Richard Hoorn

Google Glass
A Backend support for Google Glass

Degree Project for Master of Science in Engineering, Computer Engineering
June 2015
Datavetenskap

---

Richard Hoorn

Google Glass

Ett backend stödsystem för Google Glass

---

Examensarbete för civilingenjörsexamen i Datateknik

Juni 2015
Google Glass
A Backend support for Google Glass

Richard Hoorn
This report is submitted in partial fulfillment of the requirements for Master of Science in Engineering degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

__________________________________________________________
Richard Hoorn

Approved, 2015-06-05

__________________________________________________________
Advisor: Donald F. Ross

__________________________________________________________
Examiner: Thijs Holleboom
Abstract

This dissertation describes a project to create a prototype application for Google Glass, where the purpose is to help assembly line industries by allowing workers to see instructions visually while working with both hands free. This solves the problem of requiring an instruction manual since instead all instructions will be stored in a database. Google Glass retrieves and displays the information for the user after scanning a QR-code for the product which is going to be assembled. An important aspect is to see if such a system is powerful enough for industries to start working with Google Glass. This concept was developed into a working prototype system, where Google Glass can retrieve data by scanning a QR-code that contains information about a specific product. This information will give a step by step instruction on which components the product contains and the instructions for assembling them. The results presented in this dissertation shows that Google Glass is not suited for the industry at its current state.

Keywords: Google Glass, Web API, SQL, Performance Test
Sammanfattning


Nyckelord: Google Glass, Web API, SQL, Prestandatest
Acknowledgements

I would like to thank Sogeti for the opportunity to work on this project, special thanks to Mats Persson who was my supervisor at the company and Kalle Henriksson who had a code review session which helped refactoring the code. To Johan Häger, thank you for your work on the android application and the project as a whole. Finally I would like to thank Donald F. Ross, who was my supervisor at Karlstad University, for the help on the dissertation.
# Contents

1 Introduction ................................................. 1
   1.1 Goals .................................................. 2
      1.1.1 System .......................................... 2
      1.1.2 Testing .......................................... 2
   1.2 An Alternative Solution .............................. 3
   1.3 Disposition ........................................... 4
   1.4 Divided Project ....................................... 5

2 Background .................................................. 7
   2.1 Motivation .............................................. 8
   2.2 Google Glass .......................................... 8
   2.3 Augmented Reality .................................... 12
   2.4 Web API ............................................... 14
      2.4.1 ASP.NET ......................................... 14
      2.4.2 MVC ............................................. 15
   2.5 Database .............................................. 16
      2.5.1 Microsoft SQL Server ............................ 16
   2.6 Microsoft Azure ...................................... 16
   2.7 Similar Products ..................................... 18
      2.7.1 C-Wear Interactive Glasses .................... 18
   2.8 Chapter Summary ..................................... 19

3 Design ..................................................... 21
   3.1 Google Glass .......................................... 22
   3.2 Database .............................................. 24
   3.3 Measurement ......................................... 26
   3.4 Chapter Summary ..................................... 29
## Implementation

4.1 Google Glass

- 4.1.1 Set up the retrofit connection
- 4.1.2 Interface
- 4.1.3 Downloading the product
- 4.1.4 Measurement Implementation

4.2 Web API

- 4.2.1 API Part

4.3 Interface Part

- 4.3.1 Razor example
- 4.3.2 Web Pages

4.4 Database

4.5 Chapter Summary

## Results

5.1 Test Scenario

- 5.1.1 Link Speed

5.2 Theoretical

5.3 Tests

- 5.3.1 Mobile Phone
- 5.3.2 Google Glass
- 5.3.3 Comparison

5.4 Evaluation

5.5 Chapter Summary

## Conclusions

6.1 Technical Perspective

6.2 Non-Technical Perspective

6.3 Frontend Conclusions
List of Figures

1.1 Overview ................................................................. 1
2.1 System overview ..................................................... 7
2.2 How Google Glass is mounted [1] ............................... 9
2.3 The interface for voice commands [2] ......................... 11
2.4 Immersion taking over the timeline ............................ 12
2.5 GPS using Augmented reality [3] ............................... 13
2.6 Formatted response in both Xml and Json .................... 14
2.7 MVC design pattern [4] ............................................. 15
2.8 Examples of Azure services [5] ................................. 17
2.9 C-Wear Interactive Glasses BM20 [6] ......................... 18
3.1 Information flow ...................................................... 22
3.2 Screen examples ...................................................... 22
3.3 Database diagram ................................................... 24
3.4 Round-trip ............................................................ 26
3.5 Part of an image ....................................................... 27
3.6 Scaling of bytes ....................................................... 28
4.1 UML-diagram for the Product classes ......................... 32
4.2 Product information .................................................. 37
4.3 Code First and DbContext [7] ...................................... 38
4.4 Example of razor .................................................... 41
4.5 Basic layout of the web page ..................................... 42
4.6 Products index page .................................................. 43
4.7 Components index page ............................................. 44
4.8 Instructions index page ............................................. 45
5.1 Graph: Preliminary download values ......................... 49
5.2 Wifi Strength in dBm ............................................... 51
5.3 Graph: Theoretical download values ......................... 52
List of Tables

3.1 Table of measurements ............................................. 28
5.1 Theoretical Download Values: File size vs download time ........... 52
5.2 Mean results for Mobile Phone: File size vs RTT .................... 54
5.3 Mean results for Google Glass: File size vs RTT .................... 55
D.1 Table of measurements for Mobile Phone: File size vs RTT in nano seconds | 80
D.2 Table of measurements for Google Glass: File size vs RTT in nano seconds | 81

Listings

1 createService ............................................................... 33
2 GetProduct ................................................................. 34
3 AsyncTask ................................................................. 35
4 Measurement ............................................................... 36
5 Get ................................................................. 39
6 SQL-Query ............................................................... 40
7 Razor syntax ........................................................... 41
8 Instructions ............................................................ 46
9 T-SQL for the database tables ........................................ 78
1 Introduction

The world is developing in a fast paced manner. Thanks to computers, information is no longer required on physical paper but rather stored as data in a computer. Reading manuals still requires the use of a computer and that can be time consuming. Now there are new technologies such as Google Glass [8] which allow the users to carry a device with them with ease, and see information displayed within their field of vision.

   Google Glass enables information to be presented in a way that does not require interaction using one’s hands. This is the idea of the project, to create a proof of concept system for Google Glass that can inform the user of how to assemble products by looking at instructions using a Google Glass device. Google Glass identifies the product using a QR-code and retrieves the relevant information from the database. The usage of this system is mainly intended for industry.

   In February 2015, Google announced that it was withdrawing the product from the market but had plans for the further development of Google Glass [9]. Nonetheless, it was decided to continue with this evaluation project since Google Glass is representative of a new class of products and the results from this study would still be relevant.

---

**Figure 1.1: Overview**
Figure. 1.1 shows what the system built for Google Glass will be based on. All of the product information will be stored in an information database, where a product is an entity that contains a name, as well as component descriptions and assembly instructions.

This project was specified and supervised by Sogeti [10], the specification is given in appendix. [B] The equipment for the implementation was also provided by Sogeti.

1.1 Goals

The Google Glass evaluation project was divided into two separate sub-projects. The first, the frontend, was to investigate Google Glass as a user interface. The second, the backend, was to create a Web API and database system, and to measure the response time for requests from Google Glass.

1.1.1 System

The first step was to create a functional system with both backend and frontend for Google Glass. In this system the frontend is the application for Google Glass and the backend is a Web API and a database. The goal of this dissertation is to create a backend support system for the assembling application developed for Google Glass.

1.1.2 Testing

The principle focus of this dissertation and the backend project was to measure the response time of the backend to requests product information from the frontend. Is the download time for Google Glass a reasonable time that makes them worth using or does it take too much time?
Apart from the technical aspects, another interesting aspect was to see if Google Glass is comfortable to wear. Does anyone want to wear a Google Glass device for an hour or for a day?

1.2 An Alternative Solution

The idea of using Google Glass in an assembly line industry is because of lightness and wear-ability of the device. In other words, the availability of a portable device and being able to work hands-free. Whether this proves to be a good solution for mobility or not, there are other solutions that would be better in terms of information density. If the requirement is the possibility to assemble a product using both hands with the information on a screen, a fixed screen attached to a computer would be an alternative. This would contain more information than the screen displayed in Google Glass. Operating this station could be done with a foot pedal.

One advantage of using a computer screen over a pair of Google Glass is that a computer screen contains more information but a disadvantage is that it is not very portable.
1.3 Disposition

This section presents a brief overview of the dissertation layout.

Chapter 2 describes the background of the system and its components. There was a requirement from Sogeti that the components of the system had to be compatible with Microsoft products, such as an SQL server and Web API. Further details about these components are shown in Chapter 2.

Chapter 3 describes the design behind the system. An example of this is the database design and the layout of the Web API’s interface. This chapter also includes a discussion about the performance tests.

In Chapter 4, information about the implementation itself is provided. This chapter presents code extracts and discusses the implementation in more detail.

Chapter 5 presents the results of the study together with a discussion and evaluation of the results. These results are presented as text, tables and graphs.

Finally, Chapter 6 concludes the dissertation with an evaluation of the entire project and a summary of the project results.
1.4 Divided Project

It was mentioned earlier that this project consisted of a backend and a frontend part and, that this dissertation implemented the backend part. Figure 1.1 shows a graphical overview of the project. The glasses represent the Google Glass device that requests and receives product information from a database. This dissertation implements and evaluates the flow of information between the backend and frontend of the system. In addition this dissertation also describes a backend support system created for the assembling application developed on Google Glass.

The second part of the project was implemented by Johan Häger [11] where there is a wider focus on Google Glass themselves, or in other words, the frontend. That part of the project contains detailed information of how the product information is displayed as well as a discussion about different ways to present information. His dissertation contains performance tests that covers how efficiently Google Glass can display the information after all data has been downloaded as well as how well the Google Glass camera can read QR-codes. His results will be included with the results from this dissertation in Chapter 6.

This is because both parts of the project results are required in order to answer the question at hand which is if Google Glass is a viable option for this type of system.
2 Background

This project is about creating a proof of concept application for Google Glass, the purpose of this application is to be able to use augmented reality to interact with objects in the real world. Augmented reality is explained in section 2.3 but briefly is about adding virtual information in a user’s field of vision. In this case the object is a product and the purpose itself is to use the augmented reality to assist in manufacturing/assembling tasks. The example used in this dissertation is how to assemble a particular product that Google Glass scans. The information is a list of components and assembly instructions that are required to assemble the product.

Figure 2.1 presents the processes involved in the system. A second requirement in the specification was that the backend must be implemented using Windows compatible platforms.

