

Faculty of Science, Technology and Arts

**Department of Computing
Project (Technical Computing)
[55-604708]
2019/20**

Author:	Edward (Ned) Hyett
Student ID:	B7032553
Year Submitted:	2020
Supervisor:	Martin J Cooper
Second Marker:	Simon Polovina
Degree Course:	BSC HON COMPUTER SCIENCE FOR GAMES
Title of Project:	Using Modern Web Technologies to Provision Networking Equipment for Degree Level Examination

Confidentiality Required?

NO ☒

YES ☐

Table of Contents

0. Abstract.....	4
1. Introduction	4
1.1. Client	4
1.2. Aims.....	4
1.3. Proposed Deliverables	4
1.3.1. The Examination Server	4
1.3.2. The Database	4
2. Investigation and Research	6
2.1. Identifying Requirements from the Client	6
2.2. Identifying Optimisations to the Examination Process.....	6
2.3. Literature Search.....	6
2.4. ASP.NET Core	6
2.5. Cisco and Networking	7
2.6. Databases.....	7
3. Planning and Design.....	8
3.1. Selection of Software Platforms	8
3.1.1. Programming Language	8
3.1.2. Database	8
3.1.3. Web Server.....	9
3.1.4. Communications Protocols	9
3.1.5. User Interface.....	10
3.1.6. Security Considerations	10
4. Development.....	11
4.1. Building a basic web platform.....	11
4.1.1. Creating Razor Pages.....	11
4.1.2. Using Models to Populate the Views	11
4.2. Creating the Database Schema and Models	11
4.2.1. Data Structures	12
4.3. Interfacing ASP.NET with a NoSQL Database.....	15
4.3.1. Reading data from the database.....	15
4.3.2. Writing data to the database	16
4.3.3. Updating the database.....	16
4.3.4. Deleting data from the database	16
4.3.5. Interfacing with Mongo from a remote location	17
4.4. Interfacing with Physical Hardware Over a Network.....	17

4.4.1.	The First Attempt	17
4.4.2.	Using ARP to find devices on the network.....	18
4.4.3.	Understanding Cisco VRF	18
4.5.	Updating the Web Platform to Interface with Physical Hardware	18
4.6.	Uploading and Storing Binary Files within the Web Platform	19
4.7.	Sending and Receiving Binary Files from Remote Devices	19
4.8.	Automating Deployment of Configuration Files and Collection of Results	20
4.9.	General Problems.....	20
4.9.1.	Why GNS3 was not an option	20
4.9.2.	Server randomly exits without any debugging trace	21
4.9.3.	Unable to have more than one model per Razor page.....	21
4.9.4.	“ArgumentException: Expression ... cannot be used for parameter of x” in a query... 21	
4.9.5.	Why the Cisco implementation of SSH EXEC in IOS is highly flawed	21
4.9.6.	POST requests made with the interrupt class were not being reported correctly.....	22
5.	Evaluation	23
5.1.	Critical Reflection	23
5.2.	Professional, Ethical Concerns and Risks	25
5.2.1.	GDPR and Data Security	25
5.2.2.	User-Generated Content.....	26
5.2.3.	Potential for Abuse	26
5.2.4.	Stability and Interruption of Business.....	26
5.3.	Next Steps	26
6.	Conclusion.....	30
7.	References	31
8.	Appendices.....	32
	Acknowledgements.....	35

0. Abstract

This report details the development of a system designed to simplify and streamline examination protocols used by the Sheffield Hallam University Networking Department using ASP.NET Core, MongoDB, TFTP and SSH. The development process involved distilling down the complex process of setting up, delivering, and completing examinations into distinct steps that can be managed through a Graphical User Interface. This was accomplished by using remote management tools to move static configuration information to networking devices without direct user intervention. The outcome was a platform which enables users and administrators to perform these actions efficiently. This platform provides a solid foundation from which future developments can be made.

1. Introduction

1.1. Client

The client for this project is the networking department of Sheffield Hallam University. For the most part the project and the requirements of the department were communicated by Roderick Douglas, course leader. The project was originally his idea as he has first-hand experience with the issues faced by the networking department in examining students. Future potential clients could include networking departments in other universities, as this issue is surely not unique to the networking department at Hallam.

1.2. Aims

The major aim of this project is to create a system that automates the process of assessing networking students in a reliable and repeatable way. This will reduce the amount of stress and tedium experienced by networking faculty, enabling them to spend time working on more productive activities.

1.3. Proposed Deliverables

1.3.1. The Examination Server

The Examination Server will be made of two parts: the web server that is used to interface with the project and the deployment system.

