University of Waterloo

Faculty of Engineering

Department of Electrical and Computer Engineering

StreamingOS: Low Cost Education System

Final Report

Group 2020.15

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

Consultant:

Wojciech Golab

February 14, 2020

Abstract

As technology improves in the 21st century, using mobile devices for educational purposes is becoming more common in primary and secondary schools. In addition to the cost, this technology becomes quickly outdated and needs to be replaced, forcing schools to spend continuous amounts of money on maintenance. StreamingOS is a system that provides students and teachers with inexpensive thin endpoint devices, with the resource-heavy OS being streamed to these devices from a backend server using container virtualization. The objective of this project is to design a powerful, inexpensive device and streaming system that enhances the learning experience. StreamingOS uses an inexpensive endpoint device 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. The inexpensive hardware helps break 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.

Acknowledgements

Our team would like to acknowledge our consultant, Professor Wojciech Golab. Professor Golab provided feedback that was crucial in deciding on the target market for our product, as well as giving suggestions for improvements on some key areas of our design.

Table of Contents

List of Figures

List of Tables

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 In addition, this technology becomes quickly outdated and needs to be replaced, forcing schools to spend continuous amounts of money on maintenance.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 tablet (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 virtualized container running a full desktop Linux or Windows 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.

1.3.1 Low Cost Client

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

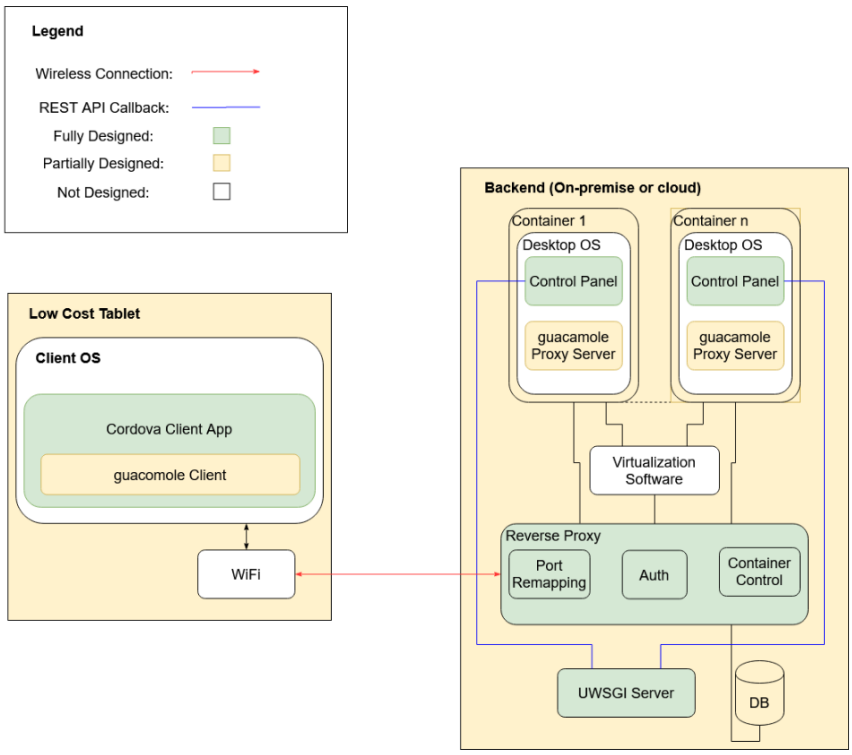

Figure 1 – Design Overview

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 client used for this system is called Guacamole. It is a JavaScript and HTML 5 based system which allows for easy embedding into web applications.

The OS running on the client device invokes the Guacamole client code with the necessary configuration parameters. This client oversees the rendering of the desktop from the container in the backend. The client must also catch all input actions from the user, including mouse clicks, keyboard clicks, and touch input. The client communicates with a VNC server in the backend through guacamole's proxy server.

There is a proxy server that acts as a middleman between the guacamole client and the VNC server. The proxy server runs on the same VM as the actual VNC server and provides full-duplex communication channels between the client and server.

The VNC server runs in the container that runs the chosen operating system to be streamed. 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 (through a proxy server) and passes that to the currently active client application.

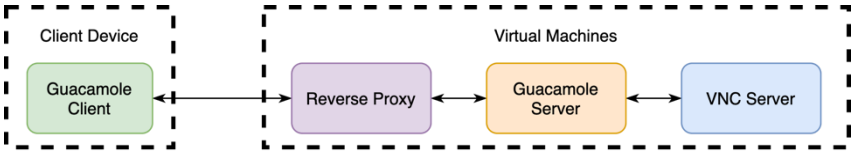

Figure 2 – Frontend Backend Communication Flow

1.3.3 Backend System

The backend system consists of five main components which includes a reverse proxy, OS containers, a database server, VNC servers, and VNC proxy servers.

The reverse proxy allows for packet forwarding between the client, and the assigned streaming containers 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 be allocated to users in a distributed manner. Additionally, the reverse proxy server also performs constant 30 second health checks on all OS containers that have established VNC connections. The VNC server is the main component that all clients talk to which orchestrates all forms of setup, cleanup, and communications between the clients and the OS containers.

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 or Windows 10. This ensures that a large quantity of these OS containers can be spun up on one backend machine while also providing students and teachers with the diverse flexibility of operating systems based on their use cases. 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 Cordova. This is referred to as the Cordova Client App in the block diagram (figure 1). This application has a simple yet powerful user interface (UI). This is implemented using a Bootstrap template which consists of boilerplate code for the HTML, JavaScript and CSS components of this application.

The UI consists of three main components. The first one is the “Students” component. The “Students” component has a set of views for each student that is currently streaming an operating system. This functionality is explained more in depth in section 3.7. Additionally, a screenshot of the Student Bird’s Eye View is shown in Figure 9, located in section 4. Another main component is the “Permissions” section. This section consists of a set of list views that gives teachers the ability to revoke and/or grant application permissions for students. Furthermore, it also provides analytics and information to teachers regarding students and their activities on the system. A screenshot of the permission section is shown in Figure 10. The third main component is the “Connect” section. If students are logged onto the system, they are met directly with this component. Teachers can access this component through the navigation menu on the left side of the system. This component has two main buttons in the center of the view. One of the buttons connects the user to a Linux OS, while the other button connects the user to a Windows OS. Figure 11 displays these two buttons. Furthermore, this view also provides information regarding the OS containers such as how many containers are currently free. When the student clicks one of these buttons, they are met with the either the Linux or Windows desktop, depending on their choice. Figure 12 shows screenshots taken from actual streams of the OS containers.

