client, are responsive, user friendly and have no errors are considered to be of quality. Different qualities can be tested such as response time, how stable the application is and how well the response time is in proportion to how many requests are made, also known as load test. These tests provide a more concrete value of the application’s quality. [25]

There are many different tools that are available for testing applications. These tools will make it easier to test performance on APIs and make it possible to see the strengths and weaknesses of them.

2.8.1 Load impact

Load impact is an online tool for creating load tests. Tests can be performed on REST and SOAP APIs. Load impact has support for chrome extension and Postman REST, which makes it possible to record API requests and automatically be able to generate a test script. This tool offers the ability to measure the application’s performance as well as validate the scalability of the server. It also helps to determine the maximum capacity that the API protocol can handle. There is a possibility to set a threshold, to see if the response time is sufficient to satisfy the end users. [26]

2.8.2 Apache JMeter

Apache JMeter is a tool for measuring performance and load. JMeter is a Java application that is open-source, it was developed to test web applications but can now also be used to run performance tests on both dynamic and statistic resources. Apache JMeter allows to perform tests on various protocols and servers such as web servers, SOAP and REST protocols, TCP and also websockets to name a few. Multithreading is also supported, which makes it possible to test how the APIs handle concurrency. It is also possible to choose load statistics with pluggable timers. Latest version of JMeter at writing time was 3.2. [27]
2.8.3 Neoload

Neoload is a tool for performance and load testing. Neoload has websocket and push support, which enables users to perform test on applications that use websockets and also has the capability of recording and playback. It also has support of handling binary and text messages. Different types of results that can be obtained by using Neoload are connection time and response time. It is also possible to see an application's behavior under stress and validate its performance. Latest version of Neoload at writing time was 5.5.2. [28]
Theory and background
3. Methods

The purpose of this chapter is to describe the different kind of performance testing tools, methods chosen to fulfill the goal and the reasons behind the choices. It also describes how the chosen APIs were implemented in the system, how the tests were conducted and how different security approaches were implemented.

The first step to reach the goal of this thesis was conducted through literature study. The primary reason of this literature study was to get a fundamental knowledge about the different web API protocols that could suit the platform Linkle wanted. This was done by studying scientific articles about different the protocols available and get familiar with the characteristics of each protocol. By analyzing related work, it was possible to obtain a more comparative result of existing protocols, like the level of security provided by each protocol. The literature study did not only provided fundamental knowledge about different protocols, but also about available testing tools.

The next step was to determine which web APIs should be involved in the tests, as well as to determine which testing tool to perform the analysis. This was done by studying and comparing the result from the literature study previously done. After selecting the protocols and the testing tools, the implementation of the chosen APIs would begin with possible security-methods.

The final step was to perform the chosen tests on each web API and analyze the result of the tests. By comparing these results, a final API protocol could be selected as the protocol for Linkle’s platform.

3.1 Comparing APIs

This section presents and compares the possible APIs that could be implemented in the application as well as their advantages which could be beneficial. From the preliminary investigation, it emerged that the alternative protocols that could be implemented in the platform are web services and websockets. The different types of web services are REST, SOAP and RPC. By analyzing pros and cons of the web API protocols presented it was possible to decide on which protocols to choose for testing purposes.
3.1.1 Web services
From the web services observed in section 2.4 it was concluded that the most used and modern option of API protocol was REST because of its lightweight and decoupled architecture, which SOAP and RPC hardly provide [22]. REST is currently the dominating web service on the market because of its decoupled characteristic, how well developers are able to re-accommodate resources, and the advantage of still having a working service without the need to modify the client application. The use of Reactive REST makes it a valuable API to use in combination with other non-blocking IO frameworks in order to achieve a full asynchronous system.

When it comes to the security of the various web services, the preliminary investigation has revealed that SOAP has the best security among the three web services that were included in section 2.4. REST had a high level of security but not as high as SOAP, even if the difference is not that big and has no major decisive role for this platform [23]. RPC, on the other hand, had almost no security at all and the lack of security is a big reason why it was discard as an option.

3.1.2 Websockets
The use of websockets is often a solution for real-time web applications, but it could suit this platform. Because websockets run in full-duplex, the application has the possibility to run asynchronous calls, which would provide faster response time when parts of the application can be executed non-blocking. Other strength of websockets is that it does not send a lot of overhead due to websockets does not use HTTP as web services do, which leads to faster response time. Websockets runs over TCP and the connection is always up until one endpoint ends the connection. [29]

When it comes to the security of websockets, it would be possible to achieve the same level of security as web services running over HTTPS. This could be done by establishing a websocket connection with HTTPS and then run the connection over secure websockets, which is equivalent to HTTPS but for websockets. Data is encrypted when transmitted.