The layout of this chapter follows the system overview seen in Figure 2.1. The information contained in all the sub-sections is background information on all of the com-
ponents presented in Figure 2.1 starting with Google Glass and augmented reality [12]. There then follows information on Web API [13] and the database. The Web API and database are hosted on a cloud service which is provided by Microsoft Azure [14]. This is marked in the figure by the dashed circle. After the description comes a section describing a similar product to the Google Glass, namely C-Wear Interactive Glasses [6]. Finally a chapter summary is presented.

2.1 Motivation

The reason for doing this project is due to the new ways of thinking that products such as Google Glass provide, in other words, the ability to incorporate augmented reality in daily tasks with the purpose of simplifying them. The tasks themselves are still performed by the user, but he or she does not necessarily need to remember every step of the procedure. What is interesting to test is to see if Google Glass is powerful enough in terms of network speed, memory and computational power to make it a viable option for the sort of application that will be created.

2.2 Google Glass

Google Glass is a device designed for presenting information in the user’s field of vision. This is achieved by a micro-computer attachment on a pair of regular glasses, see Figure 2.2. The information is projected on to a screen in the upper right of the user’s field of vision. The screen resolution is 640 x 360 pixels and the information is presented on the screen in a form that Google have decided to call cards. Google have ensured that the screen feels natural in the field of vision. One example on how the screen appears can be seen in Figure 2.2 (c).

There are two different types of cards, live cards and static cards. Static cards are rarely updated and can contain simple information such as HTML, texts, images and video. Live cards on the other hand are more complex, these cards make use of custom
graphics to deliver real-time information. In addition these cards also access low-level sensors giving them the ability to interact with systems such as the Global Positioning System (GPS). One important aspect to note is that cards are organized in the timeline system.

The meaning of the timeline system is that the cards are organized in sections such as the past, the present and the future. The cards to the right of the home screen represent past event and the cards to the left of the home screen represent future events. The current screen represents the present. The first screen displayed is the home screen.

There are currently two ways to control the timeline system. The first is a touch based system and the second, a voice activated system. The touch system runs from the temple to the ear, as shown by the arrows in Figure 2.2 (a). To interact with the interface you can tap or swipe in three different directions.
Tap

Tapping on the interface activates the glasses from stand-by mode. This will take you to the home screen. Tapping while Glass already is in active mode is used as a select tool, in other words, selecting what the current card shows.

Swipe forwards

Swiping forward takes you to the cards on the right hand side of the home screen, these cards contain information of what happened in the past, such as messages and pictures.

Swipe backwards

Swiping backwards will instead take you to the cards on the left hand side of the home screen, these cards contain information of what will happen in the future. An example of such information is the upcoming weather.

Swipe downwards

Swiping downwards while in an application will switch off the application. If no application is running when swiping down, then the system places Google Glass in stand-by mode, switching off the screen.

The voice system leads to a completely hands-free interaction. When using voice activation, a different menu is presented, in other words there are no cards to swipe right or left. Instead of the swipe menu there is a list of voice commands that are predefined in the system, see Figure.2.3

An example of a voice command is “Google”. This activates the Googling mechanism and now the next step is to name the topic to be Googled. For a complete example from the home screen what you need to do is simply say: “ok glass, google ‘How tall is Mount Everest’” and it will then Google your question.

There are a number of other commands other than those listed in Figure. 2.3. One for example is to take a picture. Just saying the phrase “take a picture” actives the Google
Glass camera and takes a picture, which is then displayed. After the picture has been displayed you can say “ok glass” again to see a menu of options for what may be done with the picture. Available options are for example send and delete.

The touch system and voice system cannot be used simultaneously inside of the timeline system since they require different menus. Within an application either touch or voice may be used, but not both. This means that tapping will not work in the voice menu and saying “ok glass” will not work in the touch menu.
There is a third way of presenting information other than static or live cards. This way is called immersion [15]. Immersions are fully customizable by the developer, in other words, immersions does not need to follow the same restrictions as cards do. What this means in effect is that swiping and tapping do not necessarily change screen and select. Instead it can do what the developer wants, for example, control an object in a game. For example the bubble in Figure 2.4 where, swiping could control its movement. Immersions are necessary if a developer wants to create a custom experience, for example, a game where touching the pad moves a character instead of switching screen.

2.3 Augmented Reality

A definition of Augmented Reality was presented by Azuma in 1997 [12], where he noted that Augmented reality is a variation of virtual reality. Virtual reality is achieved when the entire field of vision is immersed in a virtual world, replacing reality completely. Augmented reality on the other hand does not completely occupy the whole field of vision. Instead, augmented reality adds virtual objects with information in addition to what is already there, hence augmenting the real world. In other words augmented reality is in a way an enhanced reality. Enhanced in the meaning that the additional information provided is relevant to what is already presented in the field of vision. An example of augmented reality can be seen in Figure 2.5 where GPS information is added on the
windshield creating an overlay on top of the actual road to help with navigation [16].

![Figure 2.5: GPS using Augmented reality][3]

This is relevant to this topic since augmented reality can be achieved with head mounted displays [17](HMD) such as Google Glass. Google Glass has a camera which can see what is in front of the user. Scanning the environment enables the Google Glass to display information based on what it records, in other words what the user sees. The camera can be seen in Figure. 2.2 (a).

The question now is, why would we want this? The answer is of course the application areas that this technology supports. An example of such an application area is manufacturing and assembly, assembling a system can be complex and might require instructions. Augmented reality as shown earlier can assist in such tasks. For example, a worker scans a product and receives information about the product’s components together with information on how to assemble the product. This information could be either in textual or graphical form.

Manufacturing and assembly is the main topic of this project but it is definitely not
the only application that has potential. In the article written by Azuma he also covers medicine as a potential application area with the argument that it could be a useful tool when performing surgery, or as training aid for learning.

2.4 Web API

A Web API \[13\] is short for Web application programming interface. There are two types of Web APIs, a client side and a server side. This section will focus only on the server side API. A server side API is defined as a request-response system and the responses are typically formatted usually in XML or JSON. The difference between JSON and XML can be seen in Figure. 2.6. In this example the Web API responds with a list of two strings “value1” and “value2”.

![Figure 2.6: Formatted response in both XML and JSON](image)

The Web API uses the Representational State Transfer \[18\] (REST) software architecture style. REST is a series of guidelines on how to develop maintainable and scalable web service architectures.

2.4.1 ASP.NET

ASP.NET \[19\] is a Web development model created by Microsoft. It is designed to provide developers with all the tools necessary to build high class web applications. To be able to use ASP.NET, a compatible language is required, the most common languages being C# and Visual Basic. These languages allow the developer to benefit from the advantages of the .NET framework such as all of the classes, type safety and inheritance. ASP.NET is used as a tool when building web API and HTTP based services.
2.4.2 MVC

MVC [4] is an architectural design pattern with the purpose of creating a standard structure by separating the application into three components, the model, the view and the controller. Each component has a distinct role:

Model

The model is where the data logic is stored. This is usually a database or connection to a database via for example Entity Framework [20].

View

The views control what the end user sees when using the application. No data manipulation should ever occur here. The view is for presenting results and fetching user input.

Controller

Controllers handle user interaction. They collect the input from the view and subsequently work with the model and ultimately select what the view will display to the end user.

Figure 2.7: MVC design pattern [4]
A pictorial example taken from Microsoft’s web page can be seen in Figure. [2.7] The Web API does not have to be implemented with the help of MVC but it is possible and recommended.

2.5 Database

A database is a system for storing organized information. This is vital to the project because when a product is scanned, the database needs to contain information about that specific product together with its components and assembly instructions.

2.5.1 Microsoft SQL Server

Microsoft SQL Server [21] is a software product developed by Microsoft. It is a database management system. There are many versions of Microsoft SQL Server, in order to support most types of machines with different computational power. The primary query languages that Microsoft SQL Server uses are T-SQL and ANSI SQL.

2.6 Microsoft Azure

Azure [14] is a public cloud service created by Microsoft, the cloud is used to deploy services that are accessible by different end hosts. The scalability also makes it economical, you pay for the exact amount of computational power you need, no more, no less.

There are several applications that may be run on Azure, Figure. [2.8] shows most of these.
Figure 2.8: Examples of Azure services [5]
2.7 Similar Products

There is a company from Sweden that is developing HMDs for industrial tasks. The company’s name is Penny [22]. At their website you can find their product, namely C-Wear Interactive Glasses.

2.7.1 C-Wear Interactive Glasses

C-Wear Interactive Glasses [6] is a product by Penny that can be seen in Figure. 2.9

![C-Wear Interactive Glasses BM20](6)

Figure 2.9: C-Wear Interactive Glasses BM20 [6]

The reason these are comparable with Google Glass is because they already have solutions for industrial systems. One system they present on the web site is assembling and service-instructions, which, is exactly what this project is about, but in this project it will be implemented for Google Glass.
2.8 Chapter Summary

The purpose of this chapter was to give a brief introduction of what this project is about and also give a shorter introduction to the devices and techniques that are used later in Chapter 4 where the implementation itself will be discussed in detail. The most important parts to note from this chapter is what Google Glass is and what the purpose of using them are. Additionally since this project had a requirement that it must use Microsoft compatible solutions means that knowledge about Web API is crucial.
3 Design

This chapter will describe application design including the database design. The chapter will start with the design of the android application for Google Glass and how the information flow will function when the user of Google Glass starts the application. The next section includes information about the database and the Web API design. Finally there is a section where different file sizes are discussed which represent different types of information. In other words, text, pictures, audio and video. The result from this discussion will be used later in Chapter 5 in the performance tests. The reason for presenting the discussion is that the data sizes tested in the result chapter should reflect realistic values.
3.1 Google Glass

The front-end part of this project is the application for Google Glass where all of the information will be displayed for the user. The general layout is shown in Figure. 3.1, where, the rectangles represent separate screens.

![Diagram](image)

Figure 3.1: Information flow

The name and optionally an image of the product itself will appear on the first screen when it is scanned. The subsequent cards will then display the components followed by the assembly instructions. The instructions follow the components since the idea is that the user should collect all the components first before assembling the product.

![Image](image)

(a) Product  
(b) Component  
(c) Instruction

Figure 3.2: Screen examples
Figure 3.2 shows examples of the different cards that the user sees when scrolling throughout the product information.

This information may be displayed in five different ways, text, pictures, text and pictures, pictures and audio, or videos.

Text
In the simplest case the application would show the information in textual form, since text does not take as many bytes as a picture or even video. This means that this option uses the least amount of bytes.

Pictures
In some cases a user might require a picture to show what the end result looks like, or if an assembly instruction is too complex in text form. This option requires a greater number of bytes. The pictures shown in Figure 3.2 is used in the application now. Such pictures would prove useful for both displaying the product and certain steps in the assembly process. Here it shows a particular instruction. The photos in Figure 3.2(a) and (c) range between 70kB and 250kB in size.

Text and Pictures
The most useful way of presenting information is probably with a combination of pictures and text. Text is considerably smaller than pictures so the size will mainly depend on the file size of the pictures.

