NPM Chat API

Final Year Project
B.Sc.(Hons) in Software Development

BY
Cormac McHale

May 6, 2020

Advised by Gerard Harrison
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

List of Figures

Abstract

The inspiration for this body of work came from my remembering of the fascination I had with the seeming complexity of importing libraries into my projects early on while learning web development. One of the more mysterious aspects of learning to build real-time applications is trying to figure out the background processes that occur when using small method calls to start a large complex chain of processing events. This would usually inspire me to dig deeper into documentation and source code provided by accomplished teams and companies. As such for my project I was hoping to create something that could one day provide a similar experience for someone else. I also liked the idea of producing usable software to make life easier for a peer as had been done for me many times in the past. In the end I decided that providing a chat service for a novice developer to utilise in their app might achieve this. With the advent of social media and how the platform is shaping society, having a live chat functionality in your application can be a major selling point and an interesting thing to implement while building your first app. The project had some clear high-level guidelines after this decision. Build a package, importable through package manager software, that provides method calls to allow users of a web application to chat in real time. Then to achieve the necessary scope required for a final year project, construct the associated cloud infrastructure and wire the components into a small distributed system. The following document details how I completed this task.

Chapter 1

Introduction

1.1 Overview

The project is a JavaScript API that allows a web developer to make use of web-sockets for communication however they should see fit inside their web application. It was designed with Angular web applications in mind but will work in any JavaScript runtime environment. It is a Node.js package and is stored on the NPM[1] registry. The idea behind the project was to build something that a developer could use while building their own web application. As online communication is becoming increasingly relevant and important in society most developers, either consciously or not will, be taking it into consideration while building an application. My intention was to streamline the implementation of a live chat functionality for a novice developer so they could have a real time chat system between users on their app without having to worry about the infrastructure required for such a feature. This in turn allows them to focus on the content provided by their application. In the following introduction, I hope to outline and provide an explanation for the key components comprising the project along with the objectives in mind at the beginning of development to give the reader a better understanding of why I chose to build the API and implement the system as it is now.

1.2 Objectives

The main vision for the project was to develop a streamlined API that would allow a novice developer to add functional real time communication to be used by clients of the application. The implementation focused on providing short method calls to allow the user to quickly establish fast communication between clients, the user would simply have to "npm install [packagename]", import the JavaScript components and call the methods on the object to establish a channel of communication between clients. The user can then do whatever they might want with the information being distributed in terms of organisation and

display to the client. The API is intended to allow novice developers the chance to implement real time chat systems into their own projects without having to understand or worry about the infrastructure required and focus on learning the basics of app development and providing their content as intended. The objectives of the project were as follows:

- Develop a deeper understanding of Networking communications.
- Learn how to build and deploy a real-time functioning server in a distributed system.
- Learn how to publish my own API's as an independent software developer.
- Develop a functioning distributed system of an appropriate scale.
- Build a usable and effective JavaScript library with the associated infrastructure required.
- Begin to understand the distinction between development and engineering in the context of software and start to develop each as separate skill sets.
- Begin to understand and learn modern cloud deployment techniques.

1.3 Scope

The project is a published build of a working API that can be utilized by web developers. I had to learn how packages like this were developed and distributed. For the API to work for concurrent users it was necessary to implement a cloud-based infrastructure that has full message passing capabilities between components in the system. Deployment of this small distributed system on the cloud and fully implementing communication provided the bulk of the challenge and learning. For the API to fully function the system required a chat server for the users app to connect and relay messages, a server to distribute passwords for users to use as a login to the chat server, and an angular web application deployed as an example of the API and system working. The login/password system was implemented to distinguish users of the API at a server level. The full explanation of the thought behind the design and the actual implementation is documented fully in the system design and system analysis portions of the document.

1.4 Report Structure

This report is comprised of the following sections:

- Methodology
- Tech Review

- System design
- System evaluation
- Conclusion

1.4.1 Methodology

This section describes the software and research methodology used during the development of the project. Along with a breakdown of the approach taken to build the project. It leaned towards an iterative approach following the research. Each component was built through refinements to previous versions.

1.4.2 Technical Review

The technical review contains a full description of the technologies used to create the project. It required an analysis of many different sources to implement solutions correctly. This section reviews these tools in depth to highlight their appropriateness in the context of getting the API to function as intended. I researched the different technologies as much as I could early on to gauge their appropriateness.

1.4.3 System design

A documentation of the over-all design of the system as it was implemented. It contains a description of the system at a high level, and then at a lower level a breakdown of the design of the constituent parts. This section will provide an insight into the design choices used during development and give the reader information on how to apply these solutions in arbitrary situations.

1.4.4 System evaluation

The system evaluation is a documentation of how the system functions in real time. It contains an analysis of how easy the API is to integrate into an app, how effective it is once installed and a description of improvements that could be added. This section highlights the pros and cons of functionality as well as feedback from a development team who use it in their own final year project [2].

1.4.5 Conclusion

My thoughts and conclusions after completing the project. A critical analysis of the approach taken to completing all parts of the project highlighting the things that I feel I benefited most from doing. In this section I hope to clearly outline successes as well as failures to properly establish what I learned from the experience.

1.5 API - finalwebchatclientjson

The focal point of the project is a Node.js JavaScript package. It is published on the NPM registry[1]. To use the package requires an install of 'finalwebchatclientjson' into the users project through the NPM CLI. It exports an object which has a method for the JavaScript runtime to call to establish a connection to a Java chat server through web-sockets by passing a password as a parameter to join a maintained session of web-sockets specific to the application. All clients of an application send the same password and are only able to access information sent by other users of this application. After the connection has been established the user just has to implement the send message method to achieve communication between users of their application. The send method takes in two strings, one for the name of the user and one for the message. It builds a JSON object with these as a key value pair respectively and sends this to the chat server to be relayed to everyone else who using the app at that time. What happens in the logic is the client is using a subscription to an observable that maintains the observer which itself is the implemented methods that a web-socket has as a JavaScript object. The client can now in real-time receive messages sent from other clients. It is up to the developer how they wish to display the messages in a view for the user.