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
Student Birds Eye View	The teacher should be able to see the screens of all students actively streaming with at most a 10 second delay.	Non-essential
Secure Tunnel for Streaming	The VNC client and server use an encrypted tunnel to transfer data packets.	Non-essential
Connectivity State Management	The system should be capable of recognizing active/failed OS container states within one minute of becoming active/inactive. It should also add/remove containers in the allocation pool for users accordingly.	Essential

Application Level Permissions	The system should provide teachers with the ability to grant/revoke application permissions (on an OS container) per student.	Essential
Screen Sharing	The teacher should be capable of sharing their screen with their students.	Essential
Notifications	The teacher should be able to send notifications to students' clients. Notifications should be received within 10 seconds of being sent.	Non-essential
OS Container Isolation	The OS container should be network isolated such that only the Reverse Proxy should be able to directly communicate with the OS container.	Essential
Supported applications diversity	The OS streams should be capable of supporting a web browser and a suite of desktop productivity applications (i.e. word processor, spreadsheet software, presentation software, and lightweight IDE)	Essential

2.2 Non-Functional Specifications

Table 2 – Non-Functional Specifications

Specification	Description	Classification
Security	The system should be capable of providing of role based (teacher, student, etc.) access control.	Essential
Heterogeneity	The user facing client should support iOS, Android, Windows, and Linux based operating systems.	Non-essential
Cost-Effectiveness	The full cost of the system per user should not exceed $75 per year which is cheaper than the cost of a system with similar specifications.	Essential
Usability	The system should provide all users (students and teachers) functionalities through a single client facing application.	Essential
Portability	The device should be capable of fitting in a standard backpack (17"x9"). The weight of the device should be less than 600 grams.	Essential
Reliability	On accidental/unintended user disconnect(s), the system should be capable of re-establishing the connection to the same OS stream without losing user state (as long as an adequate internet connection exists). Re-connect should occur in less than 10 seconds upon internet connection re-establishment.	Non-essential

3.0 Detailed Design

3.1 Low-Cost Computer

One of the main objectives of this project is to provide access to high performance software and hardware using an inexpensive computer 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 a tablet for the thin client. Other important factors to consider while deciding on the low-cost computer include the device operating system (OS), the device processor, networking capabilities, USB ports and microSD slots. Based on the block diagram in terms of connectivity, Wi-Fi and USB connectivity are the major requirements. In terms of OS, a Linux based OS is considered to be 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.

In order to select the tablet best suited for client use, three products were considered: Amazon's Fire 7, Haoqin's H7 Tablet and Lenovo's Tab M7. These products were considered due to their low cost, 7-inch screen size, and availability. Additionally, other factors such as storage, networking capabilities, RAM and battery life influenced the choice of these tablets.

Table 3 below presents the comparison among three tablets which include Amazon's Fire 7, the Haoqin's H7 Tablet and Lenovo's Tab M7 on various parameters. The tablets have been mainly been selected on their cost, screen size of seven inches and its Android OS which allows for easier development of the client application. All 3 tablets have comparable CPU clock speeds of around 1.3 GHz for both the Amazon Fire 7 and Lenovo Tab 7. The Haoqin H7 has CPU clock speeds of 1.5 GHz. Another important parameter is to make sure the tablets have at least 1 GB of RAM to run the client application smoothly, and all three tablets satisfy the condition. In terms of cost, Amazon Fire 7 has the smallest cost which is one of the major factors in selecting the tablet. When comparing the tablets performance based on the processor, the Fire tablet with its latest 64-bit chip architecture outperforms the remaining 2 tablets. In terms of networking capabilities, the Fire Tablet comes ahead among the three options as it has dual bands to support faster download and upload speeds when compared to Lenovo's tablet M7. When compared to Haoqin's H7 tablet, the Fire 7 supports Bluetooth 4.1 in contrast with Haoqin's Bluetooth 2.1, which is very weak in terms of range and strength. All three tablets provide the ability to insert external storage using a Micro SD card slot, but the Fire Tablet 7 allows the maximum expansion of storage among the options to 256 GB [3] [4] [5].

Table 3 – Tablet Specification Comparison [3] [4] [5]

Parameters	Amazon's Fire 7	Haoqin H7	Lenovo's Tab M7
Device Cost (CAD)	54.99 + tax	69.99 + tax	79.99 + tax
Operating System	Android	Android	Android
Display	7" IPS display (1024 x 600)	7" IPS display (1024 x 600)	7" IPS display (1024 x 600)
SoC	MediaTek® MT8163V/B	Rockchip 3326	MediaTek® MT8321
CPU	(64-bit) 1.3 GHz Quad-Core Processor	(64-bit) 1.5 GHz Quad-Core Processor	(32-bit) 1.3 GHz Quad-Core Processor
GPU	ARM Mali-T720 MP2	ARM Mali-G31MP2	ARM Mali-400 MP2
Built-in RAM	1 GB	1 GB	1 GB
Storage	16 GB, MicroSD up to 256 GB	16 GB, MicroSD up to 64 GB	16 GB, MicroSD up to 128 GB
Input and Output (I/O)	1 USB 2.0 (micro-B) microSD card slot	1 USB 2.0 (micro-B) microSD card slot	1 USB 2.0 (micro-B) microSD card slot
Wi-Fi	802.11 a/b/g/n dual band	802.11 a/b/g/n dual band	802.11 a/b/g/n single band
Bluetooth	Bluetooth 4.1	Bluetooth 2.1	Bluetooth 4.0
Avg Battery Life	8 hours	5 hours	10 hours

Based on the analysis, Amazon's Fire 7 Tablet is the best option as a low-cost computer for the project. A key functional requirement for the chosen system was ensuring that it was portable. This requirement was that the device had to fit within a 17 inch by 9-inch backpack and weigh under 600 grams. The tablet

itself had the dimensions of 8.78 inch by 0.42 inch and weight of 286 grams which satisfies our functional requirement.

3.2 VNC Client and Server

In order to leverage the power of virtualization and tablets, a remote desktop protocol is required to stream the desktop from the OS container to the low-cost tablets. 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 for us to run on the OS 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.

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*. 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.

Guacamole is an open source HTML, CSS, and JavaScript based implementation of the standard VNC protocol. However, guacamole has its own additions that it has included on top of VNC which requires the

need to use Guacamole's proxy server in addition to a regular VNC server to facilitate communication with the Guacamole client [9]. To fulfil the functional requirement of having an encrypted end-to-end tunnel, Guacamole's feature of SSL/TLS encryption between the client and server was used.

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 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 [10]. 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 VM container and the client device [11].

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

The iptables rules used for packet exchange between the client devices and OS VM 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 3 [10] [11].