Pictures and Audio
Audio fits well with pictures to provide information. Short audio files take up almost as much space as a picture. For longer audio clips this option requires significantly more space, however longer audio is not preferred. This is because if the user misses out on some information in the middle of a long clip he or she might have to listen to the whole audio clip again.
Videos

Another possible solution is to display information in the form of videos. Videos require more bytes of storage compared to pictures. An example video, which was 15 seconds in length, was filmed with Google Glass and required six Megabytes. For the same reason as with the audio a longer video is not preferred.

Later in section 3.3 these information aspects will be discussed in greater detail.

3.2 Database

For this project a database was designed, where, the design itself was derived by examining a product and what information about the product is required. This information consists of a list of components and a list of assembly instructions. This resulted in a database where three tables were created, Products, Instructions and Components. A pictorial example of the database is shown in Figure 3.3.

![Database Diagram](image)

Figure 3.3: Database diagram
Products

This is the main entity of the structure and it is here a new product is added to the database. The attributes Name and Description are strings where a user can decide what to call the product and give it a description. The description attribute is not implemented in the application but it is there if a description is needed. Finally there is an optional attribute called Image, which is used if the application should require an image of the complete product. The images are saved as byte arrays.

Components

The Name, Description and Image attributes serve an identical purpose as in the Products entity. This entity contains two additional attributes, Location and ProductId. Location is an attribute that will be used when components are organized in for example warehouses. Then Location is used to tell the user where the component can be found. ProductId on the other hand is a reference to the Id attribute in the Products table. What the ProductId does is that it links the component to a specific product.

Instructions

The Instructions entity contain Instruction, ProductId and Image, where Instruction is an instruction in textual form and Image is an instruction in pictorial form. The attribute ProductId serves the same purpose as in the Components entity.

The reason that the entities have been divided in this way is that both components and instructions need to be dynamic, and such, they cannot be attributes of the Product entity. This is solved by making them into their own entities and linking them to a product via a foreign key. The foreign key in this case is the ProductId that was explained above in the Component entity. This is also represented in Figure 3.3 by the dots and keys at the end of the lines between entities. The two dots means that one or more of that entity is linked to the entity where the key points. In other words called a “one to many” relationship.
3.3 Measurement

The main metric used in this study is the Round Trip Time (RTT). Figure 3.4 shows the part of the system that will be the main focus of the test. In this case the RTT means the time for Google Glass to send a HTTP request to the Web API and receive the information as a Json string. The timer stops when the information is received and parsed, in other words, when the information is ready to be displayed by Google Glass.

![Figure 3.4: Round-trip](image)

The two main factors in these tests are number of bytes and the response time, where the response time will depend on how many bytes that Google Glass must download. The time is the measured unit and number of bytes the controlling factor. Deciding how many bytes that should be sent for each test is important in order to choose the file sizes as close to real usage as possible.

The different options listed in section 3.1 require different numbers of bytes. The examples below give a better indication of the number of bytes for each method of presenting information.
Text

The Json string that is sent across the network is encoded as utf-8. Meaning that each character in the alphabet is 8 bits which, in turn means that each character takes up one byte of space.

Pictures

A picture is composed of a matrix where, each cell represents a pixel and, each pixel has a RGB value ranging between $0 − 255$, see Figure.[3.5] The most standard type of image is a so called 24 bit RGB. That means that each pixel is three bytes long $\frac{24}{8} = 3$.

```
[255,80,72,12,
 234,45,21,19,
 145,75,84,...]
```

Figure 3.5: Part of an image

The resolution of the screen in Google Glass is 640 x 360, hence, a picture that perfectly fit the screen will be $640 \times 360 \times 3 = 691200$ bytes. This size is the maximum size of a picture for the given resolution. A picture will never be of that size since pictures are compressed to avoid unnecessary data.

Video

Videos are difficult to calculate in bytes. After doing research by downloading videos from YouTube [23], it was found that videos with a good quality ranges between approximately 500kB to 1MB.

Audio

Using MP3 [24] files for audio is the easiest way of determining byte sizes. MP3 uses Kbps and a decent quality is 64Kbps when recording speech. An MP3 audio file at 64 Kbps will require 8kB of data per audio length in seconds.
The initial tests will have a logarithmic interval 1kB, 10kB, 100kB and 1MB to establish boundary values and the shape of the RTT curve. In the case that the RTT curve differs greatly between the intervals then, there will be additional intervals for 5kB, 15kB, 50kB and 500kB. In other words the values in between the values above to increase the information that can be derived from looking at curve.

Table 3.1 shows how the testing will be documented. The number of tests per message size will be 30 since, then a 95% confidence interval can be achieved [25].

Table 3.1: Table of measurements

<table>
<thead>
<tr>
<th>#</th>
<th>Text (1kB)</th>
<th>Picture (10kB)</th>
<th>Picture and Audio (100kB)</th>
<th>Video (1MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
</tr>
<tr>
<td>2</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
</tr>
<tr>
<td>3</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
</tr>
<tr>
<td>4</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
<td>RTT</td>
</tr>
</tbody>
</table>

A 1kB file size was chosen to represent small files, where small files contain only text. The largest file size was decided to be 1MB. A 1 MB file size can represent shorter video clips or a collection of audio and images. The intervals in between were set to increase by a factor of ten between these boundary values. A 10kB file size represents text combined with pictures, where pictures make up most of the size. A 100kB file represent the addition of audio to the pictures, where a five second audio clip at the quality of 64 Kbps is about 40kB. This means that picture make up the most of the file size, unless a less detailed image that require a longer audio clip is presented. This is presented visually in Figure 3.6.
3.4 Chapter Summary

This chapter described in some detail how the system was implemented. In addition it presented the general design of the system. A crucial part of this chapter was to discuss in detail how much data different ways of representing information uses. If the information is purely textual then the data size will be as large as the number of characters in the text. With regard to pictures, video and audio one must decide which level of quality is acceptable. It seems unlikely that a representation of a product information will use only one of these exclusively and therefore a combination of the above is required. It was decided that for the smallest information size 1kB was suitable and the largest would be 1MB. These are therefore the boundary values that will be tested with two additional intervals in between.
4 Implementation

This chapter describes the actual implementation. The layout of this chapter is similar to the design chapter since this chapter reflects the design. The different sections in this chapter include the Google Glass implementation also referred to as the android application as well as the Web API and the database.

The Web API and the android application were implemented at the same time because the main contribution of the android application in this dissertation was to adapt an existing application created by Johan Häger [11], to use Retrofit [26] for downloading the product information from the Web API via HTTP GET. The order of implementation was that first the database was created, then the Web API was created which used the database as its data source. After the Web API was fully functional and connected to the database the next step was to connect the backend system to the android application, in other words, the frontend Google Glass user interface.
4.1 Google Glass

The entire application for Google Glass is written in Java. Additionally Google Glass requires that at least Android API 19 is used \[27\]. This is because Glass Development Kit is required, which, was released with Android API 19.

The Android application uses a library called Retrofit \[26\]. Retrofit turns an Android application into a REST \[18\] client. What this means is that it can connect and receive information from a Web API that uses REST (see section 2.4). For this project Retrofit is used to connect to the Web API, receive the Json string and then immediately parse it into existing classes in the Android code.

Setting up is required for Retrofit to work. Firstly, Retrofit must have existing classes that it can parse the information to. The first step will be to create the classes Products, Components and Instructions. They need to be named accordingly with the specific attributes already in the database. If they are named differently Retrofit will not know where to put the data. The class structure of the products is shown in Figure. 4.1. The class Product needs to contain instances of both Instructions and Components. Since they can be one or more they also need to be of List-type. This is because there is no relationship between the classes as is in the database so that the “one-to-many” relationship must be handled with Lists.

![UML-diagram for the Product classes](image)

Figure 4.1: UML-diagram for the Product classes
### 4.1.1 Set up the retrofit connection

After the classes have been created it is time to set up the Retrofit connection. To do this, Retrofit uses an additional library, namely OkHttp [28]. OkHttp sets up the HTTP connection to the Web API, makes sure the connection is maintained and downloads the content. Using Retrofit with OkHttp to connect to the WebAPI can be seen in the code segment below.

```java
public static <S> S createService(Class<S> serviceClass, String baseUrl) {
    RestAdapter.Builder builder = new RestAdapter.Builder()
        .setEndpoint(baseUrl)
        .setLogLevels(RestAdapter.LogLevel.FULL);

    RestAdapter adapter = builder.build();

    return adapter.create(serviceClass);
}
```

The code above shows a class which creates the rest-adapter. The important part is to set the endpoint, where, the endpoint is the URL to the Web API which in this case is `googleglichsexjobb.azurewebsites.net`. The LogLevel.FULL will show useful information for debugging and also informs the programmer how many bytes that are being sent.
4.1.2 Interface

There are additional components required before the data can be retrieved and parsed into the classes. First of all an interface is required. The interface decides if the HTTP-request is a GET or a POST [18]. This application will never update the database but only retrieve data from it, and so, only a GET is needed. This interface is shown in Listing 2 and is called GetService. The GET operates on the endpoint URL googleglassexjobb. azurewebsites.net which is the Web API. Then it requests data specifically from googleglassexjobb.azurewebsites.net/api/Productvalue/{id} where id is a variable that control which value that will be passed to the API-controller. This value comes from the QR-code scan and determines which product that should be downloaded.

Listing 2: GetProduct

```
public interface GetService {
    @GET("/api/Productvalue/{id}")
    Products GetProduct(
        @Path("id") int id
    );
}
```
4.1.3 Downloading the product

To ensure that the application actually uses the Retrofit connection, the createService method must be called. This call cannot be in the main thread (User Interface (UI)-thread) of Android, simply because Android has decided not to allow network operations on the UI thread. The reason for that decision is that the interface should not freeze while the data is being downloaded. This is solved by creating a class that extends the AsyncTask. An AsyncTask will run asynchronously with the main thread.

The code for the AsyncTask is shown in Listing. 3

Listing 3: AsyncTask

```java
private class DownloadProductTask extends
    AsyncTask<Products, Void, Products> {
    Products product = new Products();
    protected Products doInBackground(Products... params) {
        try {
            GetService client =
                ServiceGenerator.createService(GetProduct.class,
                'http://googleglassexjobb.azurewebsites.net');
            product = client.GetProduct(qrid);
            return product;
        }catch(Exception ex){
            return new Products();
        }
    }
}
```

First of all this code connects the REST client to the endpoint http://googleglassexjobb.azurewebsites.net Now what is left is the download itself. When the line product = client.GetProduct(qrid); executes then this means that it calls the API whose address is set in the GetService interface
and which specifies exactly which product to download derived from qrid.