3.1.3 Selection of APIs
The selection of which APIs that should be implemented and tested is based on what is considered the most used and common on the market today and a
new fast growing variant of it. Different alternatives have also been taken into account which are completely different in means of communication protocols and security solutions. That is why REST, RxREST and websockets were chosen.

REST is the most common web service today and since it was developed. The numbers of companies going over from SOAP (which was the biggest web service before REST) to REST have increased lately [30]. REST seems like an obvious choice to implement in the application since is the most used API by companies. The tests will determine if the biggest web service is the best in this case. Reactive REST is a new and growing variant of REST. This method is running asynchronous and can possibly be the reason why it can compete with traditional REST. Websockets is also included in this survey because it is also a popular web API protocol but more used in interactive applications. Websockets work very well on real-time applications and because the application needs to be responsive it is very likely that websockets are a strong alternative. These three protocols also have the ability to provide security measures and that is another reason why they are relevant for this thesis

3.2 Application components
This section means to explain the different parts of the application and the chosen methods that act together with the running application.

3.2.1 Apache Tomcat
Apache Tomcat was the choice of servlet engine due to a lot of experience using Tomcat and because it is open source making it modifiable if needed. Tomcat was the most used java application server last year in 2016 and it has been the most used since 2013, which makes it a comfortable choice as a servlet [31]. It was implemented on a server running the backend of the application. Tomcat is an open source servlet to create an environment for Java web applications. The tomcat version used in this thesis was 8.0.41. [32]

3.2.2 Backend
The backend was the most important part of the application and a lot of focus was put on it, as it would be the core element for the thesis. The backend was developed in Java 8 and it contained the 3 chosen APIs that would communicate with the future frontend application.
There were several frameworks that implemented Java’s API for RESTful services (JAX-RS) including Jersey and RESTEasy. The framework chosen was Jersey due to previous knowledge about this framework and is somewhat standard. The choice of using a framework instead of plain JAX-RS is to ease the implementation of REST. Jersey extends the JAX-RS API and is the reference implementation, which is ready for production without any complications. Another reason Jersey was chosen was that it works well on MVC (Model-View-Controller) architecture, which was applied on the platform. Jersey did also have support for object serialization, which was used to parse objects to JSON-format. From previous studies in the course “Server Software Development”, it turned out that Jersey worked efficiently on Tomcat. This was time saving, so the thesis could be done within the timeframe set from the beginning.

There were also several alternative frameworks for websocket such as Jetty websockets, Resin websockets and Javax websockets. The choice of framework became Javax websockets because the framework is supported by Tomcat as well as many other servlets, which makes it easy for developers to switch servlet without any problem. There was also a lot of documentations of this framework available on the web, which made it easy to implement. Javax is developed by Oracle to be used in Java applications and suited the platform as it is developed in Java.

The backend followed the MVC architecture and database entities presented by Linkle were created for use towards the database. The purpose of the backend was to serve the client application through the selected APIs, handle database queries and to communicate with external companies and fetch data from them.

3.2.3 Database
In order to perform the tests during the development phase, a preliminary server database solution was chosen in the form of MySQL database, which is a relational database. Since this solution had already begun, it was also used for testing. The choice of using a MySQL database was the fact that it is a relational database. A relational database suits the platform because of the entities in this platform are companies and user, which are related to each other. Relation between entities are much more complicated to achieve using
Methods

Object oriented databases. The database is not the center of this thesis and the choice of database was MySQL because of previous knowledge and it was also a suggestion from Linkle.

The use of Hibernate framework facilitated the object relational mapping between entities. The strengths of using Hibernate is that the developer can persist objects directly to the database instead of handling all data types within an object separately. Hibernate uses JPA (Java Persistence API), which works very well in Java. JPA provides the opportunity to express itself in Java entities instead of SQL tables and columns as well as it raise performance due to the use of caching [33]. The choice to use Hibernate was based on the fact that no need for manually handle the database. By writing the classes with Hibernate annotations, the database was created to mirror these classes. This was effective as time was saved, which meant that more focus could be placed on the key parts of the thesis.

3.2.4 Security

The final platform will handle sensitive data and personal identity numbers, hence the implementation of security was a must-have on the selected APIs. The impact in performance this might have will be later analyzed and presented during performance tests results.