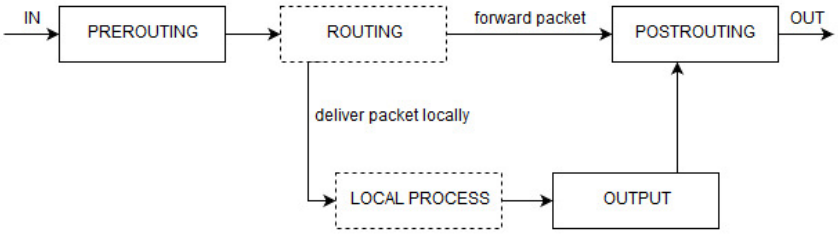

Figure 3 – iptables NAT Flow [10]

The connection flow between client devices and OS VM 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 VM 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 VM 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 [10]. Additionally, the reverse proxy server makes calls to the database to clean up the session information and set the OS VM 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 VM container), they fail and a new connection to a randomly available OS VM 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 VM 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. Relying on *iptables* and the reverse proxy server to maintain connections, allows the OS container to be isolated from the client network and only made accessible via the reverse proxy server. This satisfies the functional specification of OS container network isolation.

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 VM 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 VM 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 VM containers as shown in Figure 3, and service requests are made from client devices by making/invoking iptables rules [12]. 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 VM containers, terminated sessions, etc. can readily be stored and accessed.

3.3.3 Stream Management

A key role of the reverse proxy server is to maintain an updated view of the OS containers in the database. This facilitates OS container sharing, cleanup, and allocation amongst service users. A key functional requirement for the system is to ensure that a state change would be recognized by the system on an OS container state change within a minute of the change occurring. When allocating an OS container, the state change of the container being allocated (in the database) takes less than a second for each OS type. The largest state change delay is incurred during container clean up. When a user disconnects from the system, there is an initial 30 second delay incurred before the system attempts to clean up the users state and place the system back into the pool to be allocated to another user. This 30 second delay is expected as it gives the user enough time to reconnect in the event of a failure. Table 4 shows a few of the state change timings for OS container cleanup. These mentioned results clearly show the functional requirements being fulfilled.

Table 4 – OS Container Cleanup times

Linux (s)	Windows(s)
31.17	30.17
47.63	39.79
49.59	38.02
47.62	34.79

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" [13].

A major functional specification that is aimed for in regard 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 [13]. As Figure 4 shows, uWSGI has the lowest CPU usage compared to other web servers that are available.

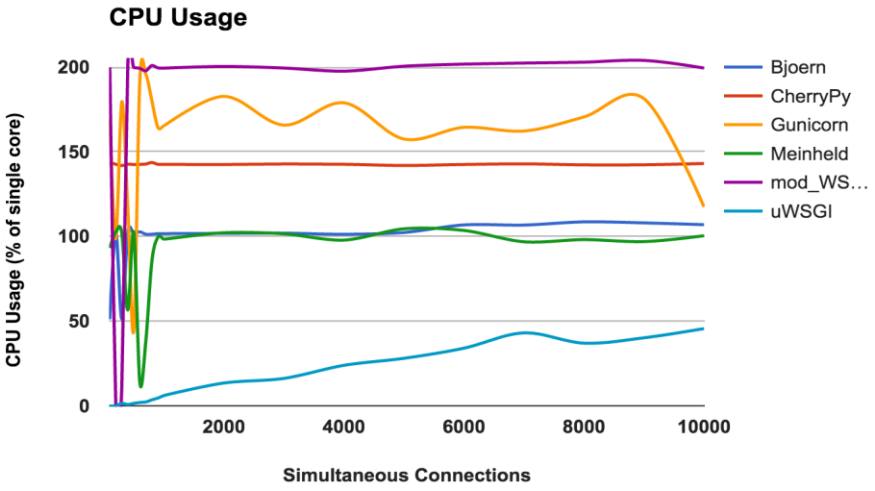

Figure 4 – CPU Usage Comparison [14]

In this test, each web server was run on 2 CPU cores, so the performance can get up to 200%, on the y-axis [14]. 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 needs to be chosen that can perform well for a relatively low number of concurrent requests. Figure 5 shows the number of requests that uWSGI and Nginx can handle concurrently.

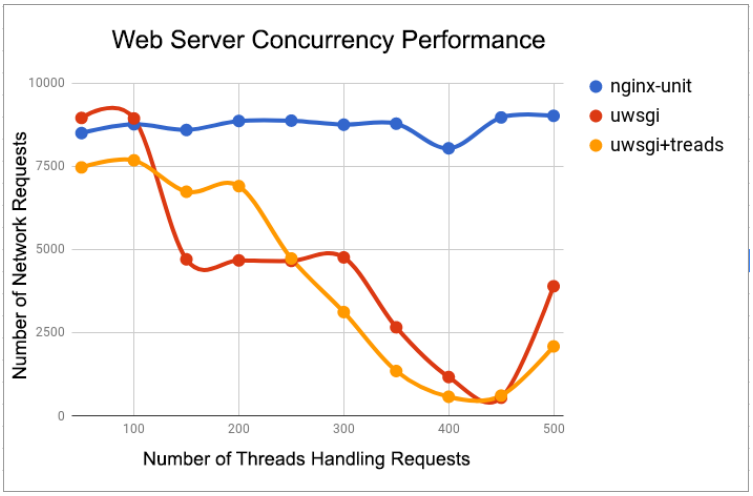

Figure 5 – Concurrency Performance Comparison [15]

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 [16]. 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 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 5 details the REST API calls that are currently in this design.

Table 5 – ReverseProxy API Endpoints

API Endpoint	HTTP Method	Description
/routes/setup/<user_id>/ <client_ip>/<os_type>/<width>/ <height>	GET	Setup routes for the OS Container, passing in userID, clientIP and desired screen resolution.
/routes/delete/<user_id>	GET	Deletes routes for the OS Container by userID
/broadcast/<user_id>	PUT	Broadcasts the teacher session to all active students using the teacher's user id
/broadcast/stop	PUT	Stops the teacher's broadcast
/setup/stream/<user_id>/ <client_ip>/<broadcast_id>	PUT	Setup routes for the view-only stream of the teacher OS Container
/subscribe	GET	Subscribe to teacher published events
/user/<username>/create/ <password>	PUT	Creates a new user. Body parameters include schoolID, name, user type, email & profession
/user/<user_id>/delete	DELETE	Removes a user from the system by userID
/user/<user_id>/screen/ snapshot	GET	Retrieves a JPEG image of <user_id>'s VNC session
/user/<username>/info	GET	Get information about a user with username <username>
/school/<school_id>/studentlist	GET	Get a list of all the students in the school specified by <school_id>
/applications	GET	Get a list of all applications supported by streamingOS
/sessions	GET	Get a list of all users that have an active session

/user/<user_id>/applications	GET	Get the list of permitted applications for a student
/user/<user_id>/revoke/ <application_id>	DELETE	Revoke access for one student to one or more apps, with the appIDs provided in the body.
/user/<user_id>/grant/ <application_id>	PUT	Gives access for one student to one or more apps, with the appIDs provided in the body.
/token	POST	Generates an OAuth 2.0 token for the user specified in the body
/availableVM	GET	Get a list of all the available VMs

3.4.3 OAuth Access Control

OAuth 2.0 is an industry-standard protocol most commonly used for Authorization. This protocol provides the ability to grant users access to resources in a system based on pre-defined roles [17]. In the case of StreamingOS, OAuth 2.0 was used to restrict user access to the Reverse Proxy server depending on what the user's role was. Roles are defined by a concept called "scopes". A scope is a mechanism which imposes limitations on what exactly a user can access [17]. In the case of streamingOS, the following scopes were created:

teacherStreamingOS: Resources that only teachers should have access to.

studentStreamingOS: Resources that only students should have access to.

studentTeacherStreamingOS: Resources that both students and teachers should have access to.