1.6 Github

The Github repository[3] for my project contains the following directories

- References - a list of resources used while building the project
- chatGUI - an angular frontend showcasing the API's functionality
- chatserverproject - The Spring boot chat server
- ScreencastParts – The different recordings for the tutorial of how to implement and use the package
- htmlQuickTesting - some html files used to quickly check web-socket connections
- npmPackages - all the nodes packages developed while learning to build/publish the API
- PythonPasswordServer - the Flask server developed to maintain separate concurrent users of the API
- dissertation.pdf - this document
- README.md – a short description/summation of the project

Chapter 2

Methodology

2.1 Software Development Methodology

The Software Methodology used to develop the final build of the project was the Agile development philosophy. I thought it would be helpful to follow the guidelines of an industry standard as it would give me an understanding of an environment that I may have to work in after finishing college. The key points I took from this methodology were to use an iterative approach to development with an emphasis on building the first working version and then continuously refining with each build. I found this way of working to be very effective for enhancing working components of the system. The resulting product benefited greatly from this. Prioritising communication with the parties involved with development and using this as a tool to drive requirements proved extremely effective. The following sections outline the plan from the beginning along with a description of the approach to each individual component and a brief synopsis of how aspects of the methodology impacted the workflow.

2.2 Planning

In keeping with the Agile philosophy I took an adaptive approach to developing. I planned each component at a high level and adapted to changing requirements as would be discovered on a weekly basis through communication with my project supervisor and final year students from a different module. These students were building their own Ionic project and after some discussions I allowed them, in the context of my project, to take on the role of a customer for the software I was building. The initial plan was documented as follows:

- Step 1

  Develop a simple package that I can publish and import to an angular or ionic project.

- Step 2

Deploy a functioning server and allow it to receive messages through a web socket connection

- Step 3

  Develop a package that has methods to send a message from the client to the server through a web socket connection (needs to send all info required for a message (sender/receiver data))

- Step 4

  Configure Server to handle Messages correctly (pass onto another client, or initially just store messages) store client information, allow them to connect with other clients

- Step 5

  If there is time add complexity (Authentication, users, etc....)

2.2.1 Step 1

I knew that to complete my project fully I would have to learn how to do this. In trying to keep in line with the methodology I developed the simplest package that I could and deployed it to the NPM registry. Once I had done the most basic version of this I was able to incrementally return to the process and build upon my initial work to develop more and more complex packages as seen on my account on the registry[1]. Eventually I had learned the process enough to be able develop a package of sufficient complexity for my API.

2.2.2 Step 2

In the beginning the most basic version of the system only had one server, without authentication or separation of clients. I set it up as a spring boot server with web socket capabilities and slowly over the course of the project, through discovery of requirements in meetings, I iteratively added in the functionality required for the final build. As the requirements developed through communication with my supervisor, the architecture of the system required a second server to authenticate clients through communication with the Java server. This is discussed in more detail in the System Design Section.

2.2.3 Step 3

This was the conclusion to an iterative approach to repeating the steps in part 1. Continually learning to build and streamline the JavaScript package up to the point of implementing a real time chat system using only two methods that build and receive a JSON message object. The development of this section was guided mostly by communication with the Digital Media students who required a chat system for their project.

2.2.4 Step 4

The iterative enhancement of the chat server from just listening on a single port locally to form a connection with a web socket up to separating clients into different communication channels based on which application from which they connect, was the biggest application of the Agile development process. The initial setup for the spring server was a simple process as there are exceptional resources for setting up these projects online[4][5]. During meetings with my supervisor and customers it became clear the scope of requirements would go far beyond this initial implementation. Through the process of adding functionality followed by re-deployment, the single chat server grew into a distributed system architecture to fully enable the API. This required me to learn cloud deployment techniques as discussed in the Technical Review section.

2.2.5 Weekly meetings

The weekly meetings were a very important part of the process. As an Agile environment is based on an iterative cycle, being able to meet every week with my supervisor and discuss the different components of the system as they were being developed I was able to uncover and add in requirements that greatly benefited the over-all design of the project. The meetings were informal, it was more of a discussion of the solutions I was implementing to problems along with what problems could arise afterwards. I tried to emulate a Scrum environment at these meetings to compliment the style of development I was going for, a typical meeting proceeded as such:

- Discussion of the current state of each initial component
- Discussion of the ideal end goals of each component and over-all project
- Deciding on solutions to aim for implementation before the next meeting
- Quick discussion on how customer requirements might affect implementation

2.2.6 Iterative approach

Even though the project was not team-based as I was on my own, I was able to simulate the environment of a one-man scrum with my project supervisor in the role of the scrum master. Each week after meeting we would come to conclusions of what my goals would be for the next week. These goals came from discussing how well each current build of the components of the system were functioning and what requirements could be added to enhance each part. Though there was no pressure from my supervisor to hit targets each week I aimed to complete at least one of the requirements and it helped me to deliver the project in an efficient and timely manner. This I feel is reflected in my source control commits[3].

2.2.7 Customer requirements

For my project it was clear early on that the users of my project would themselves be developers. As such I realised that in order to anticipate the customer requirements I would have to get some of my peers to download and implement my API. Luckily I managed to find real time customers in the form of collaboration with another final year project in a different module[6]. Their app required chat functionality between users. Through communication with them I asserted the main thing they needed was an easy to implement API with very little confusion or setup. This became the driving force for planning how to build the part of the package the user would interact with, the method calls. Through discussions with them as the weeks went on it became clear that the best way to develop the package would be to have as little code as possible required to implement the live chat. I obtained feedback as to how close I had come to the final set of requirements, this is provided in the system evaluation section.

2.3 Research Methodology

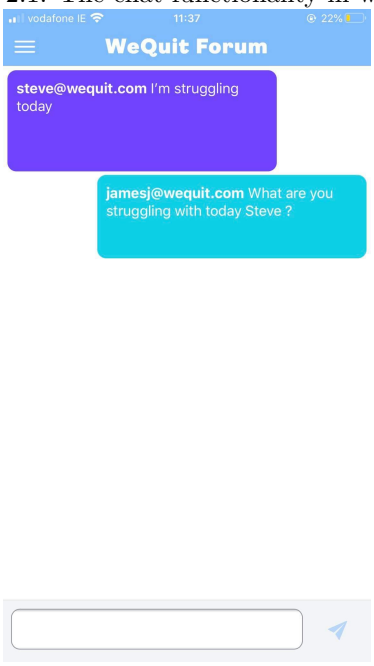
The research methodology that was used in this project was a mixture of qualitative and quantitative research. It was mostly an attempt to have the research go hand in hand with the Agile philosophy. Qualitative data collection methods vary using unstructured or semi-structured techniques, In the context of my project this research came in the form of discussions with my supervisor and customers. As the agile methodology favours "Individuals and Interactions" over "Processes and Tools" I choose to use the feedback from these meetings as the driving force for solutions as opposed to what might be easiest from a technical perspective. This complimented the "Responding to Change" part of the manifesto also, as the research from discussions leans more toward changing requirements than a rigid plan. The quantitative research was a smaller part of the over-all information gathering, I would use this mostly in a local sense when implementing a solution or requirement. This was usually done through the study of applicable documentation or live demos and is reflected in the references. A full list of resources is contained on my Github repository[3].