3.2.4.1 HTTPS

Data is sent as clear text in HTTP requests and this can be intercepted by man-in-the-middle attacks or by eavesdropping the communication channel. Because the intended application will handle sensitive information about users, HTTPS will be used in order to encrypt the data sent. For two systems to be able to communicate through HTTPS it is required that both parts possess a self-signed certificate, this certificate is the provider of the encryption key used for the encryption of data. Anyone can generate a self-signed certificate by using Java’s keytool which is a key and certificate management utility [34]. The choice of using HTTPS was due to the face it is a strong security method. Big companies such as banks and markets uses HTTPS for transmission of sensitive information. Because the platform also will handle sensitive information it was a major reason why it was implemented on the platform.

3.2.4.2 WSS

The choice of websocket implementation was pretty obvious because the security aspect is something that is highly valued in this project. The choice
was to use WSS (websocket secure) instead of WS (websocket) due to WSS has higher level of security but for performance reasons both WS and WSS will be tested, as mentioned earlier. WSS was implemented because it is the equivalent of HTTPS for REST. The implementation of WSS was done by having a self-signed certificate on the tomcat server. This certificate was then used by clients to connect via the WSS and show the server that this is a trustworthy client. The use of WSS enabled the server and client to send encrypted messages through the socket. As mentioned earlier, websockets establishes a connection with a handshake over HTTP. Using HTTPS instead of HTTP will make the connection part more secure because the information shared during handshake will be encrypted.

### 3.2.4.3 JSON web tokens

The platform used a token-based authentication with the implementation of JSON web tokens. This was implemented in such way that when a user logged into the application, the server created an encrypted token that was signed with the server’s key. The token was associated with the user’s personal identity number hence the user did not need to send the identity number in each request. After creating the token, it was sent back to the client. When the client sent a request to the server, it attached the token within the request. If the server detected an invalid token it would discard the request. It was possible to put a lifetime on the token, which meant that the token was valid for a certain amount of time [35]. By implementing JSON web tokens, the level of the platform’s security was raised further. This implementation integrated an authentication method into the system.

### 3.3 Testing

In this section it is presented the choice of testing tools and the reasons behind the choice, as well as the different types of tests that were conducted to compare speed and scalability of the APIs.

#### 3.3.1 Selection of testing tools

The selection of which testing tools to be used was dependent on resources. All of the tools that were selected in 2.8 were free to use. This can affect the accuracy of the results but in this case they were sufficient for the test purposes.

The chosen testing tool was Apache JMeter. One of the reasons JMeter was chosen was because it is one of the most common testing tools and it also has
support for both websockets and REST services. The other tools supported either websockets or web services but not both. Having different tools for each protocol could lead to unfair or misleading results, hence another reason JMeter was chosen. JMeter is an open source application which has a lot of documentation on the Web. This facilitated the implementation of the tests as guides existed on how to adjust the settings in JMeter for each test.

3.3.2 Performance tests

Performance tests were done in order to get a concrete perception of how well a particular web API protocol performs. By executing three kinds of tests, load test, scalability test and response time test, it was possible to validate the performance of each protocol.

Load and scalability tests were performed with the help of the testing tool to see how fast the response time was of one request. By making several requests simultaneously simulating concurrency to the server, it was possible to observe how well the API response time scaled under load. Another way to know how well the API scaled was to overload the server with as many requests as possible and have a threshold to see how many requests it could handle up to a certain time. Response time test with several request was also performed to analyze how response time grew for certain amounts of requests. In cooperation with Linkle a value for how many requests per minute the backend needed to handle was estimated, this value came to be 20 requests per minute and user. The platform was estimated to handle 2000 simultaneous users leading to a total value of 40 000 requests per minute.

The tests were performed with different protocols, comparing response time. The main reason this test was included in the method was to measure how the performance of a protocol changed when security measures were implemented. HTTPS was compared against HTTP and websocket secure against websocket.

All the tests were performed through a wired network connection because the packet loss rate and latency are significantly lower compared to wireless networks.
3.3.3 Test environment

The test environment consisted of two separate servers, one server running the backend application and the other running the database. These servers were virtual machines running on KTH Flemingsberg’s servers. Both servers had CentOS 7, 64-bit as the main operating system. The hardware specifications of each server was:

- Intel Xeon running at 2.66GHz, 4 core CPU
- 4GB RAM memory
- 21.5GB HDD
- 1Gbit network adapter.

A home computer on a separate network was used to measure the tests’ response time. The home computer used was a desktop unit running Windows 10, 64-bit. Hardware specifications:

- AMD FX 8350 running at 4.2GHz, 8 core CPU
- 16GB RAM memory
- 1Gbit network adapter.

