University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

StreamingOS: Low Cost Education System

Detailed Design and Project Timeline Report

Group 2020.15

Prepared by:
Anurag Joshi – 20604210
Matthew Milne – 20626854
Surag Sudesh – 20636861
Vidit Soni – 20627647
Vinayak Sharma – 20585279

Consultant:

Wojciech Golab

June 28, 2019

Table of Contents

1.0 High-Level Description of Project

1.1 Motivation

As technology improves in the 21st century, using mobile devices for educational purposes is becoming increasingly more common in primary and secondary schools. However, the devices come at a high cost. In the United States, it is estimated that the cost of educational technology ranges from $142 to $490 per student [1]. In addition, this technology becomes quickly outdated and needs to be replaced, forcing schools to spend continuous amounts of money on maintenance. With StreamingOS, we aim to alleviate these costs by providing students and teachers with inexpensive thin endpoint devices, with the resource-heavy applications/OS being streamed to these devices from a backend server using container virtualization.

1.2 Project Objective

The objective of this project is to design a powerful and inexpensive device and streaming system that enhances the learning experience. StreamingOS uses an inexpensive Raspberry Pi (or alternative) and container virtualization to visually render and stream the execution of applications from a server or the teacher's computer to these devices used by the students. The system design leverages concepts learned in distributed computing, operating systems, database theory, and networking courses. The advantage of this design over current alternatives is that it is scalable while enabling the teacher full control of what software each student views. Due to the inexpensive hardware, it breaks down the barrier of the lack of technology in school settings and empowers teachers to incorporate more modern-day means of learning in their classrooms.

1.3 Block Diagram

The proposed solution consists of a variety of different components which enable the functionality of streaming the operating system to a cost-efficient device that is provided to students and teachers. There are individual client devices (provided to students and teachers) running instances of a custom Virtual Network Computing (VNC) client which connects to a Docker container running a full desktop Linux operating system through the usage of a reverse proxy. Each of the components mentioned above is discussed in detail through sections 1.3.1 and 1.3.4. The relation between each of the modules is shown in Figure 1.
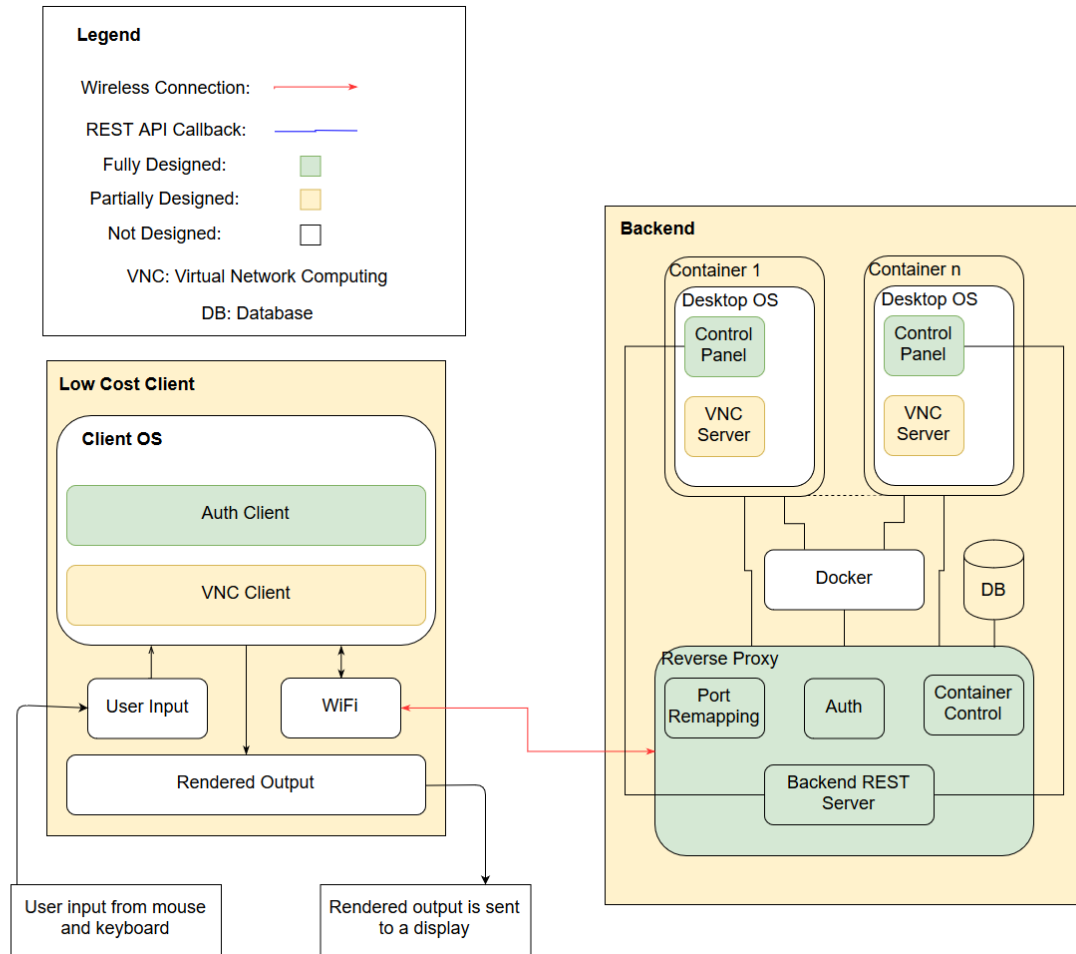
Figure 1 - Design Overview

1.3.1 Low Cost Client

The low-cost client is implemented through the means of a low-cost microcontroller. The low-cost microcontroller is used as a connectivity medium to the backend system. It runs a lightweight Linux based distribution. On the operating system, an authentication client and a custom designed VNC client are implemented. The microcontroller runs an instance of a custom designed VNC client, allowing for the streaming of the operating system to the device. The microcontroller connects to a low-cost monitor, has keyboard and mouse input/output capabilities, networking capabilities, and may support running on a battery or a direct power source.

1.3.2 VNC Client and Server

Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the remote frame buffer (RFB) protocol, developed by *Olivetti & Oracle Research Lab*, to control a computer [2]. This is implemented using a simple client-server configuration. The VNC clients and servers are implemented using *libVNC* [3].

4

The OS running on the client device runs a simple VNC Client. This client oversees the rendering of the desktop from the container in the backend. The client must also catch all input actions from the client, including mouse clicks, keyboard clicks, touch input (non-essential).

The VNC server runs in the container that runs the chosen Linux distribution. This is the actual desktop environment that the student/professor uses. The server is responsible for catching any changes to the display and sending the event data using the RFB protocol to the client for rendering. The server also receives the input from the VNC client and passes that to the currently active application.

1.3.3 Backend System

The backend system consists of four main components which include a reverse proxy, OS containers, a UWSGI (Web Server Gateway Interface) control server and a database server.

The reverse proxy allows for packet forwarding between the client, and the assigned Docker container running the desktop operating system. The reverse proxy uses a dynamically modified IP table to allow for differential packet forwarding. The reverse proxy also has the ability of dynamic scaling. If a new server is added and registered with the reverse proxy module, it can spin up new containers on multiple servers in a distributed manner.

The database server runs an instance of MySQL with proper schemas in place allowing for functionality such as adding/revoking students from the system, storing session information, application access control on student OS containers, and miscellaneous client information.

The containers run a non-resource intensive desktop Linux OS that has a GUI. This ensures that a large quantity of these OS containers can be spun up on one backend machine. The teacher's OS container runs a special version of an application developed using Electron (referred to as Control Panel in Figure 1, which allows them to have additional functionality such as controlling student's screens, sharing their own screens, revoking/enabling application access for students, revoking or enabling system access for new or existing students, and more. The backend system is easy to set up, such that it can be enabled quickly on a teacher's computer or in a server.