The web server will provide an interface that can be used by faculty to configure the system including uploading new exam templates and setting up learning groups that will enable mass deployment of an examination. The students can also use this interface to check their available examinations as well as start, submit and review their results from previous examinations. Additionally, the server will expose an API that enables remote control from other applications using a standardised API scheme.

The deployment system will consist of a remote-control protocol that enables the server to automatically issue commands to the devices that it manages and a file transfer protocol to transfer binary files to these devices.

1.3.2. The Database

The database will store all the information required by *NetExaminer*. This database can be created as a template within the Examination Server and can be created when the server first boots. This means that the database will not have to be provided as a separate deliverable. It will, however, have to be

installed separately as this is not within the scope of this project. The database should be designed in such a way as to limit data duplication as much as possible. Any data that is not required should not be kept and linking of data structures can be used to provide information contexts.

2. Investigation and Research

2.1. Identifying Requirements from the Client

After discussion with the client, there were several key aspects of the application that are necessary to provide a functioning and effective service. These are outlined in appendix 2. Unfortunately, due to misunderstandings the full list of requirements was not provided until 17/04/20, after the completion of the project and without enough time to make satisfactory changes to the codebase. The most important requirements identified by the client include:

- Automatic introduction of errors into a preconfigured system
- Automatic retrieval of information pertaining to resolution of the problem introduced

2.2. Identifying Optimisations to the Examination Process

The current system involves the networking team setting up a virtualised network in Cisco Modelling Labs (CML) that can have problems manually introduced into it by attendant networking staff. The scenario provided is static and takes time to configure manually. This limits the number of problems that the networking department can provide to students, making it harder to test a wider range of skills.

Therefore, optimising this approach will focus on automating the process of delivering problems to the student and increasing the potential variability of problems that can be presented to students, testing a wider range of their skills. This would ensure that the examination environment is controlled, and that each new examination is conducted under the same circumstances as every other.

Further information on the current examination process can be found in appendix 2.

2.3. Literature Search

As this was an uncommon approach to this issue, deciding on what topics to research posed a significant challenge. Using the university library portal, topics that were researched included: CISCO (Hardware, Hardware Configuration, Command Line, Configuration), Containerisation (Docker), Web Development, Web Security, SSH (Secure Shell), Virtualisation, REST (Representational State Transfer), ASP.NET, GNS3, MongoDB (NoSQL) and Networking Protocols.

Most of these returned no relevant results that were useful to the project. After several searches, relevant material was found mainly in e-book form. This mainly relied upon publications such as *Programming ASP.NET Core* (Esposito, 2018) due to the great overview of the new technologies provided by ASP.NET Core and more importantly, how it differs from previous versions.

Additionally, for the database portion of the project, *Practical MongoDB* (Edward & Sabharwal, 2015) was extremely useful for configuring MongoDB to be performant in this scenario, furthermore enabling foresight that will prevent potential future maintainers of the *NetExaminer* project from encountering unnecessary obstacles which have been a problem for network administrators in the past.

2.4. ASP.NET Core

According to Esposito (2018), ASP.NET Core is a massive step forward in the web server technology sector. It provides a complete rewrite of the ideology behind ASP.NET *and* the underlying .NET

Framework. In chapter one, Esposito suggests that the .NET Core project, along with ASP.NET Core signifies a complete U-turn in philosophy by Microsoft as they have rewritten the entirety of the .NET platform and associated technologies to be cross platform and multi-server compatible, unlike their previous attempts to shoehorn people into using IIS (Internet Information Services) and Windows Server. This alone is reason enough to pick ASP.NET Core over previous version and other software such as PHP as it is easier and more robust while now providing the same cross platform support that other solutions have previously enjoyed.

Esposito is extremely positive and does not discuss problems that may be encountered by using ASP.NET Core in this way as this book is simply a reference manual for understanding the platform. Additionally, the book is published by Microsoft as part of their materials and documentation to get people to use their products. Unfortunately, this platform is so new and constantly evolving that many people have not picked it up yet and other sources are scarce.

2.5. Cisco and Networking

Research into the structure of IOS was performed, however as the client was highly familiar with the topic at hand, little information was required. At the beginning of the project, a crash course in interacting with Cisco IOS was provided by the client. This covered connecting, changing settings, and moving files to and from the device. This proved adequate for the needs of this project.