Load test was conducted within KTH’s network in their institution located in Flemingsberg, due to the first network had some limitations regarding the amount of concurrent requests that could be attempted. The computer used for this test was a Macbook pro running MacOS Sierra and the hardware specifications were:

- Intel Core i5 running at 2.9 GHz, 4 core CPU
- 16GB RAM memory
- 1Gbits network adapter.
4. Result

In this chapter the test results are presented, as well as performance values and charts based on test results. There will also be conclusions drawn from the results. The tests were based on the different protocols that were selected in section 3.1.3. Each test consisted of sending 10 000 requests of the same type to the backend, the amount of requests was chosen to be high in order to get a more precise value. Finally, the results of the various API protocols were compared.

4.1 Response time

The first conducted test involved the APIs’ response time. It indicated how fast it was to send a request and receive corresponding response. This test does not take the connection time into account and the keepAlive option on JMeter was turned on which persisted the connections. Measured time is assigned milliseconds. Figure 4.1 shows a diagram of the response time it took for each protocol to respond to a user’s register request. This command checks if the user exists and if not the system registers the user by adding it to the database. Standard deviation is not included in the figure because it grew too large at 500 concurrent users, for full content over results and standard deviation see Appendix table 1.

![Figure 4.1: Average response time for a register request.](image)
The diagram in Figure 4.2 is constructed in the same way as in Figure 4.1 but in this case, the results show the response time of a login request, see Appendix table 2 for full results. This request validated the login credentials and returned a token if the credentials were valid.

![Figure 4.2: Average response time for a login request.](image)

As it can be seen in the diagrams above, there was a big difference between the two different requests and the difference occurs in the REST protocol. The REST API handled requests synchronously. The response was only sent after a user had been registered. On the other hand the other protocols functioned asynchronously, a response was sent back before the user was registered in the database. There was no big difference between all protocols when it came to login requests as the only action at the backend was the token creation which all protocols had to wait for.

### 4.2 Connection time

The second test made on the various web API protocols was connection time. It showed the time it took for each protocol to connect itself through a possible handshake. The chart in Figure 4.3 shows how long time it took for
the different protocols to connect and send a login request until it received an answer. The error bars are representative of the standard deviation.

![Average connection time for a login request](image)

The results in this diagram showed that REST variants over HTTP had the fastest connection time while websocket delayed longer probably due to the initial handshake. When it came to the more secure protocols, HTTPS and WSS, the faster ones were still the REST variants. In conclusion the more secure protocols take longer time to establishing a connection than the ones without security.

### 4.3 Load test

The third and final test performed on the APIs was load test. The purpose of the load test was to measure how well an API handled concurrent users and potential bottlenecks within the system. Users were simulated in JMeter by threads running concurrently on the client computer.
The request used for this test was a login request. In Figure 4.4 it is possible to see the average response time of the APIs while the amount of simultaneous users increased.

As seen in the figure above, the response time increased linearly with the numbers of users for the more secure protocols, HTTPS and WSS. However, the protocols with no security did not seem to be affected at all. Although there is a slight increased response time at some points.

4.4 Total response time on multiple requests

The fourth test executed on the implemented web API’s measured the total amount of time to execute all the specified requests. This test measured the response time of a login request originally from one and the same client. Figure 4.5 presents the statistics from the unsecured web API’s and Figure 4.6 presents the statistics from the secured web API’s. The x-axis represents the amount of requests sent to the server.
The conducted test demonstrated how well websockets perform during longer message exchanging. It can be seen from the figures that in the beginning websockets performed worst among the web API’s due to its long
connection time but eventually it caught up in performance and became the faster web API. Websockets performed better than the REST API’s after 480 requests on figure 4.5 and after 130 requests on figure 4.6. Table over results can be seen on Appendix table 3 and Appendix table 4.
5. **Analysis and discussion**

The contents of this chapter are information regarding the thesis work as a whole. The results and methods are being examined as well as investigating the possible factors that may have had an impact on the tests. Furthermore, the possibility of performance improvement is also mentioned and what is the benefit of performing these tests. Social and economic aspects are discussed.

5.1 **Analyzing the method**

This section discusses and evaluates the chosen methods for this thesis.

5.1.1 Analyzing the test tool

All the tests were performed with JMeter, which simulated every user with a unique thread. This does not mirror reality where incoming requests may originate from different clients at different geographic locations. JMeter was chosen because it had support for both websocket and web service. However, no other comparison was made on the various alternative testing tools and JMeter. Therefore it might be a questionable choice of testing tool.