The purpose of the control server is to allow the teacher's client to control the student client sessions. The communication between client and server is done using REST API callbacks. For example, the teacher's client could send a POST request to the server with a specific student ID and a list of available applications. The server would then inform the respective client of the updated permissions using a second REST call. Finally, a response is returned to the teacher's client showing that the call was successful.

1.3.4 Frontend Desktop Application

The front-end portion of this project consists of an interactive desktop application that is built using Electron. This is referred to as the *Control Panel* in the block diagram. This application has a simple yet powerful user interface (UI). This is implemented using a Bootstrap template which consists boilerplate code for the HTML, JavaScript and CSS components of this application. The UI mainly consists of two sets of dropdowns, the first one being a list of students currently enrolled in the class and the second being the list of applications each student has access to. In addition, teachers are also be able to view

personalized information and data about the class. The primary user of this front-end system in this context is the teacher. Once the teacher wishes to give a student access to a specific application, he/she clicks the submit button after which the desktop application calls API endpoints on UWSGI Control Server sitting in the backend. School administrators are also able to use this application to add or delete students.

2.0 Project Specifications

The project specifications are classified into two broad categories, functional and non-functional specifications. The functional specifications refer to the functions that the project is required to implement; these are listed in Table 1. The non-functional specifications describe the characteristics of the product; these are listed in Table 2. Each specification is further sub-categorized into 2 categories, essential or non-essential components. This is based on the importance of the specifications, in order to meet the project prototype objective. Essential specifications are those which must be met for the project prototype to be satisfactory, whereas the non-essential specifications make the final prototype more than just complete when they are met. In other words, these specifications are considered to be the non-critical aspects of the system. The system still functions properly even if the non-essential specifications are not met. Each specification is rated as essential or non-essential based on their importance to the overall design.

2.1 Functional Specifications

Table 1 – Functional Specifications

Specification	Description	Classification
Latency	The user should be able to interact with the desktop operating system with a latency that does not exceed 300ms.	Essential
Teacher Control	Teachers should be able to see what is on the students' screens, as well as revoke access to certain applications.	Essential
Authentication System	The client-server setup requires an authentication system.	Essential
Network Connection	We should follow the TCP/IP protocols correctly so that packets can be transferred between the server and many clients.	Essential
Performance	The desktop operating system should be able to run browsers and other compatible applications with reasonable reactivity.	Essential
Student Birds Eye View	The teacher should be able to see the screens of all the students in one convenient location in the control panel application.	Non-essential
Secure tunnel for data transfer	The VNC client and server use an encrypted tunnel to transfer data packets.	Non-essential

2.2 Non-Functional Specifications

Table 2 – Non-Functional Specifications

Specification	Description	Classification
Cost of Device	The cost of a single educational device should not exceed $75.	Essential
Reliability	The system should transfer data from the backend systems to the Raspberry Pi without any interruptions in a reliable fashion.	Essential
Multi-User Support	The central node should be able to support at least 30 systems (Teachers and students)	Essential
Backend server uptime	The servers should be up for the duration of the school day.	Essential
Portability	The dumb terminals should be portable with a touchscreen display.	Non-essential
Security	The student and teacher systems should be reasonably secure from malicious attacks.	Essential
Usability	The desktop application should be well polished and user friendly.	Essential
Heterogeneity	The application is developed in such a way that it can only support multiple operating systems and platforms.	Non-essential

3.0 Detailed Design

3.1 Low-Cost Single Board Computer

One of the main objectives of this project is to use an inexpensive computer and provide access to high performance software and hardware to run powerful applications without the worry of updating client devices every few years. Since cost is the primary consideration factor for the project, this is one of the main factors considered when choosing the single board computer for the thin client. Other important factors to consider while deciding on the low-cost computer include determining an operating system (OS) to run on that single board computer. In this case, a compatible single board computer is required. Furthermore, other factors in regard to what devices are required to be connected with the selected single board computer, and what ports required largely include microSD slots, USB ports, and GPIO headers are very common. Based on the block diagram in terms of connectivity, HDMI, Wi-Fi and USB connectivity are the major requirements. In terms of OS, a Linux based OS is considered as the best option because of it being free in addition to the fact that there are a lot of resources, documentation and third-party integration tools it supports. Additionally, the electron application can also be developed for a Linux based OS.

The top 2 options which satisfy both low-cost single board computers and the ability to run Linux based OS' include Raspberry Pi Zero W and Orange Prime Zero. Table 3 below presents the comparison between the Raspberry Pi Zero W (Wireless) and the Orange Pi Zero 512MB on various parameters. Both have comparable CPU clock speeds and the same quantity of built-in ram. While Orange Pi Zero is $1 cheaper

in addition to having a better CPU than Raspberry Pi Zero, it lacks HDMI support, Bluetooth support to able to connect an external screen and even a Bluetooth supported mouse/keyboard respectively. Another factor that supports Raspberry Pi Zero W is that there are a lot of resources, documentation and third-party integration tools which support it when compared to the orange Pi Zero W. Both options provide the ability to attach external storage using a Micro SD card slot [4] [5]. Based on the analysis Raspberry PI Zero W is the best option for low-cost computer for the project.

Table 3 - Comparing Raspberry Pi Zero W with Orange Pi Zero *[4]*

Parameters	Raspberry Pi Zero W (Wireless)	Orange Pi Zero 512MB
SoC	Broadcom BCM2835	AllWinner H2+
CPU	ARM11 (32-bit) 1GHz single core	ARM Cortex-A7 (32-bit) 1.2GHz quad core
GPU	VideoCore IV	Mali-400 MP2
Built-in RAM	512 MB	512 MB
Removable storage	Yes	Yes
USB Hosts	1 (v2.0)	3 (v2.0)
HDMI	Yes (Mini-HDMI port (1080p60 video output)	No
Camera interfaces	Yes	No
HDMI audio	Yes	No
I2S	Yes	No
Wi-Fi	802.11 b/g/n	802.11 b/g/n
Bluetooth	Bluetooth 4.1 BLE	No
Price	USD $10	USD $9

3.2 VNC Client and Server

In order to leverage the power of virtualization and thin client devices, a remote desktop protocol is required to stream the desktop from the virtual docker container to the low-cost thin client. Three main alternatives considered are remote desktop protocol, X11 forwarding and VNC (Virtual Network Computing using Remote Frame Buffer protocol).

Remote desktop protocol is a proprietary protocol developed by Microsoft. The host machine sends a description of the window and how to render the image to the client machine, then the client machine is responsible for rendering an image and displaying it. Due to the knowledge that the client has of the end operating system, simple actions can be rendered instantly on the client without sending updates to the host and waiting for a response. While a remote desktop protocol may have better reactivity compared to the alternatives, its proprietary nature offers little to no customizability. Additionally, while the client is available on many platforms, the server comes built-in on only windows based operating systems. This severely limits the choice of desktop operating systems available to us to run on the docker container [6].

X11 (X Window System) is a windowing system for bitmap displays commonly used on Unix-like operating systems. This is implemented as a client-server model where the client can run on a separate device connected on the network [7]. The data is transferred using an SSH tunnel. X11's network protocol is

based on X command primitives. This means it sends X commands to the client to render rather than sending a rendered image. Due to this reason, the X11 solution uses a lower network bandwidth in comparison to remote desktop protocol and VNC [6]. However, this also forces the client to perform rendering actions rather than the server.