4.1.4 Measurement Implementation

The round trip time measurement itself was made on application level, more specifically in the android application. The measurement experiment is designed so that each size was downloaded 40 times and the final 30 of those used as results. The first ten runs was to warm up the processor to obtain stable results.

```
for (int iteration = 1; iteration <= 40; iteration++) {
    Timer.getInstance().startTimer();
    product = client.GetProduct("ProductId");
    Timer.getInstance().stopTimer();
    Timer.getInstance().logElapsedTime("Run: " + iteration);
}
```

The code in Listing 4 shows the design of the measurement test. Timer is the class that handles the time measurement and getInstance makes sure that only one timer is running at the time.

The method startTimer saves the current time in nanoseconds and stopTimer saves the current time also in nanoseconds. After that the method logElapsedTime is called which takes the stopTime and subtracts that value with the start time and then logs the result of the elapsed time.
4.2 Web API

The main purpose of the Web API is to act as an information link between Google Glass and the database. When the Web API is called it returns the response in Json format. Another purpose is to provide a web interface where the database can be manipulated so that products can be added and altered.

4.2.1 API Part

An example of how the information is passed from the Web API to Google Glass is shown in Figure. 4.2.

```
{"Components": [{"Id":12,"Name":"Component 1","Description":"First Component","Location":"Location 1"},
{"Id":13,"Name":"Component 2","Description":"Second Component","Location":"Location 2"},{"Id":14,"Name":"Component 3","Description":"Third Component","Location":"Location 3"}],
"Instructions": [{"Id":4,"Instruction":"Instruction 1"},
{"Id":5,"Instruction":"Instruction 2"},
{"Id":6,"Instruction":"Instruction 3"}],
"Id":1,"Name":"Product","Description":"Product Description"}
```

Figure 4.2: Product information

As stated in section 2.4 the response can be in Xml or Json. This solution uses Json since it is easier to fit the data into specific objects [29]. Also as shown in section 2.4 figure 2.6 Xml needs to send larger amounts of text for the same data, which means that Json should be more efficient because it creates less data overhead. As an example of this, Xml needs to write `<String>Value</String>`, that is 22 ASCII-characters. Json only needs to write “Value” which is seven ASCII-characters, a difference of 15 characters. The difference in number of characters translates into a difference of 15 bytes of data.

This is what the API responds with on requests, following is a description of how the backend of it works.
Connecting the database

There is a package in Visual Studio that is called Entity Framework which enables .NET to work with relational data. The database for this project is a relational database. Entity Framework has multiple approaches for how the relations may be handled. This solution uses the Code First approach. Code First creates classes in the model for each table in the database and so, three classes are created, Components, Products and Instructions. So the database object is handled like any other object. The classes inherit from DbContext which is the bridge between the classes and the database.

![Figure 4.3: Code First and DbContext](image)

Figure 4.3 shows how the classes created by Code First are connected to the database. The class created that inherits the DbContext is the class that must be used when querying the database.
API Controller

The controller that handles requests of the api is called an Apicontroller. The api in this case is called Productvalue.

Listing 5: Get

```csharp
// GET: api/Productvalue/5

public object Get(int id)
{
    using (var context = new Model1())
    {
        return context.Products.Include(a =>
            a.Components).Include(b => b.Instructions).Where(b => b.Id == id).ToList().FirstOrDefault();
    }
}
```

The code in Listing 5 shows how the api call is made when a client calls the address http://googleglassexjobb.azurewebsites.net/api/Productvalue/{id} where id is a unique identifier that specifies which product information to request from the database. This code uses a construction in C# that is called using. In this case this means that the variable context only exists inside the using scope. It is preferable to code in such a way when handling databases because the database connection closes automatically. Model1 is the name of the database context which in this using block makes context the variable that handles the database calls.

Visual Studio 2008 introduced a new feature called Language-Integrated Query (LINQ) [31]. LINQ replaces SQL-querying because there is no need to write a large string which SQL parses. Line 6 in Listing 5 shows the LINQ expression, where, the purpose is to list all components and instructions that are bound to the same product. The reason to not use ordinary SQL-queries was quite simple, namely that they are longer and more
complicated to write and, C# has already optimized this procedure. Listing 6 shows an example of how the same query would have been written using SQL instead.

Listing 6: SQL-Query

```
SELECT * FROM Products
  INNER JOIN Components
    INNER JOIN Instructions
      on Instructions.ProductId = Components.ProductId
    on Components.ProductId = Products.Id
  where Products.Id = X
```
4.3 Interface Part

To be able to control this system the API also has a webpage where users can manipulate the database in a simple fashion. All of the pages are built with the Razor [32] syntax. Razor makes it possible to embed server code inside of the webpage. In other words it can combine HTML code with C# code in a similar manner that using HTML can be done with aspx. Razor contains the ASP.NET framework and is a powerful tool when creating Web API with ASP.NET.

4.3.1 Razor example

To mark that a block is in C# code the character @ is used. Using Razor simplifies Web APIs when programming with ASP.NET C#.

Listing 7: Razor syntax

1
@{ var myMessage = "Hello World"; }
2
3
<p>The value of myMessage is: @myMessage</p>

The code in Listing 7 is a simple example to describe how Razor is used to combine HTML with C# code. The first row is C# where the string “Hello World” is assigned to the variable myMessage. In the second row a normal HTML tag is used and simply prints the variable. This is preferable when using the MVC model for designing the Web API. Since both the backend logic and the frontend visual can make use of C#, this makes it easier to pass data between the backend and frontend.

Figure 4.4: Example of razor

Figure 4.4 shows the result of the code in Listing 7.
4.3.2 Web Pages

After this short introduction to the Razor syntax that the web page is built on it is time to present the web page itself. It will be divide up so that there is a short description per page.

![Start Page](image1.png) ![Menu](image2.png)

(a) Start Page (b) Menu

Figure 4.5: Basic layout of the web page

Figure 4.5 (a) shows the initial page when the site is loaded. There is no major functionality on this page other than a portal to the other pages and, this is where the Home button leads. Navigating to the other pages, is achieved by clicking the menu bar at the top right which shows a menu of the pages.

The navigation bar can be seen at the top right of Figure 4.5 (a). When that menu button is clicked in will reveal a drop down list where the other pages are located. These pages are Products, Components and Instructions. The view of the list can be seen in Figure 4.5 (b).
A description of web pages role is presented below. In the end of this section there will be a brief instruction on how to successfully build a product. As seen in Figure 4.6 when selecting the Product site the first view that comes up is the index. The index shows a list of the Product table in the database. In this page there are several options for manipulating this specific table in the database. First of all there is Create New which creates a new row in the database and, after which a new product is added to the product list in the index. For every existing item in the product list there are three options, Details, Edit and Delete. Clicking Details shows a new view where additional information about the product is presented. Examples of such information is a description of the product, what id it has in the database and if there is an image of this product. The Edit page allows the user to make changes in the product information such as, changing the name, the description or removing or changing the picture. Lastly the Delete option removes the product from the product list. Deleting the product will however only work if there are no components or instructions associated with the product to be deleted.

<table>
<thead>
<tr>
<th>Name</th>
<th>Image</th>
<th>Edit</th>
<th>Details</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Pirate</td>
<td>Yes</td>
<td>Edit</td>
<td>Details</td>
<td>Delete</td>
</tr>
</tbody>
</table>

Figure 4.6: Products index page
The Components page is almost identical to the Products page the difference being that this page is connected to the Components table in the database.

![Component Index](image)

**Figure 4.7: Components index page**

Figure 4.7 shows the view when the Components page is loaded. This list contains a list of the components which make up the product. The *Create New, Details, Edit and Delete* function in the same way as in the Products list. The detailed information here also contains a column “Location” which is intended for example in warehouses, to describe where that component is located. In this specific example there are four components that are associated with the product named Space Pirate that was shown in Figure 4.6. A component must have a name, location and an associated product. The Image is optional and if a component contains an image then the fourth column, named Image will say “Yes”.

44
Finally there is the Instructions page which contains the assembly instructions. In this list there are at least three items, the corresponding product, the instruction and a picture. An instruction can either be textual, pictorial or a combination of these.

![Instruction Index](image)

**Figure 4.8: Instructions index page**

Figure 4.8 shows the view of the Instructions page when it is loaded. There are three fields for each instruction. The text under the column Instruction is the textual instruction, then in the Image column it says either “Yes” or “No” which indicates if there is an image connected to the instruction or not. This means that if there is a blank value under the instruction column and a “Yes” under Image then there is only an image to represent the given instruction.

To be able to add a Product with components and instructions, it has to be done in a specific order. Since the creation of a new component or instructions requires a binding to a specific product. This means that the product has to exist beforehand. After a product is created then the user can add as many components or instructions to that specific product by pressing *Create New*.
4.4 Database

The database design was presented in section 3.2. See Listing. 9 in Appendix Section C for the SQL code.

Listing 8: Instructions

```sql
CREATE TABLE [dbo].[Instructions] (  
    [Id] INT IDENTITY (1, 1) NOT NULL,  
    [Instruction] NVARCHAR (MAX) NULL,  
    [ProductId] INT NOT NULL,  
    [Image] IMAGE NULL,  
    PRIMARY KEY CLUSTERED ([Id] ASC),  
    CONSTRAINT [FK_Instructions_ToTable] FOREIGN KEY  
    ([ProductId]) REFERENCES [dbo].[Products] ([Id])
);  
```

All three tables were created in similar fashion. SQL needs at least one column to be a key value, and in these tables that key value is called Id and it is unique for each row. The **IDENTITY** flag is used to make sure that the Id is unique. (1,1) means that the Id starts with the value 1 and increments by 1 automatically when new rows are added.

The relationship between the tables is created using a **FOREIGN KEY**, where the attribute ProductId from Instructions and Components is referenced to the attribute Id in Products. For a pictorial presentation please refer back to Figure. 3.3.
4.5 Chapter Summary

This chapter described in detail the work on the implementation. The implementation was divided into three separate stages; the creation of the database; the creation of the Web API and the connectivity from the android application to the Web API. Both the Web API and database were created for this project. The android application was already created by Johan Häger. The android application was extended to make use of Retrofit and make sure that it connected to the Web API and could download information. The main task of the Web API was to query the database and respond to the get request from the android application with a Json string.
5 Results

This chapter contains results and discussions for the performance tests mentioned in the design chapter. In the design, Chapter 3, it was stated that the logarithmic scaling values from 1kB to 1MB would be used for the experiment.

![Preliminary Test](image)

Figure 5.1: Graph: Preliminary download values

Figure 5.1 shows results from the test with the logarithmic scaling values. These values did not provide with enough information since the difference in round trip time between 100kB and 1MB was too large. There will be an addition to the intervals as mentioned in Chapter 3 and these additional values are 5kB, 50kB, and 500kB.

Two tests will be made in this study, the first with Google Glass and the second with an emulated mobile phone. The mobile phone test was used as a reference to evaluate Google Glass’ performance.