A problem that arose during the tests was that JMeter handled the average time in integers which led to misleading results when comparing the protocols against each other, for example during login tests in section 4.1 where REST, RxREST and WS had similar time with no load. This issue was resolved by copying the full test results to Microsoft Excel where a more precise average time and standard deviation could be calculated.

5.1.2 Analyzing the tests

The conducted tests, except load test, were performed from the same client computer on a home network. The average response time of load test was much less than the rest as load test was conducted within the same network as the servers. However, this does not affect any protocol in particular because all conditions were the same for all protocols.

The load test that was conducted is not realistic because the possibility of 2000 simultaneous users connecting to the network and making requests at the exact same time is very low. This scenario could be a possibility in the future, if millions of users join the platform. Then the result would be the same as shown in Figure 4.4.
The tests have the limitation that measured data was of the same request. As seen in response time test, the time between register and login requests varies. Therefore the response time is not general but the protocols result are fairly same.

5.1.3 Analyzing the API implementations
As mentioned earlier, the choice of the framework for REST and websocket was Jersey and Javax websockets. These frameworks were not tested against other frameworks, which mean they might not be the most effective frameworks. Comparing the different frameworks would be optimal but it was not possible due to the time limit of this thesis.

5.1.4 Analyzing the security implementations
The security choice of using HTTPS and websocket secure is somewhat equivalent in the level of security it provides. Both these protocols provides trusted authentication as well as encrypted message transmission by using self-signed certificates. The disadvantage of self-signed certificates is that web browsers do not regard the requested service as a trusted source, since the certificate is not known by the browser. This solution works fine in the development state for the tests but should in production a certificate authority approved certificate should be used.

5.2 Analyzing the results
In this section, the results are analyzed and discussed. Differences between the selected protocols will thus be discussed in this section and how security versus performance affects the outcome.

5.2.1 Comparing the result
All the web API protocols were tested with and without implemented security, to be able to detect the performance impact of security. In all of the tests, the result showed a major difference between the ones with security and the ones without. The protocols with security implementation had worse performance in the response time and the connection time as well as in the load test.

The result of the response time test shows that websockets without security implementation has the fastest response time both with and without load. Of
the security implementation protocols, it appears that even here websockets have the fastest response time. Somewhat remarkable on the response time test is that REST has less margin in response time compared to the remaining protocols in login request compared to register request. Noteworthy is that the protocols with security implementation increased exponentially while the protocols without increased linearly during response time test. The number of users affected the backend, the more users connected lead to bigger variation in response time. As it can be seen in Appendix table 1 and table 2, the standard deviation becomes higher under load, which means the response time can be very varied.

During connection time test, the result showed that websockets are the slowest to connect to the server while both REST variants are the fastest with similar time. The biggest difference in the test was the connection time for WSS compared to the REST variants over HTTPS. WSS takes in average 60ms more than WS to connect meanwhile HTTPS takes in average just below 15ms more than HTTP.

The result of load test showed that WSS had the slowest time to connect and send a message when multiple concurrent users were connected. The REST variants over HTTPS had very similar time compared to each other and were faster than WSS. These three protocols increased linearly in relation to the number of concurrent users. The protocols without security implementation have similar response time and the number of users has no significant effect on the response time.

From the response time on multiple requests test, it is shown that the REST variants are faster initially, but the longer the interactions the faster the websockets gets. This applies both when the different protocols are with and without security. WSS gets faster than RxREST over HTTPS after about 90 requests and after about 120 requests it will be faster than REST over HTTPS. When it comes to WS, it takes some more requests before it is possible to see a clear difference. After about 480 requests WS becomes faster than both of the REST variants.

5.2.2 Result discussion
REST variants use HTTP protocols to communicate while websockets communicate over TCP. Websockets must set up a socket to establish a
connection for communicating, which REST does not need because it uses HTTP. By comparing these two methods, the results were quite expected to be as they were.

From the test results, it is possible to see how the security affects the performance and vice versa. It turns out that with a higher level of security will lead to performance taking damage. It is important to choose either to have high security and that performance is moderately impaired or to have high performance but have worse security. The choice depends entirely on what the product requires. If the information cannot be leaked, it is better to focus on a security implementation variant. The biggest difference with security was that performance deteriorated under greater load, which can be seen in both the response time test as well as in the load test.

The protocols with security implementation led to a slower response time. This is expected since every time the client or the server is sending a message it needs to encrypt this message. When the receiver is receiving the message, it needs to be decrypted before handling the request or response. Encrypting and decrypting the message takes time and therefore the protocols with security implementation have longer response time.