VNC (virtual network computing) uses the remote frame buffer protocol for providing remote access to graphical user interfaces. Since this protocol works at the framebuffer level it is compatible with all windowing operating systems (macOS, Windows and X Window System on Linux). The protocol operates by sending a static image of the desktop from the server to the client. The client sends all IO (keyboard/mouse inputs) as bit updates while sending packets to the server. The images being transferred use compression to reduce bandwidth usage and reduce time spent transferring data on the network. Due to the large number of customizability options, reduced CPU load on the client (rendering performed on server) and higher general performance, VNC is the method chosen for remote desktop.

libVNC provides open-source cross platform C libraries that allow implementation of a custom VNC server and client. This is necessary to support additional features such as viewing all the students' screens concurrently. *libVNC* also supports the use of AES encryption while transferring frame updates over the network. This ensures security of data during transfer as the RFB protocol isn't inherently secure [3].

VNC sends updates as messages. The client can send 6 types of messages [8]. The three messages most commonly used in the context of this project are the *FramebufferUpdateRequest*, *KeyEvent* and *PointerEvent*. Table 4 below shows what each message implies, and the parameters sent.

Table 4 – VNC Client API

Message	Description	Arguments (Datatype)
FramebufferUpdateRequest	Notifies the server to send an update of some area in the frame-buffer.	'0x3': 1 (U8) incremental: 1 (U8) x: 2 (U16) y: 2 (U16) width: 2 (U16) height: 2 (U16)
KeyEvent	Indicates key press/release. Press/release are determined by the down-flag, and the key is specified using the *keysym* values defined by the X Windows system.	'0x4': 1 (U8) down-flag: 1 (U8) pad: 2 key: 4 (U32)
PointerEvent	Indicates pointer movement or button press/release, via position and 8-bit button mask (for 8 buttons: left, middle, right, wheel upward scroll etc.)	'0x5': 1 (U8) button-mask: 1 (U8) x: 2 (U16) y: 2 (U16)

The server can send 4 types of messages as specified in the RFC spec sheet [8]. The most commonly sent message is the *FramebufferUpdate*. This is sent in response to *FramebufferUpdateRequest* message by

the client. In order to improve latency, several update requests can be processed and returned in a single frame buffer update. This behavior is set by the incremental flag the update requests.

Table 5 – VNC Server API

Message	Description	Arguments (Datatype)
FramebufferUpdate	A sequence of rectangles sent in response to the client's *FramebufferUpdateRequest*.	**Header**: '0x0': 1 (U8) pad: 1 #rectangles: 2 (U16) **#rectangle data**: x: 2 (U16) y: 2 (U16) width: 2 (U16) height: 2 (U16) encoding-type: 4 (S32)

3.3 Reverse Proxy

Within the system, to enable functionality such as teacher control, student birds eye view, connecting client devices to specific Docker containers running desktop OS's with a GUI setup, etc., there is a need for a reverse proxy. The reverse proxy sits in front of key servers in the application allowing benefits (depending on implementation) such as load balancing, protection from attacks, caching, etc. The reverse proxy consists of two key components which includes iptables and a uWSGI server.

3.3.1 Iptables

iptables is an administrative tool that uses IPV4 packet filtering and network address translation (NAT). The Linux kernel contains tables with IP packet filter rules, which *iptables* allows control of in terms of setting up, maintenance, inspection, and more. The benefit of this functionality for StreamingOS is that it allows for the configuration of network address translation rules [9].
Network address translation refers to the modification of the source/destination addresses of IP packets as they travel through a router or firewall. The importance of this is related to how packets are exchanged between the OS Docker container and the client device [10].

Each user has their own device and OS Docker container assigned to them. The traffic/packet exchange between user devices as well as the OS Docker containers are established through the setup of specific iptables rules before attempting VNC connection between the devices and OS Docker containers.

The iptables rules used for packet exchange between the client devices and OS Docker containers consist of pre-routing/post-routing rules. Pre-routing refers to the alteration of packets as soon as they come into the reverse proxy. Post-routing rules refer to the alteration of packets right before they leave the reverse proxy server. A more in-depth diagram of the described flow is shown in Figure 2 [9] [10].
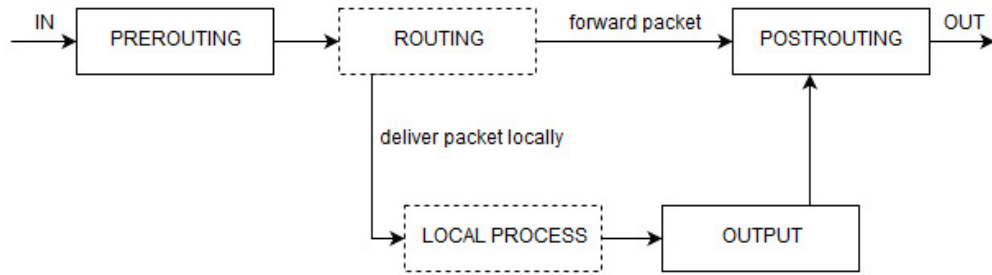
Figure 2 – *iptables* NAT Flow [9]

The connection flow between client devices and OS Docker containers for VNC connections is as follows. After the user has been successfully authenticated, the client device first makes a REST API call to the reverse proxy server providing it with the client IP address. The reverse proxy server then polls the database to find a free OS Docker container which the client device can connect to. Once a free OS Docker container is found, the Reverse Proxy server obtains that container's IP address, and sets up pre-routing/post-routing rules. All packet traffic between the client device and the OS Docker container flow through the reverse proxy (through specifically allocated ports for that communication). Once the connection/session between the device and OS Docker container is complete, the same *iptables* rules (used to setup connection) need to be replayed with a –D flag applied, which stands for delete [9]. Additionally, the reverse proxy server makes calls to the database to clean up the session information and set the OS Docker container status to be free to use. This ensures that if any attempts for reconnection are made by the client device (to the old OS Docker container), they fail and a new connection to a randomly available OS Docker container is established.

To enable the functionality for teachers to control/view students' screens, *iptables* rules are setup between the teacher's device, and the student(s) OS Docker container for packet exchange. This ensures that the student can still access and control their container, but it also provides the teacher the ability to take over/inspect the students' OS container as deemed necessary.

Additionally, the teacher can grant/revoke access for students using the system. The preliminary setup for this functionality is done at the database level during student authentication. However, removing student access during a session in progress is done at the reverse proxy level. Anytime a teacher wants to terminate a student(s) session, the corresponding iptables routes for that student(s) is all that needs to be deleted. The teacher's UI makes REST API call(s) to the reverse proxy server, providing it with a list of student ID(s) that need to have their session(s) terminated. In turn, the reverse proxy does a lookup in the database for session information and replays the corresponding iptables rules with the –D flag set to delete those rules. This breaks the connection between the client device and OS Docker container. Additionally, database level flags are also set for those students whose access is revoked to ensure that any subsequent calls for connection to an OS Docker container made by those students fail. To enable access again for those students, the corresponding flags in the database need to be updated.

3.3.2 uWSGI

The core reverse proxy server is written as a uWSGI application. uWSGI is a lightweight webserver that excels in relation to low CPU usage, and concurrency performance compared to existing alternatives. This information is described more in-depth in Section 3.4.1 of this report. The uWSGI reverse proxy server sits in-front of the existing OS Docker containers as shown in Figure 2, and service requests are made from client devices by making/invoking iptables rules [11]. To enable communication between the uWSGI reverse proxy server and client devices, specific REST API endpoints are set up (with the corresponding logic) on the server. In turn, the client devices can make REST API calls to those endpoints for functionality such as revoking student access, establishing a new VNC connection, sharing the teacher's screen, and more. Additionally, the Reverse Proxy server also has access to the database server, so that information such as existing sessions, available OS Docker containers, terminated sessions, etc. can readily be stored and accessed.