2.6. Databases

The use of NoSQL databases for this project is essential due to the amount of unstructured and potentially unknown files and data types that the system will be processing. This is because many types of UGC (User Generated Content) will be uploaded to the system. This can include, but not limited to device configuration files, student response files and software binaries. According to Edward & Sabharwal (2015) NoSQL databases are very good at storing and retrieving vast quantities of unstructured data. As most data stored and owned by the system will be unstructured, this will result in a great performance increase over RDBMS (Relational Database Management Software) which would be the only reliable alternative. Edward & Sabharwal go on to note that NoSQL databases are very good at storing binary data efficiently. This further increases the reliability, scalability, and speed of the project as *NetExaminer* will be transferring binary data to and from users and specialist hardware, even at great load.

There are, however, a few downsides of using NoSQL databases in a high availability and production critical application. Firstly, as stated by Edward & Sabharwal, NoSQL has not been around for nearly as long as RDBMS. Consequentially, the community and software ecosystem that is present for NoSQL solutions is less developed, however it is growing quickly as these types of system become more mainstream (especially due to their inclusion in cloud hosting suites such as AWS and Azure). Furthermore, NoSQL administration tasks present a completely different set of problems than traditional RDBMS administrators are trained to deal with. While using NoSQL may be easier for the developer, it may not be as easy for the administrator to maintain.

3. Planning and Design

3.1. Selection of Software Platforms

The selection of the software platforms for this project is important because it affects how the client can use and provide the software and has a secondary effect on how it will interact with hardware already owned by the client. Therefore, choosing the software that is used has a high level of responsibility especially because this is a corporate software project. Key aspects that should be considered for this project therefore are: Security, Reliability, Scalability, Compatibility and Community.

3.1.1. Programming Language

The programming language for this project is an important factor to consider as it decides what tools and features are available during the development of the project. In this case, the chosen language was C# using .NET Core 3. .NET Core is now a much more mature platform with the recent release of version 3, which provides many of the new APIs that are required for most software projects.

.NET has a high level of operational security as it is a corporate project that is owned by Microsoft, one of the biggest software development companies in existence. This means that as many people rely upon it that Microsoft has a high level of responsibility in keeping it up-to-date and patching as many exploits as possible.

3.1.2. Database

Database and information storage are an important part of this project as it is processing a large amount of user generated content and moving it around the network. Therefore, a fast and reliable database is very important. However, balancing this with quality of life and the ease of information storage is an important concept. For this reason, MongoDB (*The MongoDB 4.2 Manual — MongoDB Manual*, n.d.) with GridFS (*GridFS — MongoDB Manual*, n.d.) is the database of choice. MongoDB is a NoSQL database, which causes a large amount of controversy in the data storage community. This is confirmed in chapter 2 of Practical MongoDB (Edward & Sabharwal, 2015) where they state that among the major flaws in current NoSQL implementations are a lack of maturity, administration capacity and a general lack of expertise in the solution, among other issues. However, it is explained that NoSQL databases are created for the internet age, providing high scalability and speed over data cohesion. NoSQL databases store information as ‘documents’ which are blobs of JSON data that can be interpreted back into models by the client software. Additionally, GridFS allows the client software to store large blobs of binary data in the database. This is especially important for *NetExaminer* as it stores and processes a large amount of user generated content. Uploaded content poses a security threat if it is stored on the local filesystem (especially if given execution permissions) however storing this in GridFS increases local security because files cannot be executed inside the database. Many internet applications must take extreme measures to protect their servers from malicious uploads. Once a malicious file has been placed on the file system, it can be easily executed, and the server is easily compromised. Many solutions to this include creating a dedicated upload directory that has execution permissions disabled. This is not available everywhere and is still a stopgap measure.

.NET has a driver that works perfectly with MongoDB and enables class serialisation to and from the database instantaneously using class models. Information regarding how this operates and is used by *NetExaminer* is provided in section 4.2.

3.1.3. Web Server

Because of the selection of .NET and C# as the programming language of choice, the web server of choice is Kestrel and ASP.NET. This means that the project will be using the Model View Controller system to display web pages. Models are pieces of information that are stored in the database that represent an entity. Views are pieces of code that define how a web page looks and how the information that is provided to the view is laid out. Controllers create views, pull information from the database, and receive web requests from the client. This separation of code units enables the reuse of code as much as possible reducing the overall workload and improves the readability, testability, and reliability of the code without any increase in effort.

3.1.4. Communications Protocols

To communicate with the hardware, it is important to use the most standardised protocols in order to improve the compatibility without extra effort. To this end, the protocols that are selected for communicating with hardware will consist of SSH for remote control and TFTP for file transfer. These two protocols can be combined to create a powerful solution for remotely manipulating and performing operations on remote devices.