2.4 Validation and testing

My validation and testing came from a live implementation of the API in an application designed for people trying to quit smoking[6]2.12.2. Through communication with the customers I chose to simplify the implementation to just a few lines of code. Through the iterative process of discovering new requirements during discussion at weekly meetings the logic was condensed into two method calls, with the API providing an array of JSON objects for storing the conversation. This technical breakdown is provided in full detail in the System Design section.

2.4.1 WeQuitGP

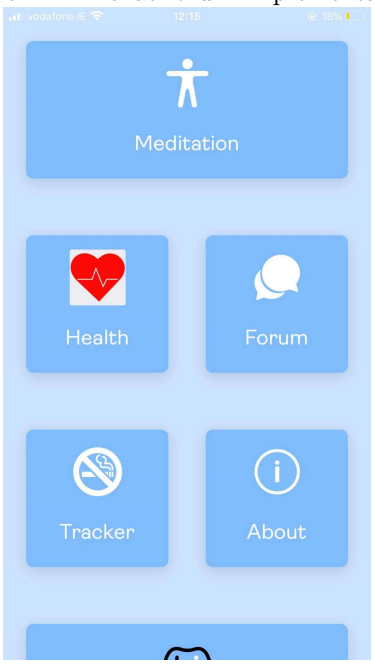
Figure 2.1: The chat functionality in we quit.



2.4.2 User validation

I asked the students involved to provide honest feedback of their experience from the using the API and also to inform of any issues or bugs along with suggestions for improvements moving forward, either for this API or another to be developed at a future date. A full breakdown of this is provided in the System Evaluation section.

Figure 2.2: A chat forum implementation.



Chapter 3

Technology Review

3.1 Http HyperText Transfer Protocol

It was clear from the beginning of the project and the initial discussions that there would be a need for separate processes on different machines to communicate. It was imperative for the ability of the infrastructure to function correctly that each process could rely on some technological standard to do so. As such the communication was implemented using Http methods[7]. As a standardized method of network communication, and the underlying technology for formatting and sending data on the web, using this as the main source of communication between components of the app has allowed me to develop a much more robust understanding of the protocol. At a development level I have learned how to implement it as a viable solution. I have also acquired a good understanding of the general problems associated with it. The web socket connections used for communication required Http handshakes and so required integration at both ends. The chat server and the password server use the request/response model to validate clients before allowing a web-socket connection to be formed. The fronted end view that provides a password for a user of the API is supported by these built in methods as well. As it is so standardised every language and environment had built in libraries to make use of this technology, Spring had web-socket handling libraries for Java and RxJS is a library that can be used in typescript to handle these connections. With these libraries at hand it is as simple as understanding that the information "on the wire" will be handled by the OS and passed into the server environment in a format you can interact with. Once the endpoints are setup you just need to write your methods to handle the information as you need for your system to work. Moving forward I think this technology is very well integrated into most languages and environments and I would recommend that every developer should get try to gain a robust understanding of how it works.

3.2 Json

An open standard file format for message passing between clients and servers in distributed systems, JSON functionality became an obvious solution to an issue that became apparent towards the final few iterations of the project. It was developed through a need for stateless, real-time server-to-browser communication protocol. Initially in my API the JavaScript logic sent a string of plain text to the server which was relayed to each client connected, however it became clear through communication with the customers that it was very hard to logically process once it was passed into memory on the client, at the very least it was unhelpful for a developer to have to organise plain text for the user of their application. So building and parsing Json into objects being sent and received respectively solved the issue. All browsers have native support for Json as it is built into JavaScript itself. The language provides methods for converting and parsing[8] into a JSON object in memory and by extension this functionality can also be utilized in TypeScript. This made the conversation of the plain text into more manageable objects in memory very streamlined as is fully explained in the System design. I would highly recommend the research and utilization of this tool to anyone building an application that uses JavaScript or TypeScript to relay information between components in a system, especially in the context of passing information that might need to be separated into constituent parts. Using open source technologies developed to solve common problems allows you to wire powerful components into your own system to build in a way that is easy to maintain and work with.

3.3 Rxjs

To further illustrate the point of the importance of open source technologies one of the key components used for building my API correctly was RxJS. RxJS is an open source library accessible from the NPM registry to use in your own packages or projects. It maintains a ready to use implementation of observable and subscription JavaScript objects. It was designed to make asynchronous JavaScript much easier to write. The reason this was so important in the development of my API is that having the web-sockets implemented using a push-system was essential and these tools were designed to provide this functionality. A push system is one where the view or logic will update according to data being added to data structures. It is an event driven style of programming where a process does not need to worry about pulling updates every so often as the view is automatically updated on a push. Having this kind of a system already developed meant I was able to use this without having to fully understand or worry about more difficult concepts, I just built the layer of logic on top that is my API. The observable system was the perfect technology to develop a real-time chat system as if you breakdown the flow of information handled it logically makes sense. The subscription object is used to subscribe to the observable which asynchronously returns the values from the event-driven observer. This model

of updating the with an event driven architecture is described in much more detail in System design section of this document. Gaining an understanding of how open source libraries such as this work and realising that other developers have published these libraries to be used to build upon can make it much easier to develop your own powerful components.