3.4 Backend Server and API Design

The system requires the use of a server located in the backend that controls access to the applications on the student devices and allows the teacher/instructor to change the access rights of certain students for certain applications. The server uses the REST API protocol for communication between the student devices and the teacher's device. The permission data for students is stored in a database.

3.4.1 Server Architecture

In order to meet the functional and non-functional specifications that have described in the Project Specifications and Risk Assessment document, the chosen server that has the ability to meet these specifications. One of the risks which are considered in the Risk Assessment is the system integration of the backend and frontend. The time taken to integrate the server into the rest of the system is also considered. uWSGI meets this criterion for integration because of its ease of installation. As uWSGI is a web server that has been written for Python, installing it on virtualized containers is can be done using pip as follows, "s*udo pip install uwsg*i" [12] .

A major functional specification that is aimed for in regards to design is the performance of the system. An important benefit of uWSGI is that it is lightweight and does not overload other web server processes (Tippets, 2013). As Figure 3 shows, uWSGI has the lowest CPU usage compared to other web servers that are available.
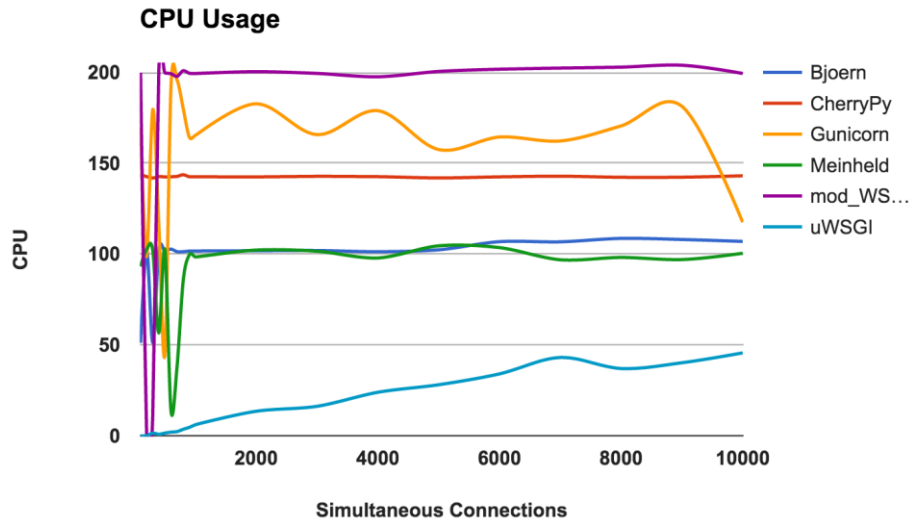
Figure 3 – CPU Usage Comparison [13]

In this test, each web server was run on 2 CPU cores, so the performance can get up to 200%, on the y-axis [13]. On the x-axis is the number of concurrent connections (or students, in this case), that are using the device. For all levels of concurrency from 0 to 10,000 users, uWSGI has the lowest CPU usage compared to other common web servers. This is one reason that uWSGI is the best option for this project.

Another comparison criterion is the ability to handle multiple requests at once. This is important because the main use case for this product is a roomful of approximately 20-50 students using the devices at once sending requests to a single server. Thus, a server a server needs to be chosen that can perform well for a relatively low number of concurrent requests. Figure 4 shows the number of requests that uWSGI and Nginx can handle concurrently.
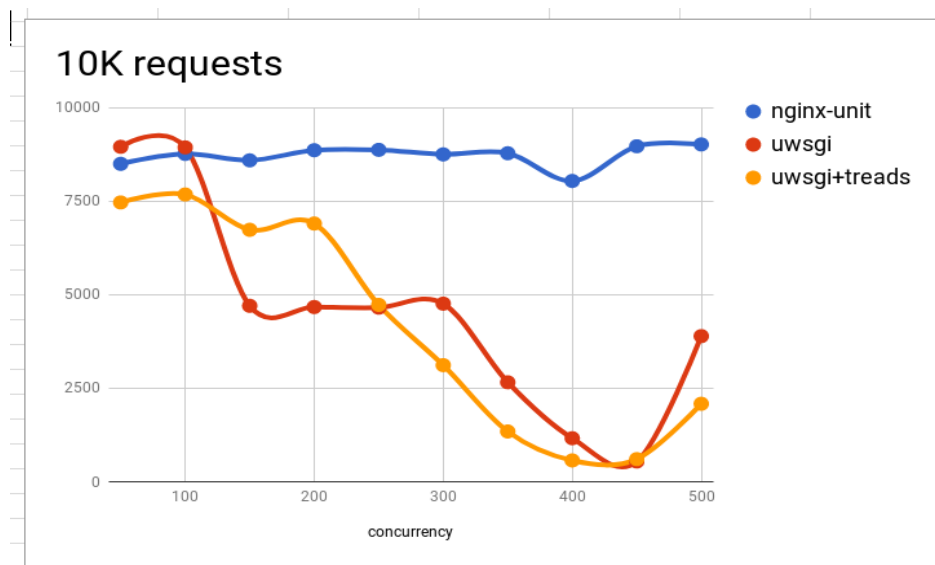

Figure 4 – Concurrency Performance Comparison [14]

As shown in red, uWSGI can handle the greatest number of total requests per second for fewer devices running concurrently. Since the target number of students using the devices is between 20-50 at a time, uWSGI is the best choice for classroom use. While it can be seen that Nginx performs better for more concurrent devices, it is out of the scope of the requirements.

3.4.2 API Design

Another important essential non-functional specification in this project is the security of the system. The REST architecture uses many different HTTP methods through which communication occurs between client and server, but there are a few methods that are common to almost every implementation of the REST API. These common methods include GET, HEAD, OPTIONS, PUT, DELETE and POST. Some of these verbs modify information on the server, and others are read-only. This is the definition of a safe method – one that does not alter the state of the server [15]. The requests that a user makes are representative of their communication with the Internet, and as such should be implemented carefully. For this reason, as many requests as possible should be implemented with safe methods such as GET or HEAD. If the REST API were to be hacked, less information could be tampered with by hackers.

Due to this the design of the REST API, that is implemented makes use of as many GET API calls as possible. The API endpoint shows the URL which the client connects with to access the API. In addition to the URL, other parameters and information can be sent to the server through the request body, which is often appended to the request URL after a question mark. An example of this is as follows, "/students/create?email=test@gmail.com&student_name="John Smith"". Table 6 details the REST API calls that are currently in this design.