Websockets has slower connection time, which was expected because of the handshake in the beginning of a connection. The connection time increases even more if the websockets use a security implementation. This is because handshake will take place over HTTPS, which in turn also makes a handshake. This causes the connection time of WSS to be slower and that is also why the HTTPS connection time takes more time than HTTP.

The advantage of websockets is that the connection between the systems is up until one part of the connection closes it. This allows the response time to be faster than REST because the client uses the socket to communicate with the server. In REST case, each request must be sent with additional information to the server, including, the server information. This makes the message bigger and takes longer time to send. The disadvantage of websockets is that the server must have a connection up for each client, which uses server resources such as memory. The more users online will lead
to more load on the server. This scenario does not occur with the use of REST.

For an application that has high interaction with a server and security is needed, it would be efficient to use WSS because of the faster response time. The slow connection time will then not be of great importance. Otherwise if the interaction is less than 130 requests per run and connection time has to be fast, the REST APIs over HTTPS are a better choice.

As can be seen from the result in Figure 4.5 and Figure 4.6 there is an interception where websockets eventually gets a faster response time than the REST variants. After the interception point is when it becomes more effective to use websockets.

The difference that can be seen between REST and RxREST was primarily seen on the result of the response time test. The big differences were between the two different requests that were tested, login and register. This is because all methods do not have an advantage of running reactive. The registry request can be run responsively efficiently, it provides a much faster response time than REST. However, in login request, RxREST was slower than REST, due to login cannot be run asynchronously. RxREST uses more resources than REST because it tries to allocate a new thread to the task which led to performance deterioration.

5.3 Possible performance improvements

During test runs, especially under load test, it was noticeable that requests involving HTTPS took longer time to be handled by the server leading to increased response times for concurrent requests. After inspecting the server’s performance during tests it was visible, as shown in figure 5.1, that the processor used almost 100% of its process capacity which created a bottleneck for incoming requests. In future work it would be recommended to use faster hardware, it would then be possible to handle a bigger amount of requests concurrently and within reasonable response time.
HTTPS and WSS use built-in encryption methods to send encrypted messages. The question is if it is the best and fastest way to encrypt data. It may be possible to implement an encryption algorithm on your own to increase performance.

5.4 Potential impacts
The responses sent by server contain only text in form of JSON strings. The test results might have looked different if the backend application also handled images and/or documents.

The servlet engine used during this thesis was Tomcat, it was thought that the choice of servlet engine would have very little impact on the test results. No other servlet engine was used to run the backend application. It is uncertain which potential performance gains it could be seen in the results if other servlet engine would have been tested.
It is completely impossible to simulate the real world where requests might come at any rate, from different locations and different users. JMeter simulates concurrent users by creating multiple threads acting as separated users. These threads were run simultaneously on the client computer each making its own requests in a loop. This does not mirror reality, as a bottleneck was experienced at the client computer when the CPU reached its max capacity. It is also hard to simulate completely concurrent users because even if the threads run concurrently they still share the same CPU. The CPU handles every thread in short periods of time switching between them, called context switching.

The reason load test had to be performed at KTH network was because the home network tested had some limitations regarding the amount of connections it was possible to make with websockets. The authors believe the reason of this delimitation have to do with the fact that home networks usually stand behind a NAT-router which provides only limited connections to the outside network.

Response time measured from the home network also showed variation compared to response time measured at KTH network, this variation in time have to do with a higher round trip time to the servers from the home network. This value was very low when measured from KTH network as the servers were located within the same building.

During the tests a noticeable increment in response time was observed when the load was increased but at some points, the increment seemed to cease. This was later discussed with KTH’s IT who explained that the servers run on dynamic resources, meaning that the server could get more CPU capacity allocated if the servers struggled with CPU power.

5.5 Social and economic aspects
The base of this thesis is to analyze various web API protocols, which will be best adapted to the platform, which would be constructed for Linkle. At present, most companies have their own customer portal, which means there are several employees working to develop this. Instead of spending time and money on maintaining this application, you can join Linkle. Linkle will manage the maintenance of the customer portal for the companies. This can help and provide a financial relief. Other companies that do not use a
customer portal can also join Linkle, which will get them a digital tool to communicate to their customers instead of the classic paper outlay. This will lead to less paper outdraw and it is friendlier to the environment.
6. Conclusion