The teacherStreamingOS endpoint was only access by teachers for accessing functionality such as revoking student app permissions, granting student app permissions, broadcasting their screen, etc. The studentTeacherStreamingOS scope is introduced as a common scope to grant resource/endpoint access to both students and teachers. Common resources such as connecting to a VM, getting basic VM statistics, etc. are functionalities that both students and teachers can access. The studentStreamingOS endpoint is introduced for student-only functionality, however, in the current prototype there has been no real need for this scope yet, as all endpoints that students have access to are also accessible by teachers, meaning the studentTeacherStreamingOS scope is used for those cases.

Ultimately, the implementation of OAuth 2.0 helped fulfil the essential functional requirement of security, which required that a security mechanism be implemented which could restrict user access based on defined roles which in this case are teachers and students.

The permission server was setup (through firewall rules) in a way such that only the Reverse Proxy could communicate with it. Therefore, OAuth 2.0 was not implemented onto the permission server.

3.5 Front End Client Application

The main goal of the front-end portion of this project is to create a consolidated single application that both the students, and the teachers can use. This satisfies the non-functional specification of usability. From the teacher's standpoint, the application should facilitate the display of permission statistics and status information for each student registered in that teacher's class. The application should also let the teacher revoke and give access to specific applications to the students. This satisfies the functional

requirement of having application level permissions in our application. The teacher should be able to view up to date snapshots of each student's screen to keep a tab on the student's activities. The application should also let the teacher connect to an instance of a Virtual Machine and broadcast their session. From a student's perspective, the front-end application has two main goals. Firstly, the application should let them connect to an available Virtual Machine. Secondly, the application should let the students see what the teacher is broadcasting to them. To achieve these goals, a web application that supports the various graphical user interface components used in the modern-day world such as radio buttons, tables, sliders etc. is developed. These components are used to display a table of various students that are 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 10 which shows a table on the right-hand column displaying the names of each student 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 the "Active" text) and which applications the student does not have any access to (as indicated by red radio buttons and the "Inactive" text). 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). This application is mainly written in CSS, HTML, and JavaScript, using Bootstrap boilerplate template code. Heterogeneity is an essential non-functional specification for streamingOS. Sections 3.5.1 and 3.5.2 explain in detail how this specification is satisfied.

3.5.1 Running the Application on a Desktop

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 be 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 of developing the application for a desktop environment while allowing for 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 6.

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 [18]. 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 [18]. 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. Even though the possibility of a cross-site scripting attack in an Electron application exists due to it being rendered via a Chromium Engine, they can still be run offline which is not true for PWAs, to begin with. Therefore, security wise, cross site scripting attacks like the ones mentioned above affect a browser-based application more fatally than a framework such as Electron.

PWAs do not require any installation, and latest updates are almost instantaneously received when the user refreshes the webpage. On the other hand, it can be difficult to automatically update a desktop application built using Electron [18]. 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 [18]. However, this does not really affect this project since many modern desktops support 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 [18]. 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 6 which suggests that Electron is the clear emerging winner.

Table 6 – Decision Matrix for the Electron and PWA Frameworks

	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

3.5.2 Running the Application on a Mobile Device

Once the application is written using the preferred set of languages and is confirmed to be working on a desktop environment, it is important to choose a hybrid cross-platform framework that can compile and translate this application to run on mobile devices. The framework chosen should preserve the security, performance, extensibility, and scalability of the original system while adding extra benefits to it such as plugin and mobile hardware support. Xamarin was one of the initial options considered for this task. However, it was concluded that Xamarin works best with the C# programming language which is not something within the scope of this project. There are essentially two choices of cross-platform development frameworks that match the requirement for this project while allowing to simultaneously continue the development of the application in the languages that are preferred. The first choice is pure

Apache Cordova while the second choice is Ionic. The resultant application package would then be compiled using one of those two frameworks on the tablet. Ideally, this could be compiled on any tablet or mobile device that operates using iOS or Android. The advantage of using hybrid cross-platform frameworks such as the ones mentioned above is that they permit the deployment of a native application to multiple platforms using a single set of source code [19].

Firstly, both Cordova and Ionic use Angular directives to implement HTML components since both are built on top of AngularJS. Both frameworks allow developers to write code that accesses the native features of a mobile device such as a camera, location access, Bluetooth, etc. [26]. From a performance perspective, both Cordova and Ionic would have worked for this project since both provide high performance to the application. Even though Ionic is said to have a lower performance than Cordova due to the latter running directly on a Web View, this difference is not noticeable to an average user. From a learning point of view, it is much easier to compile an application using Cordova since it works using a JavaScript Web View [26]. Additionally, one must run only a couple of commands to get the application running on a mobile device using Cordova. Ionic, on the other hand, has a much more tedious process related to its installation and usage. Developing a simple application using Ionic forces the developer to first learn about templating, code modularity, dependency injections, and custom directives. These concepts can be challenging to learn and overwhelming at a glance, especially, if running an HTML, CSS and JavaScript application on a tablet is all that is required. Hence, as far as the learning curve is concerned, Cordova is a winner here.

Plugins are additional functionalities that a developer can integrate on top of a basic barebones Cordova or Ionic application for added features. One such example would be a keyboard-plugin that detects, and registers input key events every time the user types on their native keyboard. As far as plugin support is concerned, Ionic has an extensively larger base of supported plugins than Cordova. For example, many high-end plugins used to compile high-performance games are absent in Cordova. Cordova, on the other hand, has hundreds of plugins that support development work. In the case of this project, this is not a big concern since the application is fairly simple, straightforward and doesn't need any other high-end plugins other than the basic ones supported by both [20]. Therefore, it can be evidently said that Ionic has a much broader support to heavy-weight and sophisticated applications, as compared to Cordova. Both Cordova and Ionic have excellent documentation and strong community support, which helps ease development [20].