Table 6 – API Endpoints

API Endpoint	HTTP Method	Description	Minimal Payload
/students/<studentID>/revoke	POST	Revoke access for one student to one or more apps, with the appIDs provided in the body.	Nothing
/students/<studentID>/add	POST	Gives access for one student to one or more apps, with the appIDs provided in the body.	Nothing
/students/create	PUT	Creates a new user. Optional body parameters include email address and name	studentID
/students/<studentID>/delete	DELETE	Remove a user from the system	Nothing
/terminal/port/<portIPAddress>	GET	Client requests a port from the server, and sends it's IP address <portIPaddress>	terminalPort

/student/<studentID>/auth	GET	User authentication call. A hash of the password is passed in through the body	Nothing
/terminal/teacher	GET	The terminal gets the video feed from the teacher's computer.	Nothing

3.5 Front End Control Panel

The main objective of the front-end portion of this project is to facilitate the display of class statistics and information to the teacher and to allow the responsible administrative staff of the school to add/remove students from each class as per enrolment criteria. However, the primary user of this system in this context is the teacher. Therefore, the control panel is benched primarily in the teacher's desktop system. In order to achieve this goal, a web application that supports the various user interface components used in the modern-day world such as dropdowns, graphs, pie charts etc. is developed. These components are used to display a dropdown list of the various students currently enrolled in the class to the teacher. In addition to that, the teacher is also able to view a list of applications each student currently has access to. This has been documented in Figure 5 which shows dropdowns indicating the names of each student, along with their student IDs and a list of applications the teacher can give the student access to. The figure also illustrates which applications the student has access to as of now (as indicated by green radio buttons) and which applications the student does not have any access to (as indicated by red radio buttons). The teacher is also able to view personalized information about the class such as the number of students enrolled, the overall average of the class in the last quiz, the performance of each student in a recent midterm, etc. The teacher is also able to remove access of certain applications to students, as they deem appropriate. This has been shown in the figure through the "Revoke Access" button. The workflow of this application is that once the teacher wishes to give a student access to a specific application, he/she clicks the "Give Access" button after which the desktop application calls certain API endpoints on the uWSGI Control Server sitting in the backend. The control panel would have an instance running in each student's container to facilitate the smooth flow of permissions to the backend component(s). However, the student would neither be able to see nor access the user interface of this web application. The web application is mainly written in CSS, HTML and JavaScript.
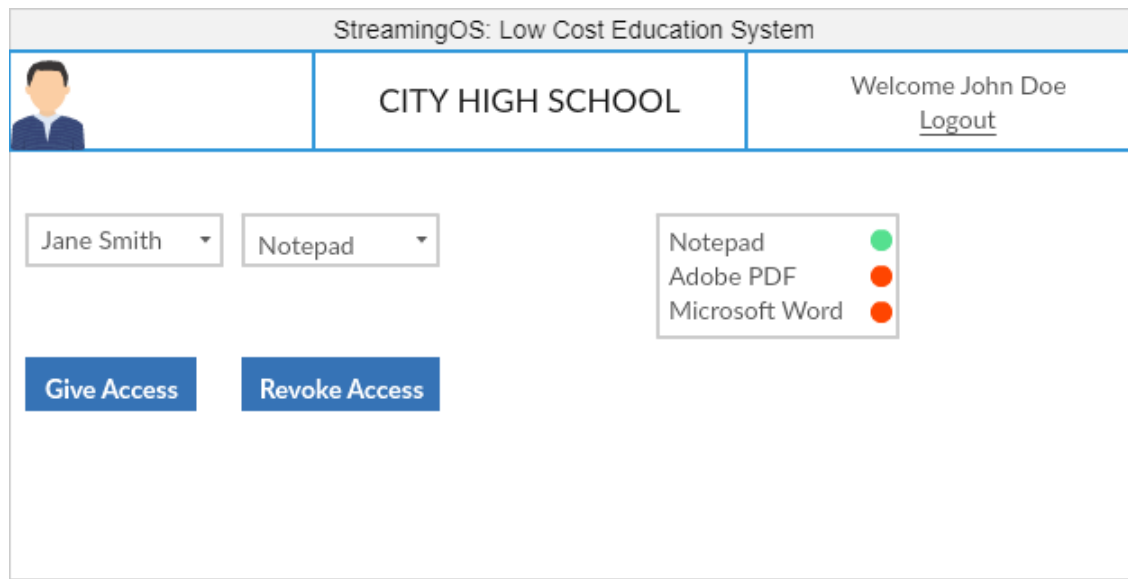
Figure 5 – Mockup User Interface of the desktop application

It is extremely important to choose the correct framework while building any web application since this choice determines the security, performance, extensibility and scalability of the overall system. If the web application is slow and is not able to stream data flowing at a high rate, then the system is deemed to have a poor performance. If the web application can get easily hacked by attackers, then the system is deemed insecure and unsafe to use. Therefore, it is very important to make the right choice at this stage of the project development phase. There are essentially two choices of frameworks that match the requirement for this project while allowing for the development in the languages that are preferred. The first choice is Progressive Web Applications (PWA) while the second option is Electron. For this project, security and performance are the two primary criterions and therefore, have been given the maximum weightage in the decision matrix in Table 7.

It is important to note here that PWA is totally different from native websites. PWA refers to browser-based web applications that are built using a collection of technologies which allow the application to be responsive and offline-capable [16]. PWA's work in any browser such as Firefox, Safari, Chrome or Edge but features such as offline connectivity depend on the browser support. Electron, on the other hand is a platform for building cross-platform desktop applications using HTML, JavaScript and other native code [16]. This makes Electron a full-blown native desktop application development environment. Electron applications live in the user-space with their own rendering engine. They do not require the user to install a browser. Firstly, it is important to consider the performance aspects of both the options. This essentially boils down to a web browser vs desktop application scenario. It is important to consider the fact that each browser has its own overhead and latency which can potentially slow down the containers running in the inexpensive hardware devices. It is expected that the front-end application is as smooth, fast and efficient as possible. It is preferable to avoid the overhead costs that are incurred by the maintenance of browser-based web applications. Due to this, Electron is seen as the ideal choice when comparing performance.
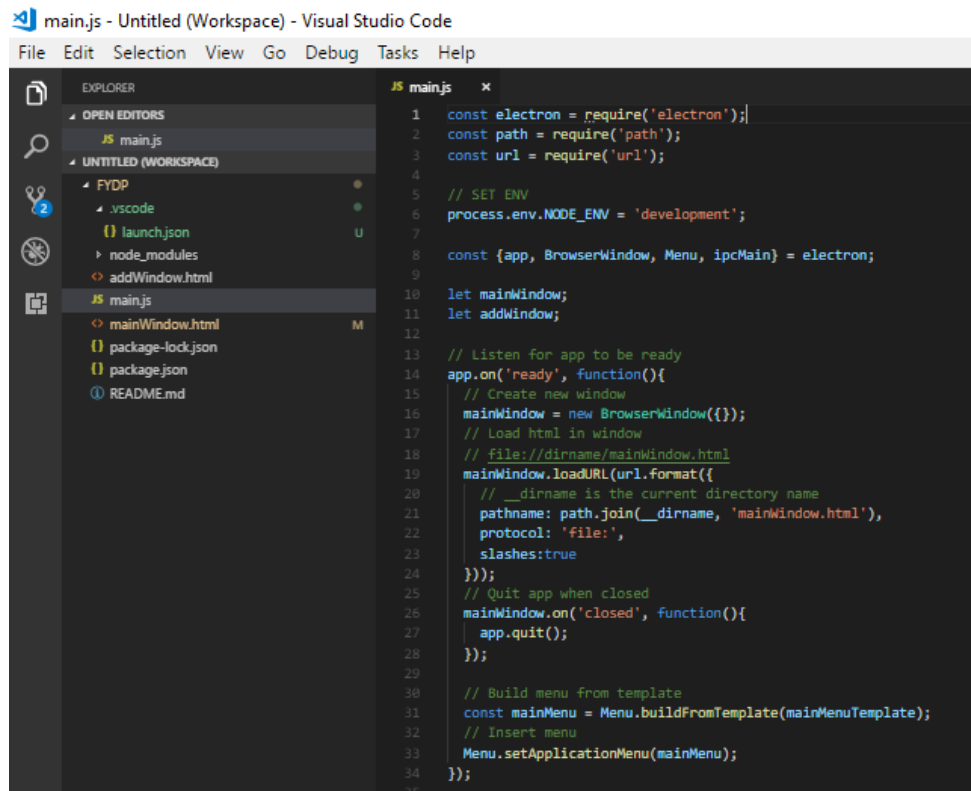
In regard to the security of the overall system, a desktop application reduces the risk of an attack over the internet as in the case of a browser such as a phishing attack. This also makes it almost impossible for any attacker to launch a Denial of Service (DOS) attack or a SQL injection attack on this application. Admittedly, even though cross-site scripting attacks are possible in an Electron application since they run on a Chromium rendering engine, Electron applications can be run offline which is not true for PWAs. Therefore, security wise, cross site scripting attacks like the ones mentioned above affect a browser-based application more fatally.