The main result shows that Google Glass has considerable limitations in terms of processing power and hardware cooling, this is because Google Glass quickly overheats when processing too much information. It is still a viable system for downloading product information but it cannot download as much information as easily as a mobile phone. When testing using 500kB Google Glass ran slower and slower for each iteration.
5.1 Test Scenario

Having designed and implemented the system, the next stage was the test stage. Before beginning any tests a description of the test scenario is required. A test scenario describes how the tests are performed and in what type of environment. The test environment used in this study is the prototype system. The reason for testing a prototype system for the tests was that the results should correspond to how the devices should work in a production system. Google Glass does not offer any other Internet connectivity interfaces other than Wifi without being paired with a mobile phone via bluetooth [33].

5.1.1 Link Speed

Testing directly to the Internet via a router could cause variations in the bandwidth. This is dependent on factors such as, how many devices that are connected to the router and the signal strength. To solve these problems a new router was set up that operated at a distance of maximum 1 meter away from the Google Glass. The close proximity should solve any potential problems with the signal strength. Additionally only Google Glass was connected to the router in order to minimize the number of devices connected simultaneously. This proved useful and gave a stable bandwidth around 27 Mbit/s both up and down.
Figure 5.2 shows a graph created by a network analyzer called Wifi Analyzer [34]. It shows that the router set up specifically for the Google Glass test, named GOOGLEGLASS, has the strongest signal strength. The signal was that strong because the router was within range of one meter from the device. This was measured from a mobile phone and it was the strongest signal available, which was the purpose of setting up the router.
5.2 Theoretical

As mentioned in section 5.1 the bandwidth for the tests was 27 Mbit/s which is approximately 3,375 Megabytes per second. With this knowledge it is easy to calculate how fast the product might be downloaded optimally if the bandwidth itself were the only limit.

Table 5.1: Theoretical Download Values: File size vs download time

<table>
<thead>
<tr>
<th>File Size</th>
<th>Download Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1kB</td>
<td>0.0003s</td>
</tr>
<tr>
<td>5kB</td>
<td>0.0015s</td>
</tr>
<tr>
<td>10kB</td>
<td>0.003s</td>
</tr>
<tr>
<td>50kB</td>
<td>0.015s</td>
</tr>
<tr>
<td>100kB</td>
<td>0.03s</td>
</tr>
<tr>
<td>500kB</td>
<td>0.148s</td>
</tr>
<tr>
<td>1MB</td>
<td>0.296s</td>
</tr>
</tbody>
</table>

Table 5.1 shows an estimated theoretical time for each file size. All values were derived from the equation:

\[
\text{time} = \frac{\text{bytes}}{\text{bandwidth}}
\]

where bandwidth has the value 3,375B/s. The file bytes sizes are the values in the header of Table 5.1. The value 3,375 is arguably not very precise, however it is based on the bandwidth that was tracked during the experiment and should prove to be a viable value to compare the experimental tests with.

Figure 5.3: Graph: Theoretical download values

These values shown in Figure 5.3 should describe the trend of the graphs that both Google Glass and the emulated phone should follow. The expectation for the experimental
tests is that there should be a larger difference from the theoretical values at larger file sizes rather than the smaller. The reasoning for this is that smaller file sizes require less computational power to process and larger sizes require more computational power to process. The times in Figure 5.3 do not represent the RTT, but represent instead the upper limit of the download time for each file size at the given bandwidth.

5.3 Tests

In the previous section a theoretical calculation was made to see what the results might look like in an optimal world with the given link speed. In this section the devices themselves will be tested and then these test values compared to the theoretical value to see if there are any valid conclusions that can be drawn.

There are two different tests since there are two different test devices. Firstly a test for Google Glass, and then one test with an emulated phone on the computer. The emulated phone has 2048MB of RAM memory and four processors. The specifications of the processor for Google Glass are not known, or at least not published by Google. The reason for including another device other than Google Glass is to see how well Google Glass performs against a device which should be more powerful. This because the theoretical values measure the upper limit of the download time and not round trip time. Devices have to process the data meaning that the actual tests are expecte to be slower since processing power and request time have to be taken in consideration.
5.3.1 Mobile Phone

The first experiment measures the performance for the emulated phone. The testing commenced after making sure that the bandwidth was the same at 27Mbit/s so that the test values may be compared. No problems emerged during the tests and in Table 5.2, the mean result for each file size is noted.

Table 5.2: Mean results for Mobile Phone: File size vs RTT

<table>
<thead>
<tr>
<th>File Size</th>
<th>RTT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1kB</td>
<td>0.07362s</td>
</tr>
<tr>
<td>5kB</td>
<td>0.06729s</td>
</tr>
<tr>
<td>10kB</td>
<td>0.06823s</td>
</tr>
<tr>
<td>50kB</td>
<td>0.08879s</td>
</tr>
<tr>
<td>100kB</td>
<td>0.16154s</td>
</tr>
<tr>
<td>500kB</td>
<td>0.2826</td>
</tr>
<tr>
<td>1MB</td>
<td>0.49419s</td>
</tr>
</tbody>
</table>

The results for this test follow the trend of the theoretical download value graph, the larger values differ more from the theoretical download values than the lower download values. Downloading one Megabyte takes about one half second. Bear in mind that the mobile phone test measures RTT and the theoretical test measures the download time.

Figure 5.4: Graph: Mobile download values

Figure 5.4 shows the graph for the values measured for the mobile phone device.
5.3.2 Google Glass

The second experiment measures the performance for Google Glass. This time however an unexpected effect emerged during the test. It seems that Google Glass overheats quickly. The RTT values in Table 5.3 show that processing file sizes of 500kB takes quite some time and an side-effect of this is the overheating, which, results in considerable degraded performance.

Table 5.3: Mean results for Google Glass: File size vs RTT

<table>
<thead>
<tr>
<th>File Size</th>
<th>RTT (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1kB</td>
<td>0.0783</td>
</tr>
<tr>
<td>5kB</td>
<td>0.0893</td>
</tr>
<tr>
<td>10kB</td>
<td>0.1092</td>
</tr>
<tr>
<td>50kB</td>
<td>0.2695</td>
</tr>
<tr>
<td>100kB</td>
<td>0.6513</td>
</tr>
<tr>
<td>500kB</td>
<td>8.7799</td>
</tr>
<tr>
<td>1MB</td>
<td>13.7197</td>
</tr>
</tbody>
</table>

The trend of these values starts off in a similar way the theoretical values. When the file size is greater than or equal to 500kB the values for the RTT increase dramatically. This is because when Google Glass process too much information they start to overheat. A warning appeared inside Google Glass which said that Google Glass needed to cool down to function properly. Clearly this problem is already known at Google. As mentioned earlier each test was run forty times with the first ten runs being discounted, however, in the case with Google Glass every test began to run slower and slower. For the test at 1MB the test did not even finish.

Figure 5.5: Graph: Google Glass download values
Figure 5.5 shows the graph for the values in Table 5.3. For the lower file size values, 1kB up to 100kB, the time for downloading the product information is reasonable with a value of under one second. After 100kB the curve increases rather quickly which is due to the increased processing requirement for larger data files, which caused Google Glass to overheat.

5.3.3 Comparison

The results from the tests on the idealised mobile phone and Google Glass are compared in this section. It is this comparison which will determine the viability of Google Glass. It has already been shown that Google Glass has some problems with downloading large amounts of information continuously because of problems with overheating. Those problems did not exist when downloading smaller amounts of data. Therefore a presentation of the results for data sizes less than 500kB for both devices has been used.

![Figure 5.6: Graph: Comparison between Google Glass and Mobile](image)

In Figure 5.6 a graph of both the test run with Google Glass and the test run with an emulated mobile phone is shown. As seen at the lower data sizes both devices behave in a similar manner, Google Glass however generally performs less well throughout the whole test. Even if Google have not published the technical specifications for Google Glass would not be expected to perform as well as a mobile phone. This can clearly be seen in the results, at the smaller file sizes even if they behave in a similar manner Google Glass still performs less well than the mobile phone. Then there is the overheating problem with
Google Glass which may be considered to be a major disadvantage with this device.

### 5.4 Evaluation

This chapter has so far presented graphs for theoretical values and then for two different devices, Google Glass and an emulated mobile phone. The interesting part is the evaluation from this, how does Google Glass actually perform against a mobile phone and is it good enough?

Firstly by inspecting Figure 5.6, it shows that Google Glass’ performance is close to that of the mobile phone. However this only applies for small file sizes and as is shown in Figure 5.5 Google Glass handle file sizes up to 100kB well. When the file sizes are larger than 100kB Google Glass cannot process the information as fast as before, which is caused by the slow processor that starts having difficulties processing the file size. A result of this is that the device is overheating, reducing the performance and taking even longer time to download additional files. Even without the overheating the download time was about six seconds which is almost 12 times slower than the mobile phone for the same size which was 1MB. This is shown in Table D.2.

The results from the theoretical values and the mobile phone result show that the back-end part of the system, in other words, the Web API and the database perform reasonably well. This is shown by the results in Figure 5.6.
Figure 5.7: Graph: Comparison between Theoretical download values and Mobile download values

Figure 5.7 shows a graph where both the values from the mobile test and the values from the theoretical calculation. This indicates that the backend system is not far from the theoretical values when used with a mobile phone. The difference between the curves increases with the increase in file size, which was expected since more data requires longer download. This is where Google Glass performs less efficiently or not at all, since there is not enough computational power.

To summarise what just have been said, Google Glass can download smaller file sizes at almost the same speed as a mobile phone but it cannot download large sizes. Therefore this system might not be optimal for Google Glass since here, all data is downloaded at once. A better fit solution would be a version where the data is divided up and downloaded on the fly. In other words a solution where the data is downloaded when it is needed and not all at once. In that way the users will not have to wait for unnecessary lengths of time. Waiting for six seconds to retrieve one Megabyte of data is not optimal. This problem has already been noted and there will be additional information about this problem in the future work section, section 6.5.
5.5 Chapter Summary

This chapter contained all the results from the performance tests that were designed to measure Google Glass’ capabilities when it comes to downloading information using different file sizes measured in bytes. The results were definitely interesting, it showed that Google Glass has potential when it deals with smaller product information downloads but the performance tends to drop off rapidly at larger sizes. The results were shown with the help of text, tables and graphs in an attempt to relay as much information as possible in different ways.
6 Conclusions

In the previous chapter an evaluation of the implementation was presented. In this chapter the focus is on an evaluation of the project as a whole. In this chapter the questions from the introduction will be answered. As a reminder, these questions were:

“Is the download time for Google Glass a reasonable time that makes them worth using or does it take too much time?” Where the term reasonable time refers to the time it takes to download and process information about a product.

“ Apart from the technical aspects, another interesting aspect was to see if Google Glass is comfortable to wear. Does anyone want wear a Google Glass device for an hour or for a day?” This question is raised because Google Glass is a new type of device and many do not know how it feels to wear them.

There will be a technical evaluation of Google Glass where it is interesting to see how well they perform in the prototype system created in this dissertation. There will also be a non-technical evaluation of Google Glass since Google Glass has personally been used throughout the testing of the system.