Cordova libraries and boilerplate code have not changed much in the past few years. However, Ionic is still considered to be in development. Due to this, Ionic's standards and support to a specific operating system and environment constantly keeps changing. It is possible that entire libraries of code are rewritten between different versions [20]. This is a huge cause of concern for this project since the idea of having to make last minute changes due to a third-party decision is not something entirely feasible. This data is summarized in a decision matrix given in Table 7 which suggests that Cordova is the clear winner.

Table 7 – Decision Matrix for the two hybrid cross platform frameworks, Cordova and Ionic

	Comparison Criteria	Weight	Unweighted Score		Weighted Score	
			Cordova	Ionic	Cordova	Ionic
1	Performance	30	0.8	0.8	24	24
2	Learning Curve	25	0.9	0.7	22.5	17.5
3	Testing and Debugging Support	20	0.9	0.9	18	18
4	Resistance to Change	35	0.8	0.6	28	21
5	Support to Heavy Weight Applications	30	0.7	0.9	21	27
6	Plugin Support	10	0.7	0.9	7	9
	Total	150			120.5	116.5

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 to the information regarding the virtual machine that's been assigned to each user. 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/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 [21] [22]. 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 [23]. The two types are known as SQL for relational database systems, and database dependent languages for non-relational database systems [24]. 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 [25].

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 only be 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 [23]. 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 [23] [25].

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, 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 [26].

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 [26]. 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 a part of memory for stack space and other parts of memory allocated on the heap for connection-specific buffers. On the other hand, PostgreSQL design 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 [27]. 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 [28].

Table 8 – Postgres vs MySQL comparison based on features [28]

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 through 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 [17]. Based on the analysis above, MySQL is 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 6 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.

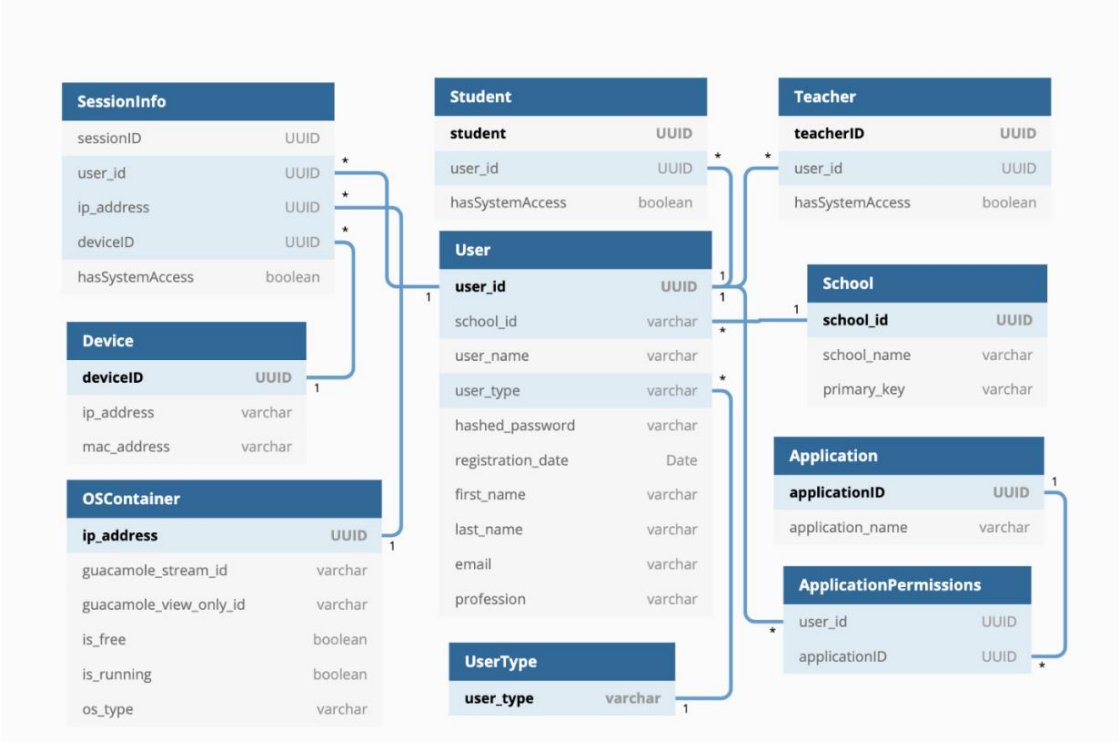

Figure 6 – Database Schema Physical Entity Relationship Schema

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.

3.7 Student Page

The student page is a critical piece of the StreamingOS web application that was created to satisfy the Student Birds Eye View functional specification. The function of this page is to allow teachers and instructors to monitor the activity on the devices that the students are using. This is done by displaying screenshots of the student's device for each active student, along with their name. Additionally, the page is updated every 5 seconds where the screenshots are updated to what the student is currently doing. This feature will further be expanded upon in section 3.7.1. The other feature of this page allows the

teacher or instructor to broadcast their own screen to all students in their class. This feature will be expanded upon in section 3.7.2.

3.7.1 Viewing Student Screens

Displaying a screenshot of the screen of each student's device is not a trivial task and took many steps to properly complete. When a user creates a session and logs into either Windows or Linux, their IP address as well as their name and student ID are placed in a python dictionary called *session_info_map* located in the reverse proxy. Similarly, this information is removed when the student logs out and kills their session. On the frontend, the /sessions endpoint is called every 5 seconds using *setInterval(getSessionInformation, 5000)*. When this is called, the application updates all images that exist as HTML s, creates new image 's for students that connected in the last 5 seconds, and deletes image 's for students that have recently logged out. While the student name comes directly from the /sessions endpoint, the image comes from the endpoint /user/<studentID>/screen/snapshot. In fact, since the reverse proxy returns a JPEG image directly, the image can be directly loaded into the HTML div as follows:

```
img.src = IPAddr + '/user/' + studentID + '/screen/snapshot?rand=' + Math.random();
```

The IP address is that of the reverse proxy. A random number is appended to force the element to update the image. Figure 9 shows a screenshot of this functionality, while the flow chart in Figure 7 shows a summary of this process for each student.