3.4 Typescript

Node.js packages are published in JavaScript. I had a lot of experience and enjoy using Typescript to write code for use in a JavaScript environment. This is possible as it is essentially an extension of JavaScript that offers unsound optional type annotations that provide some static error checking for the program where they are used[9]. The idea of an unsound type system can be counter intuitive. It is important to understand that Typescript has inherent unsoundness as it is a syntactic super-set of JavaScript and as such emits it at run time. It is the Job of the Typescript compiler to do this[10]. As such I was able to write and test the package in Typescript. Then I just compiled it to JavaScript before publishing it to the NPM registry. Typescript was developed as a way to build and manage large scale JavaScript applications. I find that Typescript is much easier to maintain as it removes a lot of the clutter associated with complex JavaScript. All JavaScript programs are valid Typescript programs and as such can be used interchangeably with any Typescript code that you are writing. It was intentionally designed like this to preserve powerful legacy components from JavaScript. It is easy to see the benefits of this language, you can write cleaner and more maintainable code that you feel is easy to develop with while still having access to useful open source libraries previously built for web development. I would recommend learning to incorporate this style of writing code as it can be a powerful tool and will also help to understand that there are always ways to optimise your working environment.

3.5 Node.js and npm

Node.js[11] is an open source, cross-platform JavaScript runtime environment that executes JavaScript code outside of a browser. It was developed in 2009 with the intention of improving upon the existing web server technologies at that time. In 2010 Isaac Schlueter created a package manager called NPM and it was introduced into the community as a tool for development of open source JavaScript libraries. It maintains a database where you can publish and share source code for other developers to install and use in their own projects[1]. The ability to be able to store my library on a registry where other developers could access my API was a crucial part of why I choose to make a JavaScript package. It meant that I could easily publish my package after it had been developed. JavaScript has become one of the most powerful and standardised tools available for web development. I wanted to build an API that a novice developer could

easily use without having to understand some of the more difficult concepts associated with a real-time chat feature. Open source tools such as this are the reason why we see such rapid growth in particular areas of technology and using this has really helped me begin to learn the importance of such tools and also how to become a contributor to open source technologies.

3.6 Angular - needs to be proof read from here

For the purposes of the project I have deployed an angular web application that makes use of the basic functionality of the API[12]. It is a progressive web app where anyone can go and live chat after selecting a name to represent themselves in a Group chat scenario. A progressive web app is application software intended to work anywhere there is a standards compliant browser. This is demonstrated in a screen cast of the web app. Angular is a web application framework maintained by Google[13]. It is based in Typescript and contains powerful pre-built components for building a web-applications. It is specifically concerned with the development of SPA's (Single Page applications). The framework itself comes with powerful routing protocols for swapping between views for the user and the documentation is well put together and informative. I would especially recommend this for anyone looking to learn modern web development. Many cutting-edge concepts are packaged into the starter template in an easy to use format.

3.7 Linux (Ubuntu) and Oracle VM Virtual box

It became clear during the early stages of development that I would also need to deploy the architecture to the cloud to fully implement the system. After some research[14][15] I choose to install the popular Linux distribution Ubuntu onto my hard drive to access and use it as a separate machine through Oracle VM Virtual box. I found this is be a steep learning curve during development, I cannot recommend this enough. I was able to spin up the machine through the virtual box software and begin learning how to develop on a Linux system. Once I had familiarised myself with the operating system I began to enjoy the nuance of how easily it would allow you to install and utilise open source programs. The main benefit enjoyed here was the compatibility with docker, once I had familiarised myself with the docker commands I was able to replicate the environment with ease on the cloud and continuous integration became very easy. This lent itself well to the iterative Agile process as I was able to make changes locally to a component and re-build in docker and deploy to the cloud in just a few steps.

3.8 Digital Ocean (Cloud Service Provider)

Digital Ocean is a cloud service provider based in New York, it has world-wide data centres. I choose to use this service provider just to see how it was different from the other major cloud service providers. Essentially it is much the same. While creating a profile you can simply add in preferences in terms of operating systems and processes you are using. A very nice feature was that after this selection I was shown a GUI where I could select a few different machines optimized for my preferences. I was able to generate a Ubuntu machine with docker installed. This made the setup for my project extremely efficient. All that was required once the machine was running was too SSH into the root of the machine and pull the images from docker hub, map the ports of the docker images to the actual machine and begin a continuous integration of the final build for my project. The setup was now stream-lined such that every time I made a change locally to a component I could re-build that docker image and test functionality locally, then push the new image to DockerHub and replace the old versions on the VM. This workflow aligned well with the Agile style of development I was trying to achieve. This part of the project was the only tool that was not open source. There was a small reasonable fee per month to keep the servers running. Which was fine as I will continue to use the service during my career.

3.9 Docker

To properly deploy the architecture, I had to utilise tools that would allow me to have multiple machines running on the cloud. For this I chose Docker, Docker is an open source program that allows you to build miniature virtual machines in memory optimised for a process you need to run. This program or process then functions inside of a virtual container inside Docker which itself runs as a program on your machine. This was a steep learning curve again as we did not cover anything like this during any modules, however by the time I had finished working with the software I had learned a lot about automated deployment. I would highly recommend that anyone building a small distributed system for a college project to familiarize themselves with a tool like this. It is not only an effective tool for deploying efficiently to the cloud but almost certainly something you will have to learn to work with in the software Industry. With the growth of cloud services over the last decade building this skill set is imperative for a someone looking to have a career in the software industry. The logic seemed not to be overly complex from researching the program and workflow. The setup was as follows download and install docker, which is utilised though command line arguments. Add a docker file to the hierarchy of the folder containing the compiled binary for the program. Add the requirements (discussed below) to this file and run the necessary commands for building and running. As expected, there was some initial trial and error to understand compatibility and what the commands did after execution but once I had learned the workflow, deployment

to the cloud became trivial. This greatly increased the effectiveness of the iterative improvement style I was aiming for. You can test these images by running them on your local machine before officially deploying them. What I found to be helpful about docker is that the docker program running locally is going to be the same environment as is on the cloud, a lot of the requirement and dependency issues associated with cloud deployment would already be resolved. This seemed to be the selling point from information I had gathered[15]. After building an image for a program which is a virtual machine with the minimum requirements for a process, run it inside a docker container locally to test then once verified push it to source control to allow it to be pulled into a docker environment on another machine. In my case this was the cloud infrastructure required for the project. Docker Hub[15] is a service that functions like Github for these images allowing you to push and pull machines you have created to your repository if you are satisfied with the result. As mentioned above once this is done cloud deployment become a very streamlined process. It is just a matter of spinning up a VM that has docker installed, pulling the images and running them using the docker CLI. The workflow of docker was something that really surprised me in terms of how optimal it was for deploying new versions of a program. I mostly followed online tutorials[16] coupled with trial and error for learning the keywords and commands. Using my own flask server as an example the general setup is:

- Generate the Docker File in the same directory as your compiled binary or source code for the program
- Add the FROM keyword followed by the name of the machine that will be the base image for running the program (i.e a dockerised version of a machine that will host the server)
- Use the RUN keyword to specify commands to setup the basic requirements for the server (i.e libraries/packages)
- Use the COPY keyword to copy the binary file or source code and any other file required for running into the root of the container
- Use the CMD keyword to specify command to run on the command line of the container (These can be used to start the server when the container is spun up)
- Run "docker build -t [NameOfContainer]:latest" to build the image
- Finally run "docker run -d -p 5000:5000 [NameOfContainer]" to spin up the server in the container
- "-t" - tag command - name the container with a tag
- "-d" - run the container as a background process
- "-p" - map the ports of the docker container to the actual machine as the ports of the server only know about the docker process it is being run inside off

3.10 Java, Spring Boot and Maven

I decided to use Java for the developing the initial chat server for a few different reasons. We have had the most exposure to the general purpose language in the four years spent at GMIT. I knew from planning and research that the server side component would have to accept a web-socket connection as a base requirement. From that point on I felt that in terms of growing requirements my experience in this language would help me overcome any challenges that might arise. The language has been around for almost two and a half decades and remains one of the most popular languages available for development. As such I felt that using this technology for a server that might require some general purpose solutions later, would be a good idea. The main requirement from the beginning was to have the server accept a web socket connection, as the client side request would be made from a node.js environment I decided through some discussions with my supervisor that using the JVM for the server side environment would also bring the scope and complexity of the project up to an acceptable standard. For setting up the project, in keeping with the same vision I had for my project, I opted to use a service designed with the needs of a software developer in mind. Spring itself is an application ready framework designed for java programming. It provides the general setup for most aspects of an out of the box Java application, security, MVC, testing, data access and native integration with tomcat for web development. As my main idea for my project was to build something another developer could make use of, though on a much smaller scale, it seemed appropriate to wire these pre-made components together in order to make use of the open-source technology as it was intended. If you navigate to the spring starter website[4] you can download a project with whatever packages you may need pre-installed. You can then import that into eclipse to begin development. So the Chat Server started as a Spring Boot project that uses the built in implementation of web sockets[5] in order to form connections with clients. I decided the best way to emulate a live chat functionality into the server would be to base the logic, at a high level, on the observer model[17]. To have the server itself as the observer with the clients as the many observables to notify when a message had been sent from one. This ties in nicely with the RxJS implementation of the logic client side where you subscribe to the observable to update the view as the messages are pushed out to the client. The full explanation of how this works is discussed in the System design section. Maven was used through the iteration of this component as the build tool, it is a necessity for someone building a project as a single party. Maven can download new packages required for the project while maintaining dependencies along with compilation to an executable and testing. A powerful open source tool for getting Java applications production ready. As Java has been around for so long the tools used in this component really felt optimised and complete, I think that having knowledge in this area of development will be of great benefit moving forward.

3.11 Python and Flask

Towards the end of the project the problem of separating different users of my API into different channels of communication became apparent. During the meetings I put forward the idea of increasing the scope and complexity of the cloud architecture by adding in a separate server to solve this problem. That is to use a different Language and web architecture and implement complete communication between all three environments as a solution to this problem. The choice was made to build the final component using the Python based web framework Flask. Python[18] is a multipurpose scripting language that wires together powerful C components 'under the hood' to produce functional, readable code. It is becoming more popular and useful every year due to the ease of use. This kept in line with the ethos of the project of using open source tools designed to help other developers. Python has a micro web framework called Flask. It requires no extra tools or libraries which formed a nice juxtaposition with the Spring Boot project as that uses many different bodies of work. Where the Java server required planning and more carefully considered code and setup this server is contained in a single script. I felt that implementing a scripting language as a solution added to the scope and complexity of the project by bringing a greater range of computing concepts into a system. This was something I benefited from as working on all these different style and ideas at the same time helped me develop a better understanding of the differences between them.

3.12 Source control

I worked in a team on my own so source control was not an issue between different members in my case, however the final project took place on three separate environments as such the source control tools helped a lot in these circumstances. For managing versions of different components, I used Github and Dockerhub.

3.12.1 Github

A staple of modern development and an essential skill for any programmer. Github is a tool for software development version control using Git. It has evolved over time to provide many different tools for managing a software project. I added my supervisor as a collaborator on the project so he could also track my progress. This allows for the easy inspection of code without having me to organise a meeting to show it. It was useful to have a single space to store and manage each of the different components over time. The online repositories at Github.com have useful tools like the issues tracker. This allows you to document bugs or requirements and explain how you managed to solve them. Over time this can develop into great documentation for the code you have written. It has helpful GUI components for project planning and assigning

work if you are in a team with multiple members. A great tool for learning about source control and project planning.

3.12.2 DockerHub

DockerHub[15] is a repository for images built for running programs. A very useful tool for transferring and deploying images to the cloud. I only used this tool to push and pull images. Moving forward I would like to develop my skills with this tool.

Chapter 4

System Design

** Diagram here tbc..

4.1 Over-view

The API is intended to provide a real time chat functionality into a Node.js environment. Node package manager is the installation tool. Once the package has been installed, the developer must obtain a password from a web page provided as part of the service. This is so the chat server can distinguish between users of the API. The server that hosts this web page stores the generated passwords and uses them later for validating web-socket requests. The users of the API can then follow the workflow of the tutorial[19] to implement the chat functionality. The developer simply calls the methods provided by the API to form a web socket connection to the chat server. The connection is validated and accepted when the server parses the password which gets sent as an optional parameter in the "sec-web-protocol" request header in the http handshake. The chat server makes a request to the python server to make sure that the password was generated and given to a user. Once the web-socket connection is formed it is placed into the same channel of communications as every other connection that requests with that password. The following section explains in details the design of each component in the context of solutions to requirements and how they fit together for the final build.