PWAs do not require any installation, and latest updates are almost instantaneously received when the user refreshes the website. On the other hand, it can be difficult to automatically update a desktop application built using Electron [16]. Electron applications have a huge footprint and consume at least 45 MB of space even for a simple" Hello World" application while PWAs do not [16]. However, this does not really affect this project since Raspberry Pi supports expandable storage. Electron, in general also has many developer tools for testing and debugging. PWAs are also flexible as they can be run on any operating system as long as they support some sort of browser [16]. This is not true for Electron applications yet as not all operating systems support Electron. This data is summarized in a decision matrix given in Table 7 which suggests that Electron is the clear emerging winner.

Table 7 – Decision Matrix for the two frameworks (Electron vs PWA)

	Comparison Criteria	Weight	Unweighted Score		Weighted Score	
			Electron	PWA	Electron	PWA
1	Performance	30	0.9	0.7	27	21
2	Security	30	0.7	0.4	21	12
3	Installation and Updates	20	0.6	0.8	12	16
4	Application Size	20	0.5	0.8	10	16
5	Testing and Debugging Support	20	0.9	0.5	18	10
6	Flexibility	10	0.7	0.9	7	9
	Total	130			95	84

Figure 6 shows how to create an instance of a barebones desktop application using Electron in Visual Studio Code (the IDE being used). It is important to note that line 6 in the code snippet given below specifies the current environment as "Development" as of now for simulation purposes. In the future, it is anticipated that this will change to "Production" when the application will be running on a real production level machine.

Figure 6 – Screenshot of the UI code to load HTML to the desktop application using Electron

3.6 Database for the Reverse proxy and Authentication

StreamingOS requires a way to store data related to every user. This data ranges from credentials required to authenticate a user into the system, information regarding the virtual machine that's been assigned to each user, and much more. Consequently, a database becomes an essential component of this system. The following sections dive into an analysis describing the design decisions made in order to formulate the database design best suited for this subsystem.

3.6.1 SQL (RDBMS) vs NoSQL

In the database and data warehouse world it comes down to two types of structures for designing a database. The options are either SQL (Structured Query language) an RDBMS (Relational Database Management System) or NoSQL, a non-relational database system. The differences between the two options are rooted in the way they are designed, types of data each supports, and the way each type stores and represents the data.

A relational database is modelled around relationally structured entities, which represents an object in the real-world; for example, a person or an online store catalog whereas a non-relational database (NoSQL) is a document-structured and distributed, holding information in a folder-like hierarchy storing data in an unstructured format [17] [18]. There are many variations of NoSQL databases ranging from key-value pairs to object-oriented designs, or even graph data structures for implementing the database [19]. The two types are known as SQL for relational database systems, and database dependent languages for

18

non-relational database systems [20]. Relations are regarded as sets in mathematics, each containing certain attributes collectively representing the information or data in SQL terms. SQL is a special purpose programming language for RDBMS databases, and is used to access/modify RDBMS data. Since this is a traditional system, which has already been widely adopted by the industry, there exists a wide array of SQL servers that implement these principles and allow for easy usage for most users who wish to use a RDBMS database [21].

In terms of flexibility and schema, an RDBMS uses a fixed schema. This means information cannot be stored into a database and then changed to contain new information, meaning that the columns must be decided and locked before any data entry. Additionally, each row must contain data for each column unless specified to handle null values. This can be amended but any kind of change would affect the whole database and can be only done offline. A NoSQL database uses a dynamic schema, which would allow the information to be added whenever required and any extra information can be stored and not even comply with the existing schema [19]. In terms of scalability, an RDBMS (SQL database) scales vertically which means that more data leads to a bigger server and storing data across multiple severs would reduce the speed for any kind of data related operation. A NoSQL database expands or scales horizontally across multiple servers. This ability of horizontal scaling by NoSQL enables it to handle more data by simply adding new servers to increase its load capacity, whereas RDBMS databases scale vertically, requiring an increase in resources such as CPU or RAM on existing servers to handle more load. NoSQL is generally faster for simple queries, whereas RDBMS is generally faster and more robust for more complex queries. One last feature to consider is the A.C.I.D compliancy where the complete form of the property is Atomicity, Consistency, Isolation and Durability. NoSQL's advantage in term of speed and scalability comes at the cost of losing out in A.C.I.D compliancy, while the RDBMS is A.C.I.D compliant [19] [21].

After comparing the two types of the databases designs, the RDBMS model is chosen for this project due to its robustness and reliability, which is the top priority. In order to fulfill the requirement of being able to authenticate users, having access to structured data about each user, information about the virtual machine assigned to each client device, and being able to support multithreaded access (which is required if 10 users are required to be served per unique device per minute), it is necessary to have a reliable system that implements ACID properties and is able to serve large queries. The queries from the API are mostly complex and compound, requiring many joins across many different blocks of data to retrieve all the required information to serve task or display analytics. Thus, the advantages of using RDBMS are worth more when compared to the advantages of using the NoSQL database model.

3.6.2 MySQL vs Postgres

There are a lot of options in terms of systems using the RDBMS model. The top two most promising options include MySQL and PostgreSQL. MySQL is considered as the most popular open-source RDBMS based on the DB-Engine rankings since 2012. This open-source system contains many features and with the help of its popularity contains of lot of resources, documentation and third-party integration tools that support it. The main reason behind MySQL's popularity is the speed, security and reliability it provides. However, all these features are at the expense of full adherence to standard SQL. A MySQL database can be accessed through multiple separate daemon processes because a server process stands between the other applications and the database itself for greater control over the one which has access

to the database. It is designed in a way that usually allows executed queries to be placed in functions known as stored procedures and executed repeatedly. However, it does have some limitations in terms of functionality with the omission of certain SQL standards, and may not be as reliable as other RDBMS databases when it comes to certain advanced features such as references, auditing, and transactions [22].

On the other hand, PostgreSQL is a well-known largescale open source RDBMS which aims to adopt most of the SQL standards when compared to MySQL. It is different from RDBMS in that it implements advanced technology to keep it A.C.I.D. compliant while supporting concurrency where it handles multiple tasks at the same time without any loss in performance. It achieves this by not avoiding any read locks and is implemented using multi version concurrency control (MVCC) instead, which also maintains A.C.I.D. compliance. Like MySQL, PostgreSQL also supports the functionality of stored procedures. Both MySQL and PostgreSQL offer the trigger functionality which enables certain stored procedures to run when a certain condition is met, PostgreSQL provides additional advanced trigger features not supported by MySQL. It is not as popular as MySQL and as thus doesn't have a lot of documentation for new developers [22]. Table 8 below provides a comparison of MySQL and PostgreSQL on multiple features and helps in making a better decision. Based on architecture feature from table 8, MySQL implements concurrent connections by spawning a thread-per-connection which has relatively lower overhead where each thread has assigned part of memory overhead for stack space and other part of memory allocated on the heap for connection-specific buffers. On the other hand, PostgreSQL designed is based on process-per-connection which is significantly more expensive than a thread-per-connection design where forking a new process occupies more memory than spawning a new thread. Additionally, IPC (inter-process communication) is more expensive between processes than between threads [23]. Based on the concurrency feature from table 8, MySQL implements a clustered index whereas the PostgreSQL implements a heap structure. A clustered index is a table structure where rows are directly embedded inside the B-tree structure of its primary key. A heap structure is a regular table structure filled with data rows separately from indexes. With a clustered index, record lookup by primary key can be achieved with a single input-output to retrieve the entire row, whereas heap would require at least two input-outputs by following the reference. The impact can be significant as foreign key reference and joins will trigger primary key lookup, which account for vast majority of queries [24].