6.1 Technical Perspective

From a technical perspective, the performance of Google Glass is not impressive. As shown in Chapter 5, Google Glass do not have enough computational power that would be required for industrial purpose. The results also show that a mobile phone can be used for the same purpose and would be much better.

There is however possibly a solution to the limited computational power, by using bluetooth. Google Glass was not actually supposed to operate as a device on its own.
Google’s idea was that Google Glass would be paired with your mobile phone. By doing this Google Glass can utilize the power from a phone and then simply display the results. This does however not affect the problem with the screen size so again it might just be easier using a phone or a tablet.

6.2 Non-Technical Perspective

In the beginning of the project Google Glass was tested personally out of curiosity to see how the device worked and if it was comfortable to wear. There have been reports about problems with people adopting Google Glass. Jack Smith IV writing for The Observer [35] reported that there were complaints of headache after prolonged usage of Google Glass. From the personal experience of wearing Google Glass it is safe to say that the problems are not misplaced. Having a screen at the top right corner of your eye is somewhat irritating and it is rather small. It is good in the sense that the user can easily look away from the screen as it is not centered in middle of the field of vision. This means that using Google Glass takes some time getting used to.

Another aspect of wearing Google Glass is the problem with overheating that was mentioned in Chapter [5]. It was not clear just how hot Google Glass actually might become. This was after downloading information of a product at approximately 1MB, and so, it is difficult recommending Google Glass for prolonged use. Google Glass became noticeably warm after five minutes of usage.
6.3 Frontend Conclusions

This section summarises Häger’s conclusions from the frontend part of the project. His conclusions will be taken into account before presenting the final conclusion for this dissertation.

6.3.1 Information Density

As mentioned in the technical conclusions of this report, Google Glass has a limited screen size. As it turns out the amount of information that can be displayed on the screen for Google Glass is roughly one quarter of the information that can be displayed on an average mobile phone [11].

6.3.2 Performance

It is already clear in this dissertation that Google Glass cannot download information as quickly as a mobile phone, and definitely not as much information. Google Glass is also slower in scanning the QR-code and slower at displaying the information once it all has been downloaded. The camera for Google Glass could not read QR-codes that were of greater complexity [11] (50 and 100 characters encoded in the QR-Code). The next version needs an upgraded camera.

6.3.3 Restrictive Design

When it comes to designing applications for Google Glass there are restrictions on what the design may look like. This is because of the small screen. Google Glass uses voice command features, however, Google requires that the developers’ voice commands must be approved by Google before release of the application on MyGlass. MyGlass is Google’s own application store.
6.4 Final Conclusion

To answer the main question, namely is Google Glass suited for industry based work? The conclusion drawn from this dissertation is, no. Google Glass does not yet possess enough power to be used in an industrial context. At the moment Google Glass is more of use to a private user since, the prototype worked, but it could not handle significant amounts of data that industrial applications might have as a requirement. As described in section [6.5] there are already ideas on how this prototype may be improved. However even with a file size of 1 MB Google Glass failed to download and process the information in a reasonable time. This means that the information that is going to be downloaded must be checked beforehand to see whether Google Glass is capable or not of handling the actual file size. This is most probably too restrictive for most industrial applications and therefore Google Glass would not be recommended.

The application for Google Glass, which was the frontend, showed that this type of cloud back end system is a viable option for Google Glass. The downloading of information was a bit too much Google Glass to process by itself. A solution to that problem could be to have the relevant product information already stored on Google Glass.

Bear in mind that this system was a prototype system that was developed to evaluate Google Glass which itself also is a prototype device. Computers embedded in devices such as Google Glass have a certain potential even if the current Google Glass prototype not yet is of an acceptable quality for industrial use. There are already other companies which have developed industrial applications for embedded computers in similar devices. For example, Penny [22] that was mentioned in Chapter 2 have developed an application for their glasses which also involve product assembly instructions.
6.5 Future Work

The current status of the project is a working prototype. There are some ideas on how to further develop this prototype, for example, an idea on how to reduce the amount of information downloaded at the same time by downloading information on the fly. This idea will hopefully also reduce overheating of the device. Currently there are two ideas of how to improve the prototype and these are presented below.

6.5.1 Download Images

In the current state of the prototype all product information is downloaded at once. This was not optimal for Google Glass when downloading larger files (>500KB). Instead of downloading all the pictures from the database at once there is a way of sending links to the pictures in the database and download them on the fly in the application when the picture is needed. This solution does not support usage without Wifi while, the current solution does. One advantage of a system that downloads the entire product at once is the possibility of downloading a product in advance in case the assembling area does not have Wifi. In the case that response time is more critical, it would be better to download on the fly rather than to download all the product information at once.

As seen in the results it is not optimal to download files larger than 1MB and, therefore it might be better to have a “Download pictures” on the fly system instead.

6.5.2 Picture Conversion

Currently the pictures in the database are saved as byte arrays. In other words identical to Figure. 3.5. By the time the byte array was processed at the android application it could present it on the screen immediately.

Normally when Json serializes a picture it transforms it into a base64 string, reducing the amount of information sent. This was a problem at the android application side, where
the Android system could not allocate enough memory to translate the base64 string back into a byte array. The Json string had to be transformed to a byte array in order to be able to display the bitmap image. This was solved by changing the Json serialization code in the Web API so that the byte array was unchanged.

6.6 Concluding Remarks

Google attempted to develop a new device, namely Google Glass. If it was a good attempt or bad attempt is actually hard to say. All of the results throughout the dissertation point to that it was a bad attempt, but bear in mind that developing technology means that someone has to carry the torch. This version of Google Glass might not have lived up to expectations, but hopefully other will be inspired to continue working on similar projects.

The implementation of an assembly system for devices such as Google Glass creates opportunities to work without being stationary. The current version of Google Glass is seemingly not intended for a longer use, and thus, is not suitable for industrial purposes. However a private user might be more suited for using Google Glass since they may be using Google Glass for less demanding tasks such as taking a picture or Googling for information. It is uncertain if Google Glass ever will be suited for industrial work since it seems right now that Google Glass’ purpose is for the private users. But in due time Google Glass might be powerful enough to expand their use and become a tool for industry. For now all that can be done is wait and see what device Google will come up with next.
References


[10] www.sogeti.se


A  Abbreviations

HMD  Head Mounted Display

MVC  Model View Controller

API  Application Programming Interface

GPS  Global Positioning System

AR  Augmented Reality

VR  Virtual Reality

HTML  Hyper Text Markup Language

ASCII  American Standard Code for Information Interchange

HTTP  Hypertext Transfer Protocol

RTT  Round Trip Time

B  Specification (In Swedish)
Uppdragsbeskrivning

Google Glass

Version 1.0

Mats Persson

Distributionslista

<table>
<thead>
<tr>
<th>Befattning</th>
<th>Bolag/enhet</th>
<th>Namn</th>
<th>Åtgärd</th>
<th>Info.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>KaU</td>
<td>Richard Hoorn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student</td>
<td>KaU</td>
<td>Johan Häger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Konsult/handledar</td>
<td>Sogeti</td>
<td>Mats Persson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Konsultchef</td>
<td>Sogeti</td>
<td>Åsa Maspers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Säljare</td>
<td>Sogeti</td>
<td>Bengt Löwenhamn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Innehållsförteckning

1. Allmän beskrivning av uppdraget ........................................................................... 3
   1.1 Bakgrund........................................................................................................... 3

2. Google Glass .......................................................................................................... 3
   2.1 Case.................................................................................................................... 4
   2.2 Optioner ........................................................................................................... 4
      2.2.1 Option 1 – Jämföra Google Glass med andra liknande produkter ........ 4
      2.2.2 Option 2 – Bildigenkänning .................................................................... 4
      2.2.3 Option 3 – Känna av position ................................................................. 4
      2.2.4 Option 4 - Röststyrning .......................................................................... 4

3. Genomförande/arbetssätt ..................................................................................... 4
   3.1 Rutiner ............................................................................................................... 4
   3.2 Genomförande ................................................................................................... 5

4. Stöd/kvalitetssäkring ........................................................................................... 5
   4.1 Granskningar .................................................................................................... 5
   4.2 Testarbete ......................................................................................................... 5

5. Leveranser ............................................................................................................. 5
   5.1 Dokumentation .................................................................................................. 5

6. Konfigurationsstyrning ......................................................................................... 5

7. Miljö ....................................................................................................................... 5

8. Uppföljning och Rapportering .............................................................................. 6
   8.1 Rapportering internt/externt ........................................................................... 6
      8.1.1 Statusrapportering .................................................................................... 6
      8.1.2 Möten ......................................................................................................... 6
      8.1.3 Slutrapportering ....................................................................................... 6

Ändringsförteckning

<table>
<thead>
<tr>
<th>Version</th>
<th>Datum</th>
<th>Ändring</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2014-10-15</td>
<td>Dokumentet skapats</td>
</tr>
</tbody>
</table>

Uppdragsbeskrivning

Google Glass
1. Allmän beskrivning av uppdraget

1.1 Bakgrund

Sogeti Sverige AB (Sogeti) är ett IT-konsultbolag med bred verksamhet, stort fokus på kompetens och modern teknik.

Sogeti jobbar mycket med applikationer inom mobilitet och en av de nyaste teknikerna här är Google Glass.

Google Glass introducerar ett helt nytt sätt att tänka på angående hur man interagerar med applikationer och helt nya användningsområden för mobila applikationer.

2. Google Glass

Syftet med detta uppdrag är att tillverka en Proof of Concept på en applikation till Google Glass, applikationen ska kunna hämta/spara data från en web service. Applikationen ska hjälpa användare i en produktion genom att bidra med vilka delar som behövs för att kunna montera något och även en enklare instruktion över hur det ska se ut när det är klart.

Ett annat viktigt syfte med detta uppdrag är att ta reda på hur Google Glass är ur ett användarperspektiv, är det behagligt att använda en hel dag? Osv.

Även prestandamässigt är det intressant för att veta om Google Glass är nog stabila för att kunna rekommenderas till kunder på företag inom tillverkningsindustrin. Och slutligen så är bra/dåliga sidor med denna nya teknik intressant. Finns det begränsningar? T.ex. hur länge räcker batteri m.m.

I uppdraget ingår en back-end och en front-end och nedan ser vi lite exempel på saker man kan fokusera på inom de olika delarna.

I back-end finns följande att fokusera på:

- Prestanda, hur kan man optimera en back-end som pratar med liknande appar för att skicka så lite data som möjligt så snabbt som möjligt?
- Eftersom vi jobbar med Microsofts plattform så är det intressant att veta om man ser tydliga plus/minus med att hosta tjänsten i Azure istället för att man t.ex. hostar den själv på sitt egna nätverk.

Vi ser helst att back-end utvecklas med WebAPI eller WCF då Sogeti har stort fokus på Microsofts plattform.