4.2 NPM API (finalwebchatclientjson)

4.2.1 Designing for the client

As the project is something made for developers, client in this context describes someone implementing the API into their own project. This section explains the design choices taken in terms of ease of use to this person or team. The node package manager can install this into the node modules folder of a Node.js

project by running 'npm install finalwebchatclientjson'. The main design philosophy behind how it would be presented came from talks with the client who wanted to implement chat functionality into their app. The development style was iterative improvements of previously working versions coupled with discussions between myself and the students who were developing WeQuitGP[1]. The package initially had multiple method calls to connect to the chat server, the developer would have to use the RxJS subscription object to manually subscribe to the observable returned from the package. This was confusing to implement for students who did not have a direct background in software development, the choice was made in the end to abstract all the background logic into the package. The students felt the best way for the package to be presented to another developer was to make the implementation light with the method name having heavy inference of what was happening. This was easy to resolve as I only needed to show that there was a connection to the server along with a send message function. I exported a JSON array to represent the stack of communications between clients using the live chat feature. This made it easier to deal with in memory for displaying. The clients can send a user name along with a message which gets put into this array as a JSON object with the user name as the key and message as the value. Eventually it was reduced to two simple method calls with the package exporting three items:

- WEBSERVERLOCATION(ip address to connect to the chat server),
- CONVERSATION(the array of JSON messages)
- WebChatConnection(the object that maintained the associated methods for connection and sending a message)

WebChatConnection

The main object in the package, this contains the method calls for connection and sending messages. It does this by taking in the WEBSERVERLOCATION and a password provided by the user. WEBSERVERLOCATION is the public ip address of the chat server, this did not necessarily need to be added to what the user was interacting with, however I felt it was appropriate to show that the websocket was connecting to a particular destination. The password was a means to separate users into different channels of communication. This requirement was added in later in the project as it became obvious that to deploy this package as something multiple users could implement concurrently there would be a need to distinguish application clients from each other. Otherwise everyone using the package would receive every message sent from any client. How this separation works is a process handled server side between the chat server and the password server.

4.2.2 Technical design

The internal technical design of the package is as follows, it has the exported members as discussed above along with observable, observer and subscription

objects imported from the RxJS library. The object WebChatConnection contains all the logic for connection and sending a message. It has three instance variables. A standard web socket, a variable initialised to one to compare with the ready state of the web socket (a check to ensure the connection is open to send a message) and a subscription object required to link the observable returned from inside the connection method to the observer which sets up the update on push architecture. In trying to abstract all the logic away from the user I packed the entire connection, observable and subscription process into a single method on WebChatConnection. The class has a private method createConnection() that returns an observable, this begins by opening the web socket using the password and the address of the chat server. The observable gets created and an observer is setup inside the observable. The observer's next() methods are implemented by placing event handlers on the web sockets on message(), onerror() and onclose() methods. These web socket methods will now alert the observable when the web socket receives a message, throws an error or is closed down. The chatServerinit() method seen by the developer uses the subscription object to subscribe to this private method. Any time data is pushed to the observable the subscription object grabs this data parses it to a JSON object and pushes it into the CONVERSATION array. This array will now live update for the user as messages get sent to each client. The sendMessage() function takes in a user name and a message sent from that user. It build a Json object and then sends the JSON.stringify() version to the chat server through the web sockets send() method to be relayed to all other clients. This allows the clients subscription to parse the message correctly and build up the conversation array for display. This is the entire setup for the push architecture that allows for a live chat.

4.3 Cloud Architecture

To have the API working correctly I designed an infrastructure that would allow for the chat service to be available all the time. I built a chat server for relaying messages and a password server to maintain unique passwords for making web-socket connections. I deployed both servers to a virtual machine hosted on the cloud. The cloud deployment consists of the servers running in two separate docker containers on a Ubuntu VM. The chat server is a Spring Boot server which sits listening for requests for web socket connections on port 50000, if a connection is successfully formed it allows and maintains that connection, relaying messages appropriately, until the client closes the web-socket. The second server is the flask server. This hosts a web page that a user of the API can visit to obtain a password to be used while requesting a web socket connection. The distributed system exists as a network within docker on this machine. For these processes to be able to communicate properly I assigned a floating IP address to the host machine and had all requests to that IP re-direct to the actual machine. The servers communicate by making requests to this IP address. Digital Ocean provides the functionality for assigning a floating

IP address. This was used as it made it easier to implement communication containers inside the same docker process. I think this was an issue with getting docker containers to message pass correctly. The screen cast [19] of the project explains the work flow of installation of the API as it is intended for a developer.

4.3.1 Communication

Communication between the different processes in the system was vital for the success of the project. The API itself is one that provides a communication function to an application. All the message passing between components is done using http. The connections formed between the clients and the chat server are the built in JavaScript web socket connections. The chat server uses a POST http request, with the password used for requesting a web socket connection as the payload, to the password server. If the password server has generated and saved this password it returns a status code of two hundred and the connection is accepted and formed. All following connections are placed into the same channel of communications as a group of observers. This was a solution to two major requirements. One was having the ability to have multiple applications using the API. The other was having no collisions from users accidentally selecting the same passwords.

4.3.2 Virtual Machine setup

The virtual machine itself is setup with docker as the main process. The two servers are running inside containers run by docker. The ports are mapped from the containers to the machine's actual ports. This is all done through command line arguments. Once the machine has started up, I SSH into the command line and run the following commands:

- docker pull cormacmchale/chatserver@latest
- docker run -d -p 50000:50000 cormacmchale/chatserver@latest

This starts the server as a background process and the same is repeated for the password server and the setup is complete.

4.3.3 Chat Server

The chat server functionality was developed with the observable design pattern in mind. It is a Spring Boot server and it uses the implementation of web sockets in Spring to establish connections as objects in memory. Tomcat is the default web server for Spring boot projects, this sits in between the operating system and the chat server to handle and direct requests. At a high level the server establishes and maintains these connections as observers, when an observer sends a message to the server, or observable, the message is relayed to the other observers in that group. It separates these observers by using a password system. The API requires an optional parameter to form a web socket connection to

this server. This optional parameter is passed to the server in the "sec-web-soc-protocol" header. The spring libraries allow you to place handshake handlers on theses connection requests which allowed me to overwrite and utilise this extra parameter in memory. The server accepts the initial request for the handshake and before the connection is formed it makes a POST request to the password server with whatever string was in that header. If the password server can find that string in the stored list of passwords on the server then it replies with a status code of two hundred. This lets the chat server know that this is a valid password. The session is then passed to a database of sessions in memory. At a low level it maintains a hash map that links a password to an array list of web socket connections. If the password is contained in the hash map as a key then the session is added to that group of observers and in the context of the API functionality is now added to the live chat for that application. When a client requests to close a connection an event handler is fired which finds the client in a group of observers and removes it from that list. This stops messages from being sent to empty parts of an array list which will cause a run-time error in java. When any of these observers send a message to the server it relays the message to all other connections. All the data structures used here were concurrent when necessary. This was the final design that implemented, what I had decided in meetings throughout the year, the full set of requirements needed for the chat server to be fully functional for the API.