SSH

SSH (Ylonen & Lonvick, 2006) stands for Secure Shell and enables a computer to remotely control another using a terminal. Mostly used by human operators to manage computers that are remote to their location. This however can be automated with use of an SSH library meaning that bulk commands can be sent to many devices at once. Using SSH.NET (*sshnet/SSH.NET*, n.d.) to connect to physical devices enables the massive parallel control of all of these devices and for the *NetExaminer* service to remotely update these devices for student use. SSH can be used to run scripts against the devices and can be used to run individual commands against the device. Running scripts against the device could open the possibility to run tests against the running device in order to increase the confidence of the device being correctly configured.

This solution is extremely scalable and compatible as it is supported by most, if not all Cisco devices that are the main target of this project. This means that the same set of commands can be run against many different systems without much extra work. Additionally, this system can be expanded to work with Linux devices.

TFTP

TFTP (Sollins, 1992) stands for Trivial File Transfer Protocol and is a system for moving files over the network with minimal overhead. This means that the implementation does not support user authentication, encryption, or anything other than moving files across a network. These issues are not to be worried about as for this purpose the protocol will only be used on a locally controlled and monitored network that has no access from unauthorised personnel. Using TFTP.NET (*Callisto82/tftp.net*, n.d.) to create a TFTP server inside ASP will enable the system to reuse the same database connection that is used for the rest of the platform to store information within the GridFS buckets. This means that any information that is taken from and sent to the server over TFTP is never

stored directly to the file system. This improves security and reduces the attack surface exposed by using the insecure TFTP protocol.

TFTP is an extremely simple protocol that is widely supported, therefore there is very little that can go wrong with it. It is for this reason the TFTP is also easily scalable as with the nature of the stateless transfer, more TFTP servers can be added without too much extra effort.

XModem

XModem (*XMODEM Protocol Overview*, n.d.) is a file transfer protocol that runs on top of a standard serial terminal connection. This is a very old protocol and as such is very slow and unreliable. Additional problems shown by the XModem protocol include most prominently the lack of simultaneous sessions, meaning that files must be copied one by one and no other terminal commands may be executed. Furthermore, there are very outdated libraries available for XModem in .NET, meaning that it is unlikely to be supported should there be any major bugs found in the implementation.

Pure Serial

This approach requires a local serial connection to the device instead of using the networking capability of the devices. This connection can be easily broken and interrupted, and failure checking is extremely difficult due to the number of potential failure points that are presented by the solution.

3.1.5. User Interface

The user interface is not a massive consideration here as most of the users of this service are going to be technically minded. Such people are going to care more about the functionality of the service than the way that it looks. For this reason, the choice of user interface does not really matter. The project will be using Bootstrap (*Bootstrap*, n.d.) which is an extremely common and popular interface framework at the time of writing. There will be very little if any custom CSS in this project as the interface does not need to be flashy or interesting and just needs to work. Exciting animations, Progressive Web Apps and excessive styling are not required for this project to fit the audience. Bootstrap provides a large amount of default components that can be used to improve the look and functionality of the application interface, including redesigned buttons, modals, and toolbars. This reduces the amount of work required to develop the application but does not provide significant overhead by loading an entire UI framework such as React, Angular or WebComponents.

3.1.6. Security Considerations

For the purposes of this project, security should revolve around linking their accounts to the university network. Most corporations and business entities, including universities, will be using Microsoft Active Directory to authenticate users. This means that *NetExaminer* should not have to store user credentials and process them. Unfortunately, due to ethical concerns, this project is unable to access the confidential information stored in the university Active Directory domain. This is because this would divulge personal information of entities that have not consented and would constitute a violation of GDPR and the Data Protection Act 1998 by the University.

4. Development

4.1. Building a basic web platform

The first step in creating this project was to create a new ASP.NET project. Configuring the project included creating a few utility classes. These include a default “RESTResponse” class that defines a basic response from the REST API that *NetExaminer* will expose to clients. This includes a basic success state, response message, response ID and payload. This object will be serialised down to JSON in order to send a standard response back to the web interface or any other consumers of the API in order to notify them of the state of the action, or the state of the server.

Additionally, creating an “AppSettings” class that stores settings that can be customised by the user enables the project to read settings values from the “appsettings.json” file that ASP.NET reads at start up. This can be used to customise the location of the database, ports used by TFTP other settings that can be important to customise. Loading these at start up involves modifying the “Startup.cs” file to read custom app settings.

4.1.1. Creating Razor Pages

Razor pages are like other technologies such as PHP, in the sense that the view page is combination of HTML and a pre-processing script that repeats, modifies, and updates the page depending on the values passed into it. This means that less HTML can be written, and more time spent making functionality. *NetExaminer* heavily makes use of this for tables whereby repeating segments of HTML can be written once inside a C# for-loop and sent to the browser many times.