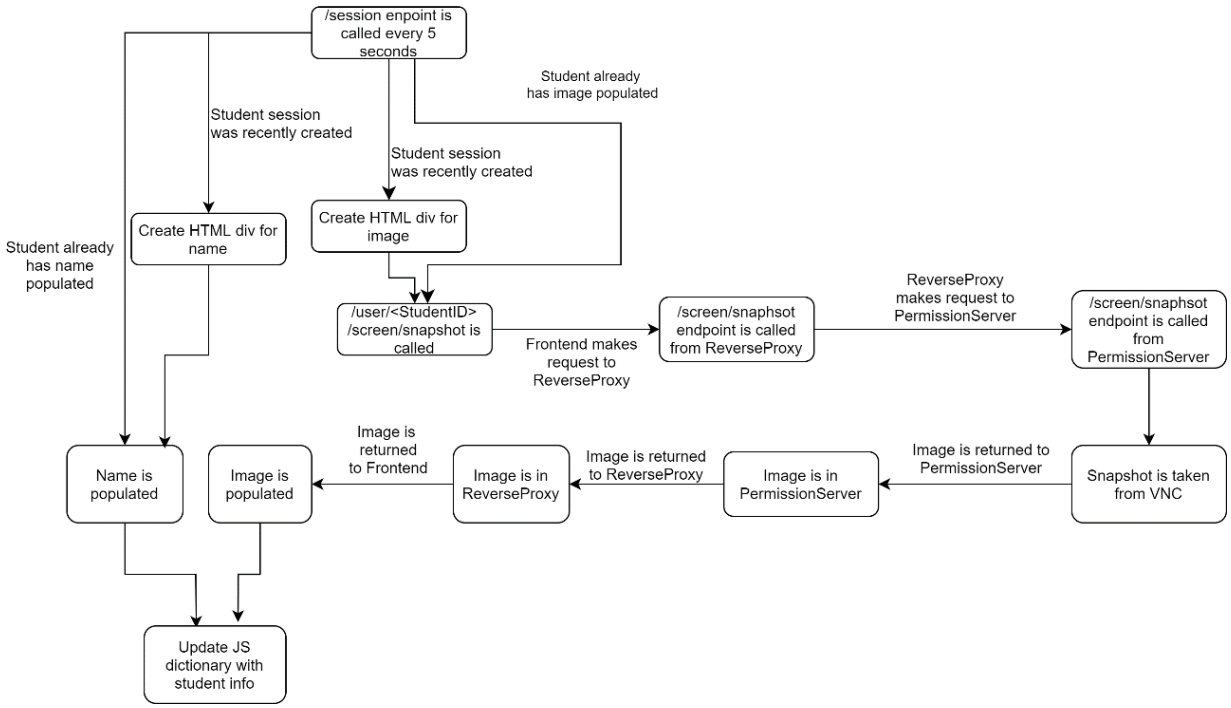

Figure 7 – Student Page Flow Chart

The main purpose for creating this page is to satisfy the Student Birds Eye View functional specification. The specification was met by displaying screenshots of the activity of every student and updating these screenshots every 5 seconds, which falls well within the limit of 10 seconds specified in the requirement.

This can be proven by the fact that a 5 second limit is coded directly into the JavaScript, and it can be shown using the tablet that the screenshots will update approximately every 5 seconds during a live demo. The data that was collected during live demos is shown in Table 9.

Table 9 – Student Screenshot Update Times for one student

Run	Time (s)
1	5.023
2	4.855
3	5.014
4	4.997
5	4.995
Average	4.9768

Within the reverse proxy code, every time a session is created, the user's session information such as name, user ID and session IP is saved in the *session_info_map*, and then retrieved every 5 seconds according to the model above.

The major design decision was to refresh the screenshots every 5 seconds. One important factor in this decision is the performance penalty in taking multiple screenshots. While it was important not to overload the VNC clients and reverse proxy with all the screenshot API requests, it was also necessary to set a screenshot refresh time that would accurately display the activity of each student. During testing, it was discovered that waiting for more than 5 seconds could easily result in missing important user actions on the system. Bringing the screenshot refresh time under 5 seconds resulted in increased loads on the VNC clients and reverse proxy due to the high frequency of requests. After some extensive testing, it was agreed that 5 seconds was the optimal screenshot refresh latency.

3.7.2 Teacher Broadcasting

Screen sharing is considered an important feature to enhance the classroom experience. When a student logs on to the client, they subscribe to the /subscribe endpoint. When the teacher decides to start broadcasting, the broadcast button calls the /broadcast/<user_id> endpoint. This endpoint stores the address of the OS container the teacher is currently connected to. All student clients currently subscribed gets an event with data every 10 seconds. The subscribe endpoint is implemented using Server-Sent events. This maintains an open connection between the server and client to update content without the overhead of periodically polling the server with AJAX requests. Once the `Broadcast` event is received by the student client, it disconnects from any existing guacamole connection. However, the user does not lose the current session as it remains stored in the *session_info_map* dictionary. The /setup/stream endpoint is called to setup the network routes in the reverse proxy to connect the student client to the teacher session in view-only mode. Once the teacher calls the /broadcast/stop endpoint, the student client receives a new event and disconnects from the teacher's broadcast session. The /restore/stream endpoint is called in order to restore any previous OS container that the student was previously connected to. This satisfies the screen sharing functional requirement. Figure 8 shows a flowchart of the workings of the teacher broadcast session. The /subscribe endpoint can also be used to receive message notifications which are broadcasted by the teacher using the /broadcast/message endpoint. Once the client receives

the JSON containing the message, a toast notification is displayed to the user. This satisfies the notification functional specification.

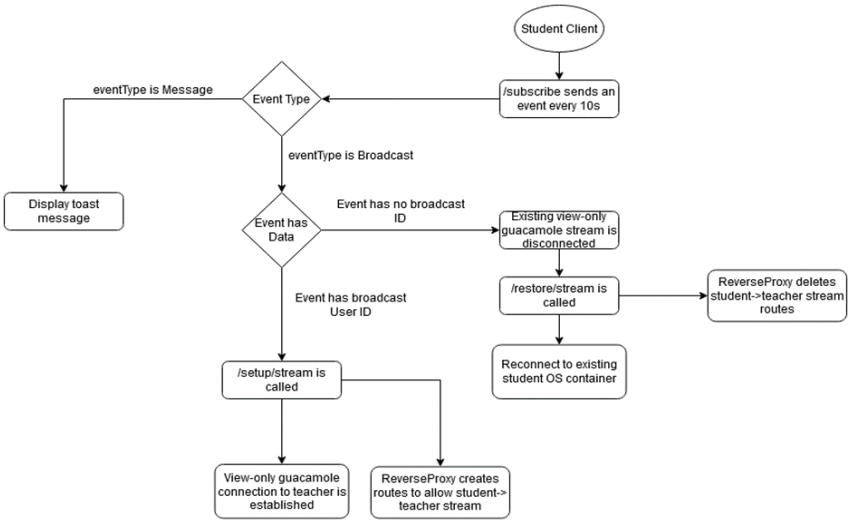

Figure 8 – Broadcast Stream Flow Chart

4.0 Prototype Data

Most of the prototype data/results are located throughout section 3 where it made the most sense to display it. However, some large screenshots and data that didn't make sense in section 3, is placed here in section 4. Additionally, information about the fulfillment of various functional and non-functional specifications is described in-depth in section 3.