4.3.4 Password Server

The Python server is a simple server that generates a GUI for the user when the visiting the IP address of the server from a browser[20]. When they click the generate password button it generates a unique sign-in id for a client to form a web-sockets connection with the chat server. These generated passwords are saved to a file on the server and an endpoint is provided for the Java server to make requests to validate web-socket passwords against this file of generated passwords. The main functionality behind the design of this part of the system was to avoid collisions between users of the API. This controls the selection of passwords so no two users of the API can have the same password. This prevents web-sockets from being improperly sorted server side.

4.3.5 Web application example

The angular web application was built as an example to show the API fully functioning it is displayed working at the end of the tutorials provided as part of the project. I wanted to build an application separate from the final year project discussed in previous sections. This was used to show multiple Node.js applications using the same API concurrently.

Chapter 5

System Evaluation

5.1 Overview

This section describes in detail the processes and actions taken to validate the final version of the system. To assert that the API was fully integrated into another project and was functioning as intended. In the context of my project the customers were a different development team, hereafter when I describe the "customer" I am referring to the WeQuitGP team. The various forms of testing used were:

- Black-box
- Exploratory
- End-to-end
- Integration

With all the testing techniques used I attempted to tie them into the Agile philosophy that I had chosen to use. That is that to say that after each iteration I would use testing to find bugs to adapt into requirements through communication with the customer or my project supervisor.

5.2 Robustness

I wanted to build a robust product for a developer. I decided early on that the best way to do this, in keeping with the agile development methodology, would be to build the first working version I could and allow the customers to integrate this early version of the product into their application. This was a form of Black-box testing as the customer would naturally produce inputs and record outputs while providing feedback on how they would desire the API to function. I would look for feedback through discussion and then through iteration and improvement, work towards the final product by implementing

new requirements. I felt that this was in line with the development methodology I had chosen. This Black-box testing also covered the end to end testing in a sense as for messages to be sent and received correctly they would have to be passed correctly through the entire system. I felt this to be the most effective form of testing used throughout the project. As they needed their project to be developed properly all communication about improvements were transformed into tangible requirements for the app. This also gave way to the most significant bug discoveries as discussed below in the Known Bugs section.

5.3 Usability and Integration testing

The benchmark for the usability came mostly from exploratory testing followed by integration testing. This type of testing is widely used in the agile development process. While handing off the API to the customers I also built my own Angular web application[13] which used the API to simulate a live chat room. As I was building the API I would deploy and use the implementation in this environment to further the scope of testing, if the API functioned correctly in an angular web application environment and also an ionic hybrid application then I considered that to be an increase in robustness. This style of testing was used to observe any major differences in how the product would function in different node.js environments. As I was using docker, which is a tool used for automated building, it became a very streamlined process to provide full system integration testing. Whenever a component such as the chat or password server had a requirement added. I would re-build the system and re-deploy the project on the virtual machine on the cloud. All that was needed was to re-compile the components, re-build the docker images, push to DockerHub, pull to the machine on the cloud and run the Docker commands to re-deploy. I felt I was able to emulate a continuous integration process using the automated build tools provided by Docker, being able to continuously build and check integration greatly decreased the severity of the bugs introduced to the project through iterative development. I was able to use the live angular application to ensure the system was still functioning as intended after each cycle before getting feedback from the customer.

5.4 Live Implementation (WeQuitGP)

WeQuitGP is a fourth year Digital Media project[2] which makes use of the API. It is an android application for people trying to quit smoking. They have a section of the app where users of the app can have a real-time live chat as advertised in their promotional video[6]. The feedback so far has been positive in terms of the functionality of the API. As it runs in a node.js environment it deployed to the Android and IOS versions of the phone without much difference. Their application has a Firebase login system which allows people to login and register at the home screen before being able to access the main functionality of

the app. This allowed them to use the name of the current user as the sender of a message. Collaboration and communication with this team was one of the most effective forms of testing and development I have ever used. As they already had a robust set of requirements of how they wanted the API to function for ease of use. I was able to use their needs to drive development of the project in the right direction.

5.5 Limitations

As the project is now finished it seems appropriate to discuss the inherent limitations of what was eventually developed as the final working version of the project. As a single person working on a team there were areas that either I did not have time to examine, or could have used the help of a team member who had more specialised knowledge in the problem domain. The limitations explain the bugs that remain in the project. The main areas of limitation are that of scalability and security. These are admittedly quite significant issues to have as any research will highlight the importance of these two things in a software system. Lack of scalability testing comes from the fact the as a single team member working on continuous integration of the entire system it was infeasible to be able to devote enough time to implement a truly scalable system while keeping within the time constraints of the project. Security suffered in this way also.

5.5.1 Known Bugs

- Android minimum sdk 28
- 4g web socket

Android 28

At the very end of the project in a meeting with the WeQuitGP team. They noticed that building for android while targeting the minimum SDK as version 28.0 the API would not work correctly. I did not have time to debug this in android studio as this happened too close to our due dates for projects.

4g web socket

The API does not function with cellular data. I believe this to be an issue with connecting to unsecured endpoints through a cellular data tower however I am unsure. This would be an area of security research if I returned to the project.

5.6 Feed back

Upon completing the library and building the infrastructure I was in the advantageous position of having the live implementation of the API deployed by

another development team, probably the most dynamic aspect of the project was integrating my own project into another. This revealed most of the issues that would need to be resolved if I were to carry on the project after it has been submitted for correction. I asked the team to provide some small feedback for the effectiveness off the API, I have provided this here:

> "It was good" -James O'Malley

Chapter 6

Conclusion

6.1 Goals

From the outset of the project, there were two main things I was focused on in terms of learning. I wanted to develop a deeper understanding of communication within a distributed system and learn about the process of publishing a useful body of work as an independent software developer. I felt that if I could build this system the way it was intended I would be able to satisfy the goals I had set in the beginning. To address these goals in the context of the project being complete I provided some thoughts on what I have learned while completing each one.