I front-end som ska köras på Google glass ser vi följande man kan fokusera på:

- Användarvänlighet. Som när touchenheter kom så är detta också ett nytt unikt sätt att interagera på, hur görs detta på bästa sätt? Vad skiljer sig från applikationer till telefoner?
- Multitasking, hur mycket kan göra samtidigt och hur hanteras detta?
- Prestanda
2.1 Case

Följande case ska uppdragstagarna utgå ifrån.

Ett företag har en produktionslinje med ett antal stationer där man monterar ihop saker vid varje station. Delarna man använder för monteringen har en kod på sig som man kan scanna av. Problemet företaget har är att det är många variationer av saker som monteras på varje station och arbetarna byter ofta platser så de har svårt att hålla reda på vad som behövs för att montera och hur det ska monteras.

2.2. Optioner

Följande är förslag på vidareutveckling av detta som uppdragstagarna själv får plocka från om tid finns.

2.2.1 Option 1 – Jämföra Google Glass med andra liknande produkter

Jämföra Google Glass med en annan liknande produkt genom att välja en mindre del av applikationen som man implementerar. Ett förslag på annan produkt är företaget Pennys motsvarighet till Google Glass.

2.2.2 Option 2 – Bildigenkänning

För att slippa att ha koder på varje produkt så skulle det vara bra om applikationen kan känna igen produkterna genom att man bara tittar på produkterna, detta dels för att se vilka möjligheter det finns för bildigenkänning på Google Glass och för att ta reda på vad som redan finns färdigt inom detta område och vad som krävs av företaget för att få bildigenkänning att fungera.

2.2.3 Option 3 – Känna av position

För att underlätta för arbetarna så skulle det vara bra om applikationen kunde känna igen vart i produktionslinjen man sitter och jobbar och då filtrera listan av produkter man har att välja mellan så det blir lättare att hitta produkter.

2.2.4 Option 4 - Röststyrning

Kunna styra applikationen med rösten och ta reda på vad det finns för begränsningar för detta. T.ex. hur tyst måste det vara runt användaren för att det ska fungera?

3. Genomförande/arbetssätt

3.1 Rutiner

Sogeti tillhandahåller arbetsplatser, datorer, hårdvara samt erforderliga utvecklingsverktyg.

Uppdragstagarna kommer att ha access till Sogetis nätverk och förväntas nyttja vår TFS-server för versionshantering.
3.2 Genomförande

Uppdragstagarna planerar själv genomförandet och Sogeti tillhandahåller stödning både projektstyrningsmässigt och rent implementationstechniskt. Sogeti tillhandahåller all programvara och hårdvara som behövs.

Förslagsvis används SCRUM med en sprintlängd på 2-3 veckor som sätts upp där updragstagarna specificerar vad de tror att de hinner med i början av varje sprint och har en demo för en eller flera på Sogeti i slutet på varje sprint.

4. Stöd/kvalitetssäkring

4.1 Granskningar

Vid behov genomförs granskning som kan initieras av både handledare och updragstagare.

Lämpligen definieras några granskningspunkter vid planeringen av projektet.

4.2 Testarbete

Funktions-, system- och integrationstest görs av updragstagarna.

5. Leveranser

5.1 Dokumentation

Systemdokumentation görs av updragstagarna. Dokumenten lagras i projektarkiv hos Sogeti.

Lämpligen levereras en enkel användarinstruktion.

6. Konfigurationsstyrning

All programkod och tillhörande specifikationer och andra utvecklingsdokument ska versionshanteras med hjälp av Microsoft TFS.

7. Miljö

Utvecklingsverktyg väljs av updragstagarna tillsammans med handledare. Hårdvara som behövs för projekttet tillhandahålls av Sogeti.
8. Uppföljning och Rapportering

8.1 Rapportering internt/externt

8.1.1 Statusrapportering
Rapportering av status och framskridande i utvecklingen beslutas i samråd vid projektuppstart.

8.1.2 Möten
Möten hålls vid behov. Vid uppstart läggs lämpligt antal avstämningsmöten in i projektplanen.

8.1.3 Slutrapportering
Arbetet presenteras för Sogeti i samband med lämpligt månads- och alternativt lunchmöte.
C  Database SQL

Listing 9: T-SQL for the database tables

```sql
CREATE TABLE [dbo].[Products] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (MAX) NOT NULL,
    [Description] NVARCHAR (MAX) NOT NULL,
    [Image] IMAGE NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
));

CREATE TABLE [dbo].[Components] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (MAX) NOT NULL,
    [Description] NVARCHAR (MAX) NOT NULL,
    [Location] NVARCHAR (MAX) NOT NULL,
    [ProductId] INT NOT NULL,
    [Image] IMAGE NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_Components_ToTable] FOREIGN KEY ([ProductId]) REFERENCES [dbo].[Products] ([Id])
));

CREATE TABLE [dbo].[Instructions] (
    [Id] INT IDENTITY (1, 1) NOT NULL,
    [Instruction] NVARCHAR (MAX) NULL,
    [ProductId] INT NOT NULL,
    [Image] IMAGE NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC),
    CONSTRAINT [FK_Instructions_ToTable] FOREIGN KEY ([ProductId]) REFERENCES [dbo].[Products] ([Id])
));
```
D Result tables
Table D.1: Table of measurements for Mobile Phone: File size vs RTT in nano seconds

<table>
<thead>
<tr>
<th>#</th>
<th>1kB</th>
<th>5kB</th>
<th>10kB</th>
<th>50kB</th>
<th>100kB</th>
<th>500kB</th>
<th>1MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.81E+07</td>
<td>6.26E+07</td>
<td>6.51E+07</td>
<td>9.05E+07</td>
<td>1.09E+08</td>
<td>4.93E+08</td>
<td>5.35E+08</td>
</tr>
<tr>
<td>2</td>
<td>6.77E+07</td>
<td>6.31E+07</td>
<td>6.51E+07</td>
<td>7.39E+07</td>
<td>1.13E+08</td>
<td>3.11E+08</td>
<td>4.61E+08</td>
</tr>
<tr>
<td>3</td>
<td>5.63E+07</td>
<td>6.71E+07</td>
<td>6.85E+07</td>
<td>9.54E+07</td>
<td>1.22E+08</td>
<td>3.15E+08</td>
<td>4.79E+08</td>
</tr>
<tr>
<td>4</td>
<td>5.67E+07</td>
<td>7.21E+07</td>
<td>7.80E+07</td>
<td>8.95E+07</td>
<td>2.37E+08</td>
<td>2.60E+08</td>
<td>7.18E+08</td>
</tr>
<tr>
<td>5</td>
<td>7.71E+07</td>
<td>6.28E+07</td>
<td>6.82E+07</td>
<td>7.39E+07</td>
<td>1.24E+08</td>
<td>3.28E+08</td>
<td>4.63E+08</td>
</tr>
<tr>
<td>6</td>
<td>7.99E+07</td>
<td>6.18E+07</td>
<td>6.96E+07</td>
<td>7.83E+07</td>
<td>1.18E+08</td>
<td>2.67E+08</td>
<td>5.41E+08</td>
</tr>
<tr>
<td>7</td>
<td>6.67E+07</td>
<td>6.54E+07</td>
<td>6.49E+07</td>
<td>9.95E+07</td>
<td>1.14E+08</td>
<td>2.73E+08</td>
<td>5.03E+08</td>
</tr>
<tr>
<td>8</td>
<td>6.91E+07</td>
<td>6.10E+07</td>
<td>7.20E+07</td>
<td>1.25E+08</td>
<td>1.18E+08</td>
<td>2.31E+08</td>
<td>4.88E+08</td>
</tr>
<tr>
<td>9</td>
<td>6.73E+07</td>
<td>6.37E+07</td>
<td>6.22E+07</td>
<td>1.24E+08</td>
<td>1.62E+08</td>
<td>2.54E+08</td>
<td>4.77E+08</td>
</tr>
<tr>
<td>10</td>
<td>7.38E+07</td>
<td>6.50E+07</td>
<td>6.62E+07</td>
<td>1.18E+08</td>
<td>1.14E+08</td>
<td>2.52E+08</td>
<td>4.70E+08</td>
</tr>
<tr>
<td>11</td>
<td>7.35E+07</td>
<td>6.39E+07</td>
<td>6.51E+07</td>
<td>1.27E+08</td>
<td>1.24E+08</td>
<td>2.66E+08</td>
<td>4.57E+08</td>
</tr>
<tr>
<td>12</td>
<td>1.21E+08</td>
<td>6.36E+07</td>
<td>7.56E+07</td>
<td>7.27E+07</td>
<td>1.15E+08</td>
<td>2.42E+08</td>
<td>4.76E+08</td>
</tr>
<tr>
<td>13</td>
<td>1.02E+08</td>
<td>6.49E+07</td>
<td>6.45E+07</td>
<td>7.29E+07</td>
<td>1.20E+08</td>
<td>2.90E+08</td>
<td>4.66E+08</td>
</tr>
<tr>
<td>14</td>
<td>1.24E+08</td>
<td>6.19E+07</td>
<td>7.01E+07</td>
<td>7.09E+07</td>
<td>1.06E+08</td>
<td>2.72E+08</td>
<td>4.99E+08</td>
</tr>
<tr>
<td>15</td>
<td>6.82E+07</td>
<td>7.02E+07</td>
<td>6.52E+07</td>
<td>9.33E+07</td>
<td>1.08E+08</td>
<td>2.52E+08</td>
<td>4.86E+08</td>
</tr>
<tr>
<td>16</td>
<td>5.99E+07</td>
<td>7.44E+07</td>
<td>6.95E+07</td>
<td>8.77E+07</td>
<td>3.47E+08</td>
<td>2.66E+08</td>
<td>4.82E+08</td>
</tr>
<tr>
<td>17</td>
<td>6.12E+07</td>
<td>6.57E+07</td>
<td>7.06E+07</td>
<td>7.86E+07</td>
<td>2.20E+08</td>
<td>2.59E+08</td>
<td>5.14E+08</td>
</tr>
<tr>
<td>18</td>
<td>9.93E+07</td>
<td>6.50E+07</td>
<td>6.25E+07</td>
<td>7.96E+07</td>
<td>1.69E+08</td>
<td>2.71E+08</td>
<td>4.84E+08</td>
</tr>
<tr>
<td>19</td>
<td>1.16E+08</td>
<td>6.26E+07</td>
<td>8.31E+07</td>
<td>8.79E+07</td>
<td>1.80E+08</td>
<td>2.82E+08</td>
<td>4.99E+08</td>
</tr>
<tr>
<td>20</td>
<td>6.99E+07</td>
<td>6.40E+07</td>
<td>7.58E+07</td>
<td>7.67E+07</td>
<td>2.25E+08</td>
<td>2.57E+08</td>
<td>4.88E+08</td>
</tr>
<tr>
<td>21</td>
<td>6.04E+07</td>
<td>6.34E+07</td>
<td>6.61E+07</td>
<td>9.28E+07</td>
<td>3.59E+08</td>
<td>2.66E+08</td>
<td>4.57E+08</td>
</tr>
<tr>
<td>22</td>
<td>7.17E+07</td>
<td>7.40E+07</td>
<td>6.46E+07</td>
<td>9.08E+07</td>
<td>1.15E+08</td>
<td>2.67E+08</td>
<td>4.97E+08</td>
</tr>
<tr>
<td>23</td>
<td>7.34E+07</td>
<td>6.45E+07</td>
<td>6.51E+07</td>
<td>9.83E+07</td>
<td>1.11E+08</td>
<td>2.35E+08</td>
<td>4.86E+08</td>
</tr>
<tr>
<td>24</td>
<td>7.33E+07</td>
<td>6.10E+07</td>
<td>7.12E+07</td>
<td>8.74E+07</td>
<td>1.03E+08</td>
<td>2.72E+08</td>
<td>4.57E+08</td>
</tr>
<tr>
<td>25</td>
<td>5.62E+07</td>
<td>6.18E+07</td>
<td>6.98E+07</td>
<td>9.86E+07</td>
<td>8.67E+07</td>
<td>2.93E+08</td>
<td>5.04E+08</td>
</tr>
<tr>
<td>26</td>
<td>5.96E+07</td>
<td>6.02E+07</td>
<td>6.21E+07</td>
<td>7.19E+07</td>
<td>1.47E+08</td>
<td>2.82E+08</td>
<td>4.75E+08</td>
</tr>
<tr>
<td>27</td>
<td>6.16E+07</td>
<td>6.84E+07</td>
<td>6.22E+07</td>
<td>7.54E+07</td>
<td>3.62E+08</td>
<td>2.70E+08</td>
<td>4.75E+08</td>
</tr>
<tr>
<td>28</td>
<td>6.45E+07</td>
<td>9.09E+07</td>
<td>7.46E+07</td>
<td>7.34E+07</td>
<td>3.01E+08</td>
<td>3.99E+08</td>
<td>5.06E+08</td>
</tr>
<tr>
<td>29</td>
<td>6.02E+07</td>
<td>8.97E+07</td>
<td>6.61E+07</td>
<td>7.70E+07</td>
<td>1.09E+08</td>
<td>3.18E+08</td>
<td>4.70E+08</td>
</tr>
<tr>
<td>30</td>
<td>6.05E+07</td>
<td>8.39E+07</td>
<td>6.34E+07</td>
<td>8.17E+07</td>
<td>1.08E+08</td>
<td>2.34E+08</td>
<td>5.15E+08</td>
</tr>
</tbody>
</table>
Table D.2: Table of measurements for Google Glass: File size vs RTT in nano seconds