Additionally, utilising the ASP server-side HTML attributes, Visual Studio can refactor the changes automatically instead of manually setting up anchor links that can get broken over time as the project grows.

```
<a asp-controller="ExamTemplate" asp-action="Assign" asp-route-id="@exam.NEObjId">Assign</a>
```

Figure 1 ASP server-side routing attributes

4.1.2. Using Models to Populate the Views

When creating a page in ASP.NET it is possible to pass a model class to the view. This means that the data stored by the database can be passed directly to the view page without too much interference. This can be combined with forms to easily pre-populate data for web forms on page load. The separation between the views, models and the controllers is an important aspect of using ASP.NET to build websites. It prevents the application from becoming hard to handle should small parts of it need to be updated as the dependencies are loosely coupled.

4.2. Creating the Database Schema and Models

For *NetExaminer* to operate successfully, there are many different pieces of data that must be stored and retrieved for later use. These pieces of data are structured using “models”. These are blueprints for how the computer should organise and store the information. Furthermore, these data structures can pull double duty to operate as the models for ASP.NET in the MVC structure, enabling data to be passed immediately from the database straight into ASP.NET without time consuming reformatting.

```
[HttpPost]
public IActionResult Create([Bind(
    nameof(NEDeviceTag.Name),
    nameof(NEDeviceTag.Description),
    nameof(NEDeviceTag.Type),
    nameof(NEDeviceTag.Options))] NEDeviceTag tag)
```

Figure 2 An example of the Bind attribute being used to limit model binding

This is accomplished safely by using the “Bind” attribute. When reading information from a web request, ASP.NET will attempt to use model binding to translate the raw data into structures understood by the rest of the application. Above is an example of how this works. When an action is called, such as the “Create” action, the web browser will have also provided information that represents a “NEDeviceTag”. ASP.NET will attempt to read this information to create a new instance of NEDeviceTag. The Bind attribute will allow the platform to discard information that is not safe to receive from the user. For example, this model also contains parameters such as “_id”, “NEObjId”, and “Enabled”. These are internal properties used to control data state. Enabling users to upload data that can set these parameters is dangerous, therefore ASP.NET is instructed to only bind the user editable data. Additionally, the use of the C# “nameof” directive enables the physical field names to control the binding. If the model is ever refactored (variable names are changed), then the nameof directive will also be updated by the IDE. This is far superior to using plain strings and remembering to update them every time a change is introduced.

4.2.1. Data Structures

In *NetExaminer* all data structures to be stored in the database inherit from the “NECommonModel” abstract base class. Below is the field layout of this model.

Name	Type	Description
_id	Bson ObjectID	This is a common index that must be present in all pieces of information saved to MongoDB.
NEObjId	Guid	This is a custom identifier used by <i>NetExaminer</i> to link objects together. It is extremely unique and due to its alphanumeric nature, makes sequencing attacks impossible as you cannot guess the neighbouring IDs.
LastUpdatedUserId	Guid	The ID of the user that last updated this record. Useful for auditing changes made to the system.
UpdateTimestamp	Date and Time	The last time this record was updated. Useful for auditing changes made to the system. Stored in UTC.
Enabled	Boolean	Indicates if the data is relevant. If set to false, <i>NetExaminer</i> will consider the data as if it were deleted. However, should <i>NetExaminer</i> still need the data later (i.e. to link remaining records) the data is still present.

In order to allow the user to use the service, two more models are required. Firstly, user data is stored in the application using the “NEUser” model. This information describes a physical entity and is subsequently subject to both GDPR and the UK Data Protection Act 1998.

Name	Type	Description
Username	Text	The username that the user must provide to sign in. I.e. “b7032553”
Password	Text	The SHA-256 hash of the user’s password. This password cannot be understood by the application except when the original password is provided.
Name	Text	The real name of the user, i.e. “Ned Hyett”
Role	User Role	The role, and subsequently the permission level of the user. Defined as three levels, Student, Staff, and Admin.

Coupled with this, to allow a user to authenticate with *NetExaminer*, a model known as “NEToken” is required. This contains a session token that is given to users that have passed the authentication stage and can be transmitted back and forth to the server without exposing a user’s password. These tokens can be invalidated at any time to prevent misuse.

Name	Type	Description
UserId	Guid	This is the ID of the user that this token belongs to, allowing the token to be easily resolved back to the owner for authentication.
CreationTimestamp	Date and Time	The time that this token was created, stored in UTC.
ExpiryTimestamp	Date and Time	The time that this token is due to expire, stored in UTC.
Value	Text	The value of the token that is given to the user. This is what resolves a request to this token.