The goal of this thesis was to analyze different forms of web API protocols and find a suitable one to implement on Linkle’s requested web-platform. The goals have been successfully met through a research that was made on common web API protocols, their performance in terms of response time and scalability as well as their security. Documentation of these protocols have been put into this paper. The tests performed on these protocols measured response time, connection time, load capacity and response time on multiple requests. Only after analyzing the test results, a protocol to Linkle's platform could be chosen. The choice is RxREST over HTTPS. RxREST is selected ahead of REST because response time was noticeably faster in those requests that could be run asynchronously and where tasks were unable to be executed asynchronously it performed as good as REST. Another reason that RxREST is selected is because clients do not need constant connection to the server. Websocket is ruled out due to high connection time and, if many users are connected, it affects the server’s performance negatively because websockets connections occupy memory. The fact that the platform does not have high interaction with the server makes it more efficient to use REST instead of websockets. Security is a very important factor needed on the platform, hence the choice of HTTPS.

6.1 Future work

Future work that can be done is to investigate and compare the various web API protocols frameworks. If there is any difference in performance between these then it will be noticeable in the results. Another interesting factor would be to run the tests in an isolated environment because the communication will then not be influenced by other traffic in the network. This would probably lead to more accurate results. If the tests were to be run again in the future, consideration should be given to what was discussed in section 5.4. This may have affected the tests and may thus give different results.
Conclusion
References


Appendix

Table 1: Average response time of register request, with standard deviation.

<table>
<thead>
<tr>
<th></th>
<th>REST</th>
<th>REST HTTPS</th>
<th>RxREST</th>
<th>RxREST HTTPS</th>
<th>WS</th>
<th>WSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No load</td>
<td>27,0635</td>
<td>28,3518</td>
<td>9,878</td>
<td>10,636</td>
<td>9,6865</td>
<td>10,0416</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>6,0909</td>
<td>7,3816</td>
<td>3,2012</td>
<td>5,4188</td>
<td>2,4895</td>
<td>2,6634</td>
</tr>
<tr>
<td>50 active users</td>
<td>32,7273</td>
<td>35,4142</td>
<td>11,019</td>
<td>13,4433</td>
<td>10,8151</td>
<td>12,1313</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>14,9837</td>
<td>14,7898</td>
<td>4,009</td>
<td>8,3752</td>
<td>15,0017</td>
<td>5,2269</td>
</tr>
<tr>
<td>500 active users</td>
<td>37,735</td>
<td>73,768</td>
<td>12,3941</td>
<td>36,0571</td>
<td>11,9287</td>
<td>20,7365</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>25,6378</td>
<td>321,5575</td>
<td>23,921</td>
<td>265,4448</td>
<td>22,6019</td>
<td>29,5856</td>
</tr>
</tbody>
</table>
Table 2: Average response time of login request, with standard deviation.