Table 8 - Postgres vs MySQL comparison based on features [24]

Features	MySQL 8	PostgreSQL 10
Architecture	Single Process	Multi Process
Concurrency	Multi Thread	fork(2)
Table Structure	Clustered Index	Heap
Page Compression	Transparent	TOAST
UPDATEs	In-Place / Rollback Segments	Append Only / HOT
Garbage Collection	Purge Threads	Auto-vacuum Processes
Transaction Log	REDO Log (WAL)	WAL
Replication Log	Separate (Binlog)	WAL

MySQL uses less overhead by implementing concurrent connections by spawning a thread-per-connection whereas PostgreSQL would fork multiple process. Also, the fact that MySQL structures the table using a clustered index over a heap allows to handle concurrent changes to the database faster and concurrently. MySQL is also designed in way that allows executed queries to be placed in functions known as stored procedures and executed repeatedly, which are critical technical specifications and form the foundation to store user records and virtual machines assigned to each user. Based on the analysis above, MySQL presents as the perfect candidate for the project as it can provide high speed and performance which is imperative for the project.

3.6.4 Database Design and Schema

Based on the analysis done, it is important for all the data collected to be stored for future access. To satisfy this requirement, the ER model in Figure 7 is developed. The database schema is the most important aspect of the database as it allows to design and holds metadata about users along with the network traffic data in addition to application(s) currently assigned to students by teachers in a simple yet intuitive fashion.
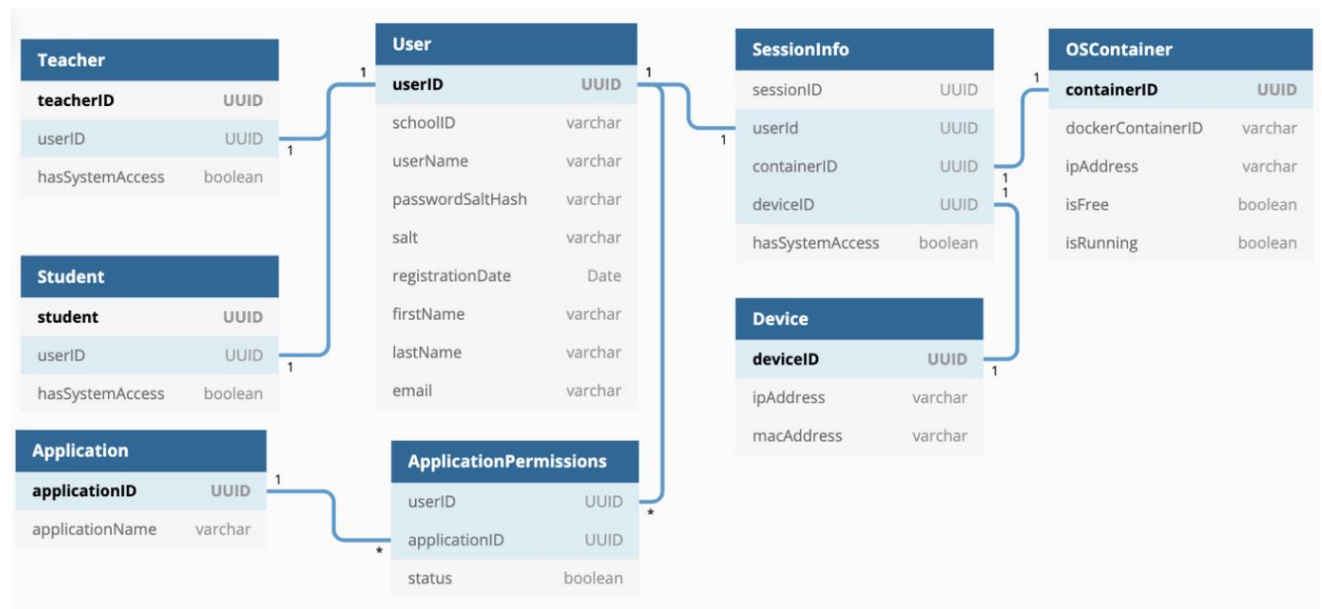

Figure 7: Database Schema Physical Entity Relationship schema

The major entities in the model are the User, Device, OSContainer, ApplicationPermissions and SessionalInfo tables from which all the user and virtual machine data can be retrieved, along with the applications assigned and accessible for each student by the teacher. The User table stores login credentials about each user type like student and teacher along with the schoolID, first name, last name and even email. A userID is assigned to uniquely identify a user. The OSContainer table is used to store the information about each docker container, their status and IP address. The containerID uniquely identifies each docker container. The SessionalInfo table is used to store and retrieve the current status of students and teachers currently using the system. Finally, the ApplicationPermissions table is used to store, retrieve, and check the applications currently assigned for a particular user.

4.0 Discussion and Project Timeline

4.1 Evaluation and Final Design

Based on the evaluation performed in Section 3, a number of decisions have been made. The REST API calls implemented are chosen to be as safe and idempotent as possible, in order to meet the security requirement. Electron is used to build the desktop application in order to meet the high-performance requirement. In addition, uWSGI is chosen to implement the web server as it is reactive with very little system performance overhead. To reduce the costs of this project, Raspberry Pi Zero W is the hardware chosen to power the thin clients.

4.2 Use of Advanced Knowledge

The project combines knowledge gained from ECE 356, ECE 358, ECE 452, ECE 454 and ECE 458. The Reverse Proxy subcomponent requires knowledge of the TCP/IP stack and how IP packets are forwarded in a typical environment. Knowledge for the development of this subcomponent relies heavily on concepts learned in ECE 358 – Computer Networking. The Database Systems (ECE 356) course outlines relational database theory, entity-relationship diagrams and schema design concepts. This knowledge is instrumental in designing the database subcomponent in a normalized fashion. General software related architecture and design decisions are made based off of concepts taught in ECE 452 - Software Design and Architecture. Knowledge of symmetric/asymmetric encryption and decryption techniques are borrowed from ECE 458 - Computer Security. This is applied in the custom designed VNC client which uses AES. The course also outlines various security vulnerabilities observed in software development. The project subcomponents actively avoid these to ensure an end-to-end secure system. This core of the project relies on leveraging distributed systems to reduce cost on endpoint client systems. The design of the project borrows heavily from concepts explained in ECE 454 – Distributed Computing, which is taught by Professor Wojciech Golab, who is the consultant for this project. The assignments in the course form the basis of the load balancing algorithm implemented to choose the next available Docker container.

4.3 Creativity, Novelty and Elegance

The traditional method of using technology in the classroom to enhance learning is to give students laptops, PC's or phones to work on. Furthermore, upgrades/replacements for these devices can be fairly expensive. While these devices are much more powerful than a Raspberry Pi Zero W, they are much more expensive than a Raspberry Pi Zero W. The chosen devices will need to be eventually replaced, as with any electronic device. However, they will be cost effective to replace and easy to install in the existing network when necessary. If the device crashes or stops working during a class, it is easy to switch it out with another one, and the issue can be troubleshooted after the class is over. In addition, since most schools already have computer labs, replacing the costly computers with these devices means that existing infrastructure in the lab, such as a mouse, keyboard for the device, power, network devices can be reused.