When assigning exams there are three data models that are required, each storing more information about the context of use than the last. Firstly, the “NEExamTemplate” describes the most basic information about the exam. This includes what it is called, who made it and the files that it needs to operate.

Name	Type	Description
Name	Text	The name of the examination.
Description	Text	A brief description of what the examination is about.
ConfigurationData	Lookup table (string => guid)	A mapping of files to upload to the device. Filenames are mapped into GridFS bucket file IDs.
RequestedFiles	List of paths	A list of files to pull off the device when the examination is over.

When the exam is assigned to a group or a student the platform requires further context. This is made possible by the “NEAssignedExam” model. This model stores information about a specific examination set by the lecturers for a group of students. It is not intended to be reused past the due date of the examination.

Name	Type	Description
ExaminationId	Guid	The ID of the examination template that backs this assignment. Used to resolve the template for basic information.
TargetType	Target Definition	The type of the “Target” field, either Student or Group. Controls how <i>NetExaminer</i> will resolve the “Target” field.
Target	Guid	The ID of the target of this assignment. This can either be the ID of a student or the ID of the group. This is described by the “TargetType” field.
Duration	Nullable Time Span	An optional value that specifies a time limit for the examination. If this is not set, the exam will not stop students until they are finished.
DueDate	Nullable Date and Time	An optional value that specifies the latest time that a student can start an examination. If this is not set, the examination will not prevent students from starting.
MaxAttempts	Nullable 16-bit Number	An optional value that specifies the number of attempts a student can make on the examination. Not providing this value will allow unlimited attempts.

When a student starts attempting the examination, the system will require another model to store information relating to that specific attempt of the assigned instance of the examination template. This is known internally as the “NEAttempt” model. This stores the attempt information and eventually the results of the examination.

Name	Type	Description
AssignedExaminationId	Guid	The ID of the assignment that this attempt is related to. Used to resolve the assignment model.
StudentId	Guid	The ID of the student that is currently attempting this examination.
CompletedTimestamp	Nullable Date and Time	An optional value that specifies the time that the examination was completed. If this value is not present, the attempt has not yet been marked as complete.
Files	Lookup table (string => guid)	A lookup table mapping file paths for marking to GridFS bucket file IDs. This is used to construct the attempt archives.
UsingResources	List of Guids	A list of hardware IDs that are in use for this attempt. It stores the hardware selected by the student in order to access them later. This list is not used to compute access control.
StaffResponse	Text	A textual response that can be entered by a member of staff detailing the mark and the reasons behind it. This can only be set after the attempt is completed.

When physical devices are registered to the *NetExaminer* system they are stored in the database as “NEPhysicalResource” models. These models describe what the device is, and where to find it.

Name	Type	Description
Name	Text	The name of the device presented to users.
Description	Text	Arbitrary text describing the device to administrators.
Location	Text	Arbitrary text describing the physical location of the device to administrators.
MACAddress	Text	The MAC address of the device used for locating the device on the network.
IPAddress	Text	An optional value that provides a backup address for contacting the device should the resolution of the MAC address fail.
Tags	List of TagLinks	A list of tags that describe the properties of the device. What tags are required and what they mean can be configured by the administrators. These can be used to slot devices into examination profiles.

4.3. Interfacing ASP.NET with a NoSQL Database

Setting up the MongoDB database is a set-and-forget situation for development. Run the installer and leave it. Additionally, it is useful to install a piece of software called “MongoDB Compass”. This software enables you to modify and view data stored inside MongoDB. Once this was completed, it was time to install the driver into ASP.NET. Using NuGet to install the MongoDB driver was simple enough – however configuring Mongo is a bit more involved. Creating a database class called “NEDatabase” (NE standing for *NetExaminer*) that contains two other classes “Buckets” and “Collections”. “Collections” are MongoDB’s version of tables. These store data for the application to use later and can be viewed with Compass. “Buckets” however are very similar but operate very differently. “Buckets” use collections to store binary files in MongoDB in a very easy and indexable way. They can be streamed in and out of the database with the driver, meaning that uploaded files that are provided by the users of *NetExaminer* can be stored safely within Mongo without storing in an executable location, improving security and speed of the application.

The database class connects to the database using a lazy load method. The database is not contacted until the database is required. This means that whenever a collection or bucket is interacted with for the first time, the variables are initialised and the connection to the database is made. This means that if the server is restarted, it can start faster without having to spend time connecting to the database until someone uses it.