<table>
<thead>
<tr>
<th></th>
<th>REST</th>
<th>REST HTTPS</th>
<th>RxREST</th>
<th>RxREST HTTPS</th>
<th>WS</th>
<th>WSS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No load</strong></td>
<td>8,9518</td>
<td>9,662</td>
<td>9,0756</td>
<td>9,407</td>
<td>8,6342</td>
<td>8,8725</td>
</tr>
<tr>
<td><strong>Standard deviation</strong></td>
<td>2,211</td>
<td>3,569</td>
<td>2,313</td>
<td>3,277</td>
<td>4,788</td>
<td>1,881</td>
</tr>
<tr>
<td><strong>50 active users</strong></td>
<td>10,9024</td>
<td>11,6738</td>
<td>11,304</td>
<td>12,826</td>
<td>10,9997</td>
<td>11,8935</td>
</tr>
<tr>
<td><strong>Standard deviation</strong></td>
<td>5,3675</td>
<td>5,7215</td>
<td>4,2447</td>
<td>7,0319</td>
<td>5,1893</td>
<td>5,6753</td>
</tr>
<tr>
<td><strong>500 active users</strong></td>
<td>11,9506</td>
<td>26,989</td>
<td>12,895</td>
<td>33,1372</td>
<td>12,5371</td>
<td>19,2489</td>
</tr>
<tr>
<td><strong>Standard deviation</strong></td>
<td>14,0273</td>
<td>147,823</td>
<td>9,2141</td>
<td>200,4488</td>
<td>8,3801</td>
<td>30,35</td>
</tr>
</tbody>
</table>
Table 3: Response time on multiple requests, unsecured API's.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>REST</th>
<th>RxREST</th>
<th>WS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>21</td>
<td>86</td>
</tr>
<tr>
<td>60</td>
<td>613</td>
<td>617</td>
<td>623</td>
</tr>
<tr>
<td>90</td>
<td>923</td>
<td>897</td>
<td>898</td>
</tr>
<tr>
<td>120</td>
<td>1208</td>
<td>1207</td>
<td>1179</td>
</tr>
<tr>
<td>150</td>
<td>1493</td>
<td>1480</td>
<td>1483</td>
</tr>
<tr>
<td>180</td>
<td>1767</td>
<td>1762</td>
<td>1766</td>
</tr>
<tr>
<td>210</td>
<td>2063</td>
<td>2055</td>
<td>2077</td>
</tr>
<tr>
<td>240</td>
<td>2338</td>
<td>2341</td>
<td>2342</td>
</tr>
<tr>
<td>270</td>
<td>2624</td>
<td>2623</td>
<td>2622</td>
</tr>
<tr>
<td>300</td>
<td>2939</td>
<td>2920</td>
<td>2915</td>
</tr>
<tr>
<td>330</td>
<td>3268</td>
<td>3220</td>
<td>3239</td>
</tr>
<tr>
<td>360</td>
<td>3575</td>
<td>3502</td>
<td>3522</td>
</tr>
<tr>
<td>390</td>
<td>3858</td>
<td>3785</td>
<td>3805</td>
</tr>
<tr>
<td>420</td>
<td>4182</td>
<td>4092</td>
<td>4093</td>
</tr>
<tr>
<td>450</td>
<td>4462</td>
<td>4367</td>
<td>4386</td>
</tr>
<tr>
<td>480</td>
<td><strong>4803</strong></td>
<td><strong>4668</strong></td>
<td><strong>4657</strong></td>
</tr>
<tr>
<td>510</td>
<td>5092</td>
<td>4991</td>
<td>4924</td>
</tr>
<tr>
<td>540</td>
<td>5387</td>
<td>5347</td>
<td>5203</td>
</tr>
<tr>
<td>570</td>
<td>5697</td>
<td>5624</td>
<td>5485</td>
</tr>
<tr>
<td>600</td>
<td>6030</td>
<td>5904</td>
<td>5774</td>
</tr>
</tbody>
</table>
### Table 4: Response time on multiple requests, secured API’s.

<table>
<thead>
<tr>
<th></th>
<th>REST HTTPS</th>
<th>RxREST HTTPS</th>
<th>WSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>223</td>
<td>197</td>
<td>260</td>
</tr>
<tr>
<td>20</td>
<td>327</td>
<td>322</td>
<td>364</td>
</tr>
<tr>
<td>30</td>
<td>439</td>
<td>422</td>
<td>461</td>
</tr>
<tr>
<td>40</td>
<td>538</td>
<td>531</td>
<td>569</td>
</tr>
<tr>
<td>50</td>
<td>633</td>
<td>646</td>
<td>674</td>
</tr>
<tr>
<td>60</td>
<td>738</td>
<td>756</td>
<td>770</td>
</tr>
<tr>
<td>70</td>
<td>847</td>
<td>856</td>
<td>872</td>
</tr>
<tr>
<td>80</td>
<td>939</td>
<td>958</td>
<td>979</td>
</tr>
<tr>
<td>90</td>
<td>1043</td>
<td>1078</td>
<td>1075</td>
</tr>
<tr>
<td>100</td>
<td>1135</td>
<td>1181</td>
<td>1176</td>
</tr>
<tr>
<td>110</td>
<td>1275</td>
<td>1291</td>
<td>1282</td>
</tr>
<tr>
<td>120</td>
<td>1370</td>
<td>1385</td>
<td>1371</td>
</tr>
<tr>
<td>130</td>
<td><strong>1469</strong></td>
<td><strong>1480</strong></td>
<td><strong>1459</strong></td>
</tr>
<tr>
<td>140</td>
<td>1596</td>
<td>1579</td>
<td>1561</td>
</tr>
<tr>
<td>150</td>
<td>1702</td>
<td>1682</td>
<td>1656</td>
</tr>
<tr>
<td>160</td>
<td>1793</td>
<td>1782</td>
<td>1778</td>
</tr>
<tr>
<td>170</td>
<td>1891</td>
<td>1887</td>
<td>1888</td>
</tr>
<tr>
<td>180</td>
<td>1995</td>
<td>1994</td>
<td>1984</td>
</tr>
<tr>
<td>190</td>
<td>2108</td>
<td>2091</td>
<td>2082</td>
</tr>
<tr>
<td>200</td>
<td>2208</td>
<td>2196</td>
<td>2171</td>
</tr>
</tbody>
</table>