4.1 StreamingOS Screenshots

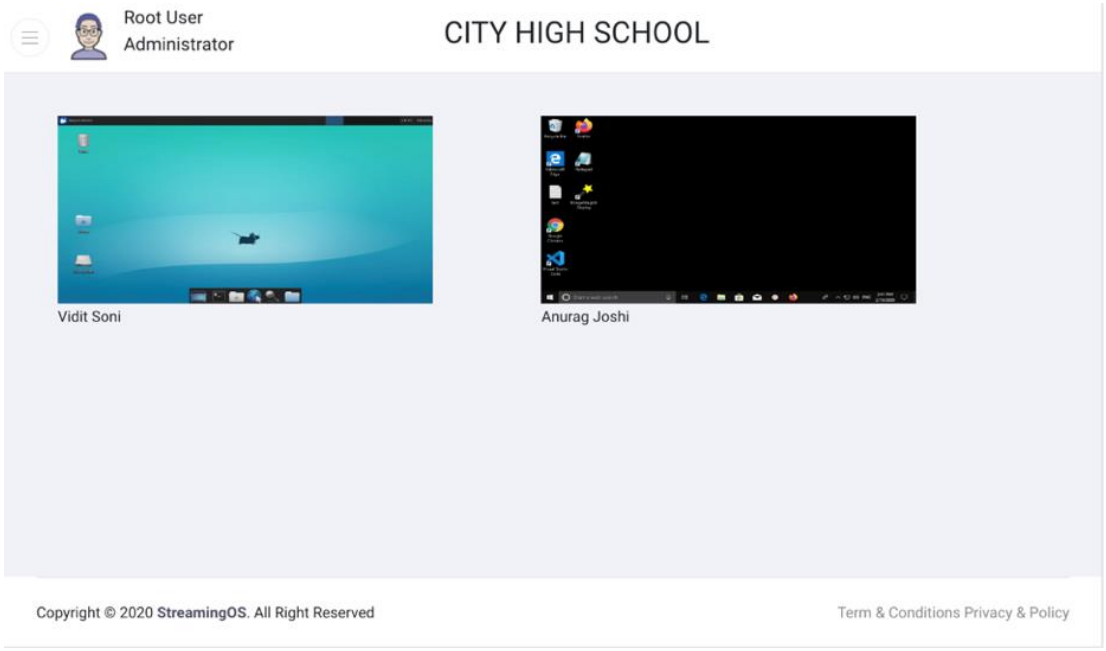

Figure 9 – Student Bird's Eye View

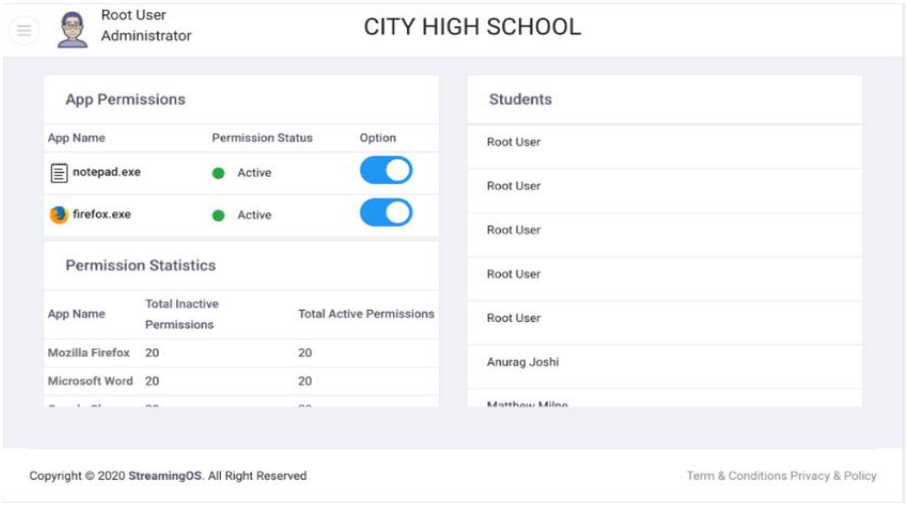

Figure 10 – Student Applications Permissions

Figure 11 – Connect page displaying number of free OS Containers

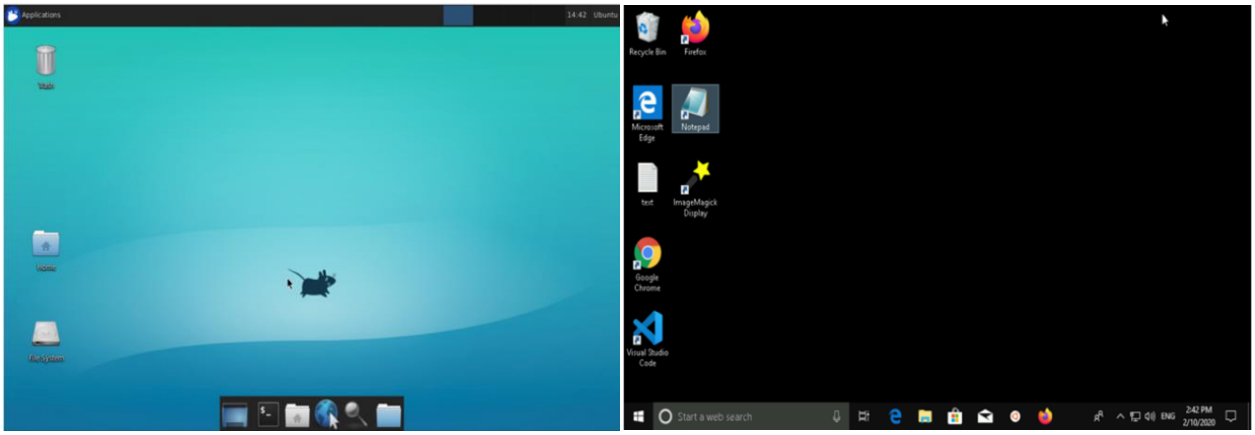

Figure 12 – Linux and Windows shown in an OS container

4.2 Student Cost Calculations

An important non-functional specification and selling point for the use of the StreamingOS system is the relatively low cost for using it. In this section, the monthly total VM cost, yearly total VM cost and yearly cost per student are calculated for two different cost models. For both models, some assumptions were made in the calculations. It is assumed that one Windows 10 VM and one Linux VM will be used for six students (allowing for one VM per three students). Additionally, the tablet costs used was $49.99, and a keyboard cost of $19.99 as obtained through Amazon. Both assumed to have a lifespan of 4 years.

The first cost model assumes that the virtual machines are reserved for StreamingOS use for 3 years. Operating under this basis, the costs can be calculated using the specifications that are collected in Table 10 [29].