4.3.1. Reading data from the database

Using C# to read data from Mongo is extremely easy with the correct knowledge. C# contains a technology called LINQ. LINQ enables programmers to use C# to define search algorithms using predicates. An example of this is shown here:

```
ViewData["Resources"] = NEDatabase.Collections.PhysicalResources.AsQueryable().Where(x  
=> x.Enabled);
```

This means that the driver will convert the predicate to a MongoDB query command without further interaction from the developer. It also means that very complex commands can be created without

having to fully understand MongoDB query language. The “AsQueryable()” call converts the collection into a LINQ compatible object which can be interacted with in a similar way to any other enumerable.

4.3.2. Writing data to the database

Writing data into MongoDB is as easy as creating a new copy of the model class, populating the data, and writing it to MongoDB. An example of this is shown here:

```
resource.NEObjId = Guid.NewGuid();
while (resourcesQ.Any(x => x.NEObjId == resource.NEObjId))
    resource.NEObjId = Guid.NewGuid();

resource.Enabled = true;
NEDatabase.Collections.PhysicalResources.InsertOne(resource);
```

This means that data stored by the application will always be stored in the same format and can be easily converted from the format used by MongoDB to data that is useful to the application. When writing data to the database, there are currently no functional ways to set default property values without explicitly setting them after model initialisation. However, as shown in the example above, *NetExaminer* code does not need to explicitly initialise the model. The model shown above is initialised by ASP.NET during a request parameter binding to read information from the user. This would normally be dangerous, however ASP.NET is only binding parameters that are intentionally user editable. Mongo will also populate an internal document ID automatically (by default called “_id”) however this is interpreted as a number which reduces the amount of records that can be stored. Therefore, as shown above, a parameter named “NEObjId” is set by *NetExaminer*. This is a .NET Globally Unique Identifier which can provide more records than there are stars in the universe (Spector-Zabusky, 2010).

4.3.3. Updating the database

Updating documents with MongoDB uses predicates too. This is accomplished with UpdateDefinitionBuilders to create the commands that update the selected records. An example of this is shown below:

```
void do_update<TField>(Expression<Func<NEPhysicalResource, TField>> a, TField b) =>
    NEDatabase.Collections.PhysicalResources.UpdateOne(x => x.NEObjId ==
existingResource.NEObjId,
    new UpdateDefinitionBuilder<NEPhysicalResource>().Set(a, b));
```

This function is an inline method that can be used to reduce code reuse but also exemplifies the way to update a document in Mongo. Firstly, the document is selected by the NEObjId, and then an update builder is used to create an instruction to set TField a to value b. TField is selected through a predicate provided through the method call, as shown below.

```
do_update(a => a.Name, resource.Name);
```

4.3.4. Deleting data from the database

Deleting data from Mongo is easily the simplest operation to perform. Like inserting a document, this instruction can be performed in a single line, as shown below.


```
NEDatabase.Collections.PhysicalResources.DeleteOne(x => x.NEObjId == id);
```

By providing a predicate to the DeleteOne call, the driver can create an instruction that will delete any one record where the NEObjId parameter is equal to the ID that was provided to the predicate.

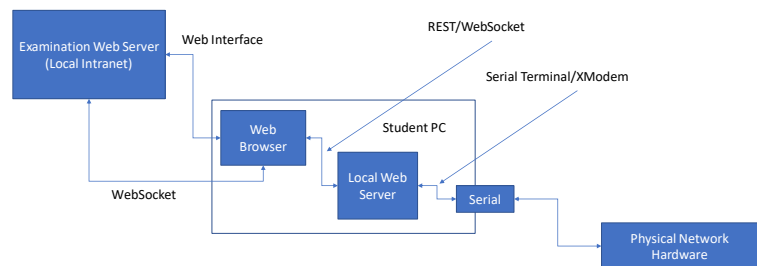
4.3.5. Interfacing with Mongo from a remote location

During development, it became necessary to connect to an already established Mongo instance from another computer. In order to make this work, the MongoDB configuration needs to be slightly edited. The change required to the configuration simply boils down to changing the bind IP (that tells Mongo what IP address to bind the public port to) from localhost (or 127.0.0.1) to 0.0.0.0. This instructs Mongo to bind to all interfaces on the device. This is not ideal, however for development this is an acceptable solution. Furthermore, the firewall rule enabling port 27017 to be accessed remotely needed to be created.

4.4. Interfacing with Physical Hardware Over a Network

4.4.1. The First Attempt

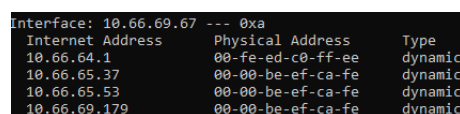
The first iteration of *NetExaminer* was created without the knowledge that the client was willing to create additional networking infrastructure to support the project. Therefore, a solution was devised to use student PCs as proxy hosts for physical hardware not connected to any kind of managed network. This involved creating a third deliverable that sat on a student PC with the sole purpose of enabling the web browser to proxy a WebSocket connection through to the server that sent commands over the serial port and proxying data back to the central server.