<table>
<thead>
<tr>
<th>#</th>
<th>1kB</th>
<th>5kB</th>
<th>10kB</th>
<th>50kB</th>
<th>100kB</th>
<th>500kB</th>
<th>1MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8,52E+07</td>
<td>7,38E+07</td>
<td>8,29E+07</td>
<td>2,56E+08</td>
<td>3,30E+08</td>
<td>2,52E+09</td>
<td>6,52E+09</td>
</tr>
<tr>
<td>2</td>
<td>1,00E+08</td>
<td>8,67E+07</td>
<td>1,30E+08</td>
<td>2,95E+08</td>
<td>3,09E+08</td>
<td>7,18E+09</td>
<td>6,96E+09</td>
</tr>
<tr>
<td>3</td>
<td>7,24E+07</td>
<td>8,31E+07</td>
<td>8,33E+07</td>
<td>2,51E+08</td>
<td>3,13E+08</td>
<td>1,68E+10</td>
<td>7,08E+09</td>
</tr>
<tr>
<td>4</td>
<td>6,39E+07</td>
<td>1,16E+08</td>
<td>8,52E+07</td>
<td>1,81E+08</td>
<td>3,73E+08</td>
<td>1,90E+10</td>
<td>6,44E+09</td>
</tr>
<tr>
<td>5</td>
<td>7,68E+07</td>
<td>7,70E+07</td>
<td>1,30E+08</td>
<td>1,98E+08</td>
<td>2,62E+08</td>
<td>5,45E+09</td>
<td>7,43E+09</td>
</tr>
<tr>
<td>6</td>
<td>1,12E+08</td>
<td>7,84E+07</td>
<td>8,83E+07</td>
<td>2,02E+08</td>
<td>3,84E+08</td>
<td>6,37E+09</td>
<td>7,39E+09</td>
</tr>
<tr>
<td>7</td>
<td>8,98E+07</td>
<td>1,21E+08</td>
<td>7,98E+07</td>
<td>3,97E+08</td>
<td>2,72E+08</td>
<td>7,44E+09</td>
<td>6,67E+09</td>
</tr>
<tr>
<td>8</td>
<td>1,20E+08</td>
<td>1,27E+08</td>
<td>1,28E+08</td>
<td>2,31E+08</td>
<td>3,72E+08</td>
<td>8,13E+09</td>
<td>7,07E+09</td>
</tr>
<tr>
<td>9</td>
<td>6,27E+07</td>
<td>7,65E+07</td>
<td>1,02E+08</td>
<td>2,23E+08</td>
<td>2,81E+08</td>
<td>7,23E+09</td>
<td>7,09E+09</td>
</tr>
<tr>
<td>10</td>
<td>6,73E+07</td>
<td>7,67E+07</td>
<td>1,18E+08</td>
<td>2,79E+08</td>
<td>3,38E+08</td>
<td>7,37E+09</td>
<td>9,10E+09</td>
</tr>
<tr>
<td>11</td>
<td>6,85E+07</td>
<td>8,51E+07</td>
<td>1,24E+08</td>
<td>3,04E+08</td>
<td>2,71E+08</td>
<td>5,62E+09</td>
<td>1,17E+10</td>
</tr>
<tr>
<td>12</td>
<td>7,22E+07</td>
<td>7,31E+07</td>
<td>8,99E+07</td>
<td>2,18E+08</td>
<td>2,96E+08</td>
<td>1,45E+10</td>
<td>1,21E+10</td>
</tr>
<tr>
<td>13</td>
<td>7,67E+07</td>
<td>1,28E+08</td>
<td>9,32E+07</td>
<td>3,46E+08</td>
<td>3,27E+08</td>
<td>5,74E+09</td>
<td>1,39E+10</td>
</tr>
<tr>
<td>14</td>
<td>9,65E+07</td>
<td>7,37E+07</td>
<td>1,21E+08</td>
<td>2,86E+08</td>
<td>3,08E+08</td>
<td>7,29E+09</td>
<td>1,58E+10</td>
</tr>
<tr>
<td>15</td>
<td>7,39E+07</td>
<td>7,71E+07</td>
<td>8,68E+07</td>
<td>3,14E+08</td>
<td>2,64E+08</td>
<td>1,38E+10</td>
<td>1,40E+10</td>
</tr>
<tr>
<td>16</td>
<td>7,03E+07</td>
<td>7,95E+07</td>
<td>1,70E+08</td>
<td>2,88E+08</td>
<td>3,41E+08</td>
<td>7,30E+09</td>
<td>1,40E+10</td>
</tr>
<tr>
<td>17</td>
<td>6,89E+07</td>
<td>9,40E+07</td>
<td>2,64E+08</td>
<td>4,98E+08</td>
<td>2,97E+08</td>
<td>7,14E+09</td>
<td>1,48E+10</td>
</tr>
<tr>
<td>18</td>
<td>7,17E+07</td>
<td>1,14E+08</td>
<td>8,84E+07</td>
<td>3,33E+08</td>
<td>3,12E+08</td>
<td>6,18E+09</td>
<td>1,64E+10</td>
</tr>
<tr>
<td>19</td>
<td>7,27E+07</td>
<td>8,39E+07</td>
<td>1,01E+08</td>
<td>3,11E+08</td>
<td>3,06E+08</td>
<td>8,33E+09</td>
<td>1,44E+10</td>
</tr>
<tr>
<td>20</td>
<td>6,63E+07</td>
<td>7,87E+07</td>
<td>1,19E+08</td>
<td>3,24E+08</td>
<td>2,91E+08</td>
<td>7,48E+09</td>
<td>1,39E+10</td>
</tr>
<tr>
<td>21</td>
<td>6,54E+07</td>
<td>7,26E+07</td>
<td>8,38E+07</td>
<td>3,25E+08</td>
<td>3,32E+08</td>
<td>5,38E+09</td>
<td>1,57E+10</td>
</tr>
<tr>
<td>22</td>
<td>6,24E+07</td>
<td>1,24E+08</td>
<td>7,96E+07</td>
<td>2,48E+08</td>
<td>2,72E+08</td>
<td>7,76E+09</td>
<td>4,80E+10</td>
</tr>
<tr>
<td>23</td>
<td>8,12E+07</td>
<td>7,53E+07</td>
<td>1,27E+08</td>
<td>3,27E+08</td>
<td>3,45E+08</td>
<td>9,55E+09</td>
<td>1,73E+10</td>
</tr>
<tr>
<td>24</td>
<td>1,05E+08</td>
<td>6,90E+07</td>
<td>8,83E+07</td>
<td>2,07E+08</td>
<td>2,66E+08</td>
<td>7,69E+09</td>
<td>1,63E+10</td>
</tr>
<tr>
<td>25</td>
<td>1,16E+08</td>
<td>7,56E+07</td>
<td>9,43E+07</td>
<td>2,17E+08</td>
<td>3,19E+08</td>
<td>7,38E+09</td>
<td>1,74E+10</td>
</tr>
<tr>
<td>26</td>
<td>6,84E+07</td>
<td>1,00E+08</td>
<td>1,30E+08</td>
<td>2,76E+08</td>
<td>3,21E+08</td>
<td>5,90E+09</td>
<td>1,88E+10</td>
</tr>
<tr>
<td>27</td>
<td>6,35E+07</td>
<td>1,19E+08</td>
<td>9,88E+07</td>
<td>1,99E+08</td>
<td>3,09E+08</td>
<td>9,79E+09</td>
<td>1,65E+10</td>
</tr>
<tr>
<td>28</td>
<td>6,49E+07</td>
<td>7,15E+07</td>
<td>8,17E+07</td>
<td>1,66E+08</td>
<td>1,05E+10</td>
<td>1,83E+10</td>
<td>1,61E+10</td>
</tr>
<tr>
<td>29</td>
<td>6,78E+07</td>
<td>8,48E+07</td>
<td>1,21E+08</td>
<td>2,08E+08</td>
<td>2,74E+08</td>
<td>1,33E+10</td>
<td>1,62E+10</td>
</tr>
<tr>
<td>30</td>
<td>6,64E+07</td>
<td>8,52E+07</td>
<td>8,54E+07</td>
<td>1,78E+08</td>
<td>3,23E+08</td>
<td>7,51E+09</td>
<td>1,67E+10</td>
</tr>
</tbody>
</table>