Table 10 – Windows and Linux VM Spec Comparison

Parameters	Windows	Linux
Type of OS	B2S	B1S
Number of Cores	2	1
RAM	4 GB	1 GB
Temporary Storage	8 GB	4 GB
VM Cost (USD) Per Month	$19.48	$3.42
Disk Type	Standard SSD	Standard SSD
Disk Storage	32 GB	32 GB
Storage Cost (USD) Per Month	$2.40 + $0.20 (for transactions)	$2.40
Subtotal (USD) Per Month	$22.08	$6.02

Looking at the Subtotal row in Table 10, the cost for the usage of three Windows and three Linux VMs is $22.08 + $6.02 = $28.09 USD per month. Dividing this by 6 students, each student cost will be $4.68 USD per month. Multiplying this cost by 12 months, each student cost will be $4.68*12 = $56.18 per year [29].

Total Cost Per Year = 56.18 + 49.99/4 + 19.99/4 = $73.68 (per person)

The other cost model takes a pay-as-you-go approach. This assumes that VMs are kept active for 8 hours of the school day, and 5 days per week. In 4 weeks (1 month), 8h * 5 days * 4 weeks = 160 hours are spent using StreamingOS. The specs of the VMs are identical, but the costs are slightly different. For Windows, the cost is $9.22 USD per month for the VM, and the cost is $2.40 + $0.20 (for transactions and storage) per month, making the subtotal to be $11.82 USD per month. For Linux, the cost is $1.97 USD for the VM per month, and the storage/transactions cost is $2.40 per month, making the subtotal $4.57 USD per month. As a result, the cost for the usage of three Windows and three Linux VMs is $11.82 + $4.57 = $16.39 USD per month. Dividing this by 6 students, each student cost is $2.73 USD per month. Multiplying this cost by 12 months, each student cost is $2.73*12 = $32.76 per year [29].

Total Cost Per Year = 32.76 + 49.99/4 + 19.99/4 = $50.26 (per person)

Both of the cost models show that the per user cost of the full system is below $75 per year which fulfills the non-functional requirement of cost effectiveness.

5.0 Discussion and Conclusions

5.1 Evaluation of Final Design

Based on the evaluation performed in Section 3, the final design met most of the essential functional and non-functional specifications discussed in Section 2.1.

From the perspective of the front-end client, the application supports the functional specification of creating a students' birds-eye view where the teacher can view the screens of all students actively streaming. The user interface also correctly displays application-level permissions for each student and allows the teacher to grant/revoke these permissions at any time. Since the interface of the application is simple and straightforward, manually testing the interface is proved to be sufficient. Further testing needs to be done to ensure that the teacher can send notifications to the students' clients and that these notification messages get delivered to the clients correctly.

The backend system which consists of the reverse proxy, database, and the OS containers satisfied most of the functional specifications marked as essential. The reverse proxy constantly coordinates with the OS containers to ensure the connection is still active between the clients. The database is used to store the application permissions of each student which can be modified by a teacher in real-time. When the containers are enrolled in the system, a view-only guacamole ID is generated. This facilitates the student screen sharing feature mentioned above. The client is always unaware of the IP of the OS Container. During testing, attempts were made to reach the OS container directly from the client and this failed as expected. Server-Sent events is an industry-standard method to send reliable live updates to connected clients. This is utilized for the notification system. A variety of applications ranging from browsers (Firefox) to productivity tools (LibreOffice) to light-weight IDEs were installed and tested on the OS container to ensure the application diversity was satisfactory. The REST API calls implemented are chosen to be as safe and idempotent as possible, to meet the security requirement.

5.2 Use of Advanced Knowledge

The project combines knowledge gained from ECE 356, ECE 358, ECE 390, 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. The cost calculations performed in this report are based on concepts learned in ECE 390. General software related architecture and design decisions are made based on 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 guacamole client which uses SSL/TLS. 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 costs 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 OS container.

5.3 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 Amazon's Fire Tablet 7, they are much more expensive than Amazon's Fire Tablet 7. The chosen tablet will eventually need to be replaced. 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 troubleshot 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.

5.4 Quality of Risk Assessment

As this is mainly a software project, there are few safety hazards associated with it. The only risk comes with the use of the Amazon Fire 7 tablet. As with all electrical circuits, there is a risk of short circuits occurring and the tablet heating up or catching fire. However, this is extremely unlikely, as the tablets are well tested by the manufacturer before being sold. To reduce latency during the demo, the system will make use of a LAN connection instead of using the campus Wi-Fi, which could be heavily overloaded on the day of the demo. To improve the front-end and server security, OAuth was added and the client was applications using Cordova and Electron, both of which run on the desktop instead of a web-based client. Each software component was individually tested after which the full system was tested in an integrated environment to ensure that the risk of software failures was well mitigated. Overall, as far as the demo is concerned, it is strongly believed that all necessary steps have been taken to mitigate any potential risks and hazards that may arise.

5.5 Student Workload

Table 11 shows the percentage of the workload each student has taken on for the project in ECE 498A and ECE 498B.

Table 11 – Percentage of workload taken by each team member

Student	**ID**	**Percentage Workload**
Anurag Joshi	a35joshi	20%
Matthew Milne	mrmilne	20%
Surag Sudesh	ssudesh	20%
Vidit Soni	vmsoni	20%
Vinayak Sharma	v44sharm	20%

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] "Haoqin H7," 2020. [Online]. Available: https://www.amazon.ca/HAOQIN-Tablet-Inch-Android-Tablets/dp/B081RPYBPT/ref=sr_1_5?keywords=fire%2Btablet%2B7&qid=1581011662&sr=8-5&th=1. [Accessed 28 Jan 2020].

[4] T. M7, Lenovo, 2020. [Online]. Available: https://www.lenovo.com/ca/en/tablets/android-tablets/lenovo-tab-series/Lenovo-TB-7305/p/ZA550012US . [Accessed 28 Jan 2020].

[5] T. D. Specifications, Amazon, 2019. [Online]. Available: https://developer.amazon.com/docs/fire-tablets/ft-device-and-feature-specifications.html. [Accessed 28 Jan 2020].

[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] "Apache Guacamole," Apache Foundation, [Online]. Available: https://guacamole.apache.org/. [Accessed 15 01 2020].

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

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

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

[13] 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].

[14] 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].

[15] 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].

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

[17] "OAuth," [Online]. Available: https://oauth.net/2/. [Accessed 30 Jan 2020].

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

[19] V. Belski, "jotform," 2020. [Online]. Available: https://www.jotform.com/blog/ionic-vs-pure-cordova-97503/. [Accessed 11 Feb 2020].

[20] "JavaPoint," [Online]. Available: https://www.javatpoint.com/ionic-vs-cordova. [Accessed 11 Feb 2020].

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

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

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

[24] 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].

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

[26] 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].

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

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

[29] "Azure Pricing Calculator," Microsoft Corporation, [Online]. Available: https://azure.microsoft.com/en-ca/pricing/calculator/. [Accessed 15 01 2020].