6.1.1 Develop a deeper understanding of Networking communications

I knew going into the project that I would not be able to develop my knowledge to the standard I truly wanted, I feel with most areas of modern computer science that there has simply been too much progress and development to be able to achieve a robust understanding of a single topic in a short space of time. In the context of completing my project I was at least able to build a system that supported real-time communication between multiple processes that had no native knowledge of each other. As I also discovered from the System Evaluation it is probably the area where the most improvement could be made if I return to enhance the project in the future. I was able to begin to understand the general problems associated with this aspect of a system. I felt I had laid the ground work for developing my own problem solving process when it comes to networking communication. This is something I hope to build upon each year moving forward.

6.1.2 Learn how to build and deploy a real-time functioning back end Java server

In retrospect one of the less involved goals of the project. We had a lot of experience in college building Java programs that were fully functioning. This goal was more to do with building a functioning part of a system that someone else would use with no knowledge that the server was working somewhere along the distribution of processes. With all of the support for building back end Java programs, and the support for the language as a general purpose programming language, there was more than enough resources to accomplish this. I was happy with how it was deployed and functioning in the end.

6.1.3 Learn how to publish my own API's as an independent software developer

Something I was very excited to achieve. I really wanted to start to build the skill set of developing and publishing my own software packages that someone else could use for their own application. Throughout the course of the project I published several packages while trying to learn a general workflow[1]. This is probably the area that I would be most interested to follow up on after the project. I think I could put more time into learning how to unit test code correctly to build more unique and robust packages. Learning to build and publish something that another developer could use I found to be very interesting and rewarding. I also believe this to be the most effective way for developers to make continuing progress in terms of application development. Following on from this I would love to be able to contribute to open source technologies.

6.1.4 Develop a functioning Distributed System of an appropriate scale

I knew this would be required before the package could function as a service for another developer. As a single person working on the project there was a trade-off between what could be accomplished for the three major components of the system. To try and achieve this goal I implemented the most basic fully working version I could. As such certain aspects of building a robust distributed system were left unimplemented. To be specific, security and scalability were affected the most. I simply did not have the time to implement fully functioning solutions for these on my own. I understand the importance of these requirements however and if I could re-visit the project with a team these would be the first aspect I would focus on for improvement. I was happy that I accomplished what I had set as a goal at the expense of certain aspects as the system did work in real-time.

6.1.5 Build a usable JavaScript library with the associated infrastructure required

This ties in with the goal of learning to build and publish an API and building the distributed system. This was one of my main goals as achieving this meant the project was completed, working as intended. As seen from the deployment in WeQuitGP[2]. I was satisfied I had achieved what I set out to do here. This is something I plan to expand to different languages and platforms.

6.1.6 Understand the distinction between development and engineering in the context of software and start to develop each as separate skill sets

I think for most developers this is one of the more nuanced parts of developing understanding. I wanted to separate the idea of just producing a solution as a student from understanding and building the best version of a system. That is to engineer the system correctly and not just to be concerned with coding software solutions. I feel this was where it was the hardest to make progress as being a student requires you to implement solutions for marks, with time pressure you don't have as much breathing room to sit back and re-design components whenever you might need. As such this is something I feel I will need to invest more time in moving forward.

6.1.7 Understand and learn modern cloud deployment techniques

For this goal I just wanted to begin to understand automated building and deployment at any level I could achieve. Once I got the hang of building docker images I started to understand just how effective this type of development is. It is very easy now to understand why this has become such a powerful tool for companies. In terms of building a distributed system it completely streamlined the deployment of components to the cloud infrastructure. I think this was the most significant new skill I developed throughout the project. I think moving forward knowledge in this area will become more and more valuable.

6.2 Conclusion

I think one of the most important things I learned from the project and development process was how important the customer requirements are. I was fortunate to have managed to get some customer feedback throughout development and it made discovering meaningful requirements much easier. Looking back, the discussions with the WeQuitGP team impacted parts of the system they had no knowledge of. Having a clear understanding of who you are building it for can greatly increase your chances off adding beneficial features to a product. Learning the automated build and deployment process has also changed my view on

designing systems. I was amazed at the solutions that have been developed to counteract the issues associated with deploying processes across multiple environments. I think I will spend much more time in the future learning the different aspects to this. It is a very relevant and helpful skill in modern development. I also have a better understanding of the scope of building a system and how important it is to have a team to build it correctly. Moving forward I am excited about future projects and feel I have a better understanding on how to go about building more robust and effective systems.

Bibliography

[1] Npm registry. https://www.npmjs.com/settings/cormacmchale/packages.

[2] We quit github link. https://github.com/DigitalMediaGP/WeQuitGP-master.

[3] Github. https://github.com/cormacmchale/npmChatApi.

[4] Spring start-up. https://start.spring.io/.

[5] Spring web socket. https://spring.io/guides/gs/messaging-stomp-websocket/.

[6] We quit promotional video. https://youtu.be/9fZAz88SFsE.

[7] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. 1999.

[8] Angular documentation. https://javascript.info/json.

[9] Gregor Richards, Francesco Zappa Nardelli, and Jan Vitek. Concrete Types for TypeScript. In John Tang Boyland, editor, *29th European Conference on Object-Oriented Programming (ECOOP 2015)*, volume 37 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76–100, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In *European Conference on Object-Oriented Programming*, pages 257–281. Springer, 2014.

[11] Stefan Tilkov and Steve Vinoski. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.

[12] Frontend example. http://46.101.229.154:4200.

[13] Angular documentation. https://angular.io/.

[14] Hey docker! why do you hate windows so much? https://medium.com/faun/hey-docker-why-you-hate-windows-so-much-de7a7aa4dd7.

[15] Dockerhub. https://hub.docker.com/u/cormacmchale.

[16] Docker tutorial. https://angular.io/.

[17] Dominik Gruntz. Java design: on the observer pattern. *Java Report*, 2002.

[18] Guido Van Rossum et al. Python programming language. In *USENIX annual technical conference*, volume 41, page 36, 2007.

[19] Screen cast tutorial. https://youtu.be/PH3hp4UcDCQ.

[20] Password generator. http://46.101.229.154:5000.