4.4 Student Hours

Table 9 shows how many hours each student has worked on so far for the project.

Table 9 - Number of hours invested by each team member

Student	# of Hours Invested
Anurag Joshi	55
Matthew Milne	56
Surag Sudesh	54
Vidit Soni	59
Vinayak Sharma	54

4.5 Potential Safety Hazards

As this is mainly a software project, there are few safety hazards associated it. The only risk comes with the use of the Raspberry Pi. As with all electrical circuits, there is a risk of short circuits occurring and the Raspberry Pi heating up or catching on fire. However, there is extremely unlikely, as the devices are likely well tested by the manufacturer before being sold.

4.6 Project Timeline

Due to minimal dependencies between the different subcomponents, they can be developed in parallel if a virtual test environment has been setup. Figure 8 shows the project timeline.
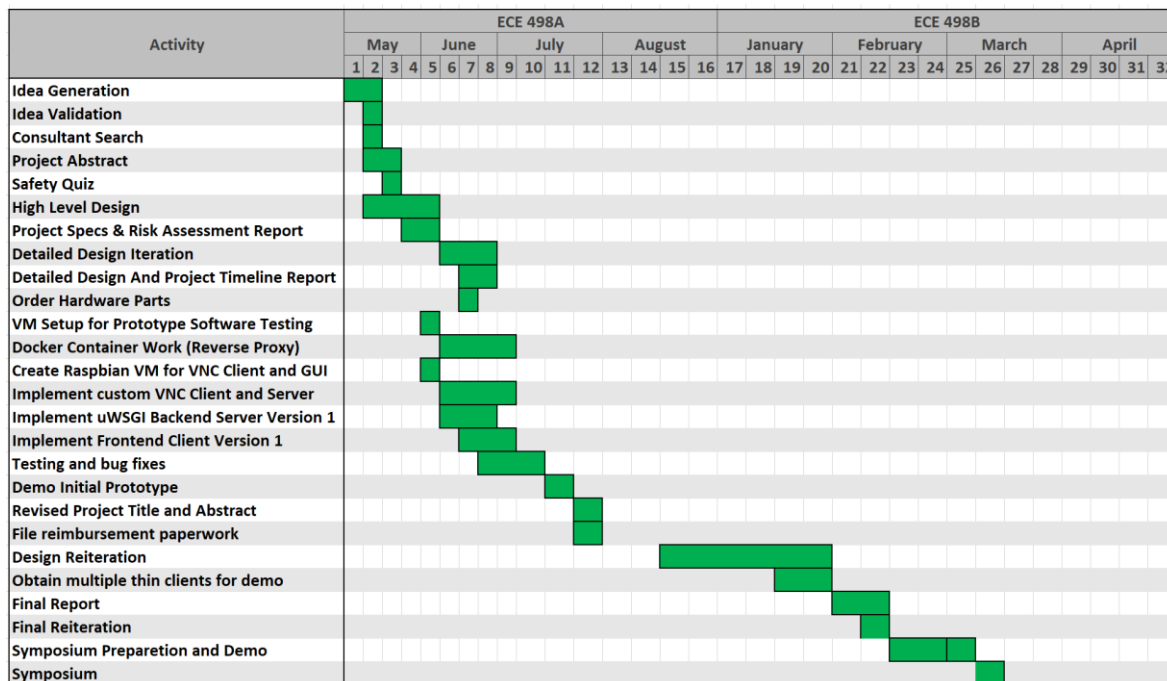

Figure 8 – Project Timeline

23

References

[1] B. B. a. R. L. Ross, "The Cost of School-Based Educational Technology Programs," 3 March 2013. [Online]. Available: https://www.rand.org/pubs/monograph_reports/MR634/index2.html. [Accessed 1 June 2019].

[2] "Virtual Network Computing," [Online]. Available: https://en.wikipedia.org/wiki/Virtual_Network_Computing. [Accessed 2 June 2019].

[3] "LibVNC," [Online]. Available: https://libvnc.github.io/.

[4] HackerBoards, "HackerBoards.com," [Online]. Available: https://www.hackerboards.com/compare/158,139/. [Accessed 26 June 2019].

[5] "Raspberry Pi Zero W (Wireless) vs. Orange Pi Zero 512MB," board db, [Online]. Available: https://www.hackerboards.com/compare/158,139/. [Accessed 23 June 2019].

[6] I. Ambanwela, "Graphical Remote Desktop Protocols RFB(VNC), RDP and x11," 23 Nov 2013. [Online]. Available: http://ishanaba.com/blog/2012/11/graphical-remote-desktop-protocols-rfbvncrdp-and-x11-2/. [Accessed 24 Jun 2019].

[7] "X Window System," Wikipedia Foundation, [Online]. Available: https://en.wikipedia.org/wiki/X_Window_System.

[8] T. Richardson and J. Levine, "The Remote Framebuffer Protocol," March 2011. [Online]. Available: https://www.ietf.org/rfc/rfc6143.txt. [Accessed 24 June 2019].

[9] H. Eychenne, "iptables(8) - Linux man page," [Online]. Available: https://linux.die.net/man/8/iptables. [Accessed 20 June 2019].

[10] K. Rupp, "NAT - Network Address Translation," [Online]. Available: https://www.karlrupp.net/en/computer/nat_tutorial. [Accessed 23 June 2019].

[11] "The uWSGI project," [Online]. Available: https://uwsgi-docs.readthedocs.io/en/latest/. [Accessed 22 June 2019].

[12] S. Tippets, "Digital Ocean," 19 August 2013. [Online]. Available: https://www.digitalocean.com/community/tutorials/django-server-comparison-the-development-server-mod_wsgi-uwsgi-and-gunicorn. [Accessed 24 June 2019].

[13] O. Habib, "AppDynamics," 11 May 2016. [Online]. Available: https://www.appdynamics.com/blog/engineering/a-performance-analysis-of-python-wsgi-servers-part-2/. [Accessed 24 June 2019].

[14] M. Gavrilov, "ITNext," 18 January 2018. [Online]. Available: https://itnext.io/performance-comparison-between-nginx-unit-and-uwsgi-python3-4511fc172a4c?gi=bb89601c22b3. [Accessed 2019 24 June].

[15] S. Allamaraju, "RESTful web services cookbook," Sebastopol, CA, OReilly, 2014.

[16] F. Rieseberg, "Progressive Web Apps & Electron," 8 April 2019. [Online]. Available: https://felixrieseberg.com/progressive-web-apps-electron/. [Accessed 23 June 2019].

[17] "Cloudflare," [Online]. Available: https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/. [Accessed 24 June 2019].

[18] "NoSQL," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/NoSQL. [Accessed 19 June 2019].

[19] E. McNulty, "SQL vs. NoSQL- What You Need to Know," DATACONOMY, [Online]. Available: https://dataconomy.com/2014/07/sql-vs-nosql-need-know/.

[20] A. Brody, "SQL Vs NoSQL: The Differences Explained," 9 March 2017. [Online]. Available: https://blog.panoply.io/sql-or-nosql-that-is-the-question. [Accessed 20 June 2019].

[21] "SQL," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/SQL. [Accessed 23 June 2019].

[22] M. Drake, "SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems," 19 March 2019. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Accessed 21 June 2019].

[23] E. Klitzke, "Uber Engineering," Uber, 26 July 2016. [Online]. Available: https://eng.uber.com/mysql-migration/. [Accessed 27 June 2019].

[24] K. Ejima, "Hackernoon," 23 May 2018. [Online]. Available: https://hackernoon.com/showdown-mysql-8-vs-postgresql-10-3fe23be5c19e. [Accessed 26 June 2019].