The main take-away from this approach was that it had many different points of failure that were outside of the control of the system. Especially problematic were the amount of points that were directly influenceable by the student taking the examination. This prevents the system from achieving any state of resiliency against cheating and exploitation, effectively reducing the reliability of the system to zero. Additionally, this system was using the serial connection of the device. This is an extremely slow connection and is unreliable for use in automation. The serial connection can only support one activity at a time, meaning that while the system was copying files it was unable to receive future commands. This system was known as the “Local Service” model and was scrapped as soon as possible. Very few elements of the system were fully developed before it was replaced.

4.4.2. Using ARP to find devices on the network

Before exploring how to solve the reliability issues posed by the Local Service solution, it is important to consider how an IP solution would operate. When there are many physical nodes that are managed by this platform, storing their location on the network becomes difficult. This is because each device is given an IP address by the DHCP. On a very simple network, these IP addresses can change as they are not statically assigned by the DHCP and therefore without proper knowledge, *NetExaminer* is likely to lose the devices on the network. In the high-availability situation that *NetExaminer* will be deployed in it is important for the system to “just work”. Additionally, this system will prevent *NetExaminer* from having to contain a duplicate list of IP addresses that the DHCP has, decreasing the need to interfere with the configuration.



Internet Address	Physical Address	Type
10.66.64.1	00-fe-ed-c0-ff-ee	dynamic
10.66.65.37	00-00-be-ef-ca-fe	dynamic
10.66.65.53	00-00-be-ef-ca-fe	dynamic
10.66.69.179	00-00-be-ef-ca-fe	dynamic

Figure 3 An example of the ARP table

Simply put, *NetExaminer* will send out a PING packet to all devices connected on the network via the Broadcast Address. This means that the underlying network infrastructure will update the ARP table with fresh information. *NetExaminer* will then read out the ARP table from the host operating system, locate the MAC address of the required device and read out the associated IP address. This can then be used by *NetExaminer* to communicate with the device until the lease expires. When *NetExaminer* experiences a communication failure with the device, it will automatically start the ARP discovery process again until it can regain connection.

4.4.3. Understanding Cisco VRF

Cisco VRF (Virtual Route Forwarding) is a capability provided by Cisco devices to effectively section off different ports on the device to different networks. *NetExaminer* uses VRF to create what is known as the “Management Network”. This replaces the “Local Service” with a new set of networking infrastructure that connects the test equipment back to an interface on the *NetExaminer* host system. This infrastructure is separate to that used by the students in order to prevent congestion and interference. The VRF provisions the management port onto a separate domain which is *technically* unreachable by the student. From here the device has a fully working networking configuration on the management port while enabling the rest of the device to be “broken” by the examination configuration.

The student can then work in the default domain, completely unaware of what *NetExaminer* is able to do to the device. However, the VRF enables *NetExaminer* to maintain a persistent connection to the device and manipulate it. The VRF ensures that whatever network is created by the student during the examination is not able to interfere with the management network created by *NetExaminer*. Using this connection, the system therefore is theoretically able to stop the exam early, and potentially even able to monitor for cheating in real time. The only caveat to this system is that the VRF is configured in the same way that the user is configuring the device and relies on the student not intentionally damaging the VRF configuration. Other than this, it presents an excellent solution to the problems posed by the Local Service model.

4.5. Updating the Web Platform to Interface with Physical Hardware

At this stage, the two systems are complete: the web interface and the management interface. The next big task is marrying the two. This is accomplished by creating a new web interface controller that handles attempting examinations. This controller will start the actions of interfacing with the hardware and transferring files. Unfortunately, the initial version of this system will be unable to

configure more than one device at a time. Future versions will support device tagging and resource slots for exams.

4.6. Uploading and Storing Binary Files within the Web Platform

The main problem with accepting User Generated Content (UGC) in a web application is the process of storing it on the server(s) and making sure that it is safe and will not infect or exploit the server. Most of the time this is done by uploading files directly to the server filesystem and then indexing them in the database, therefore requiring both a database lookup and a filesystem query before the data is ready for uploading or downloading. This is also considerably dangerous as files that are uploaded could potentially be executed while lying on the filesystem. The ideal solution would be safe and streamlined to fit the requirements of this project.

MongoDB GridFS was the solution to this problem. Using MongoDB GridFS means that the database can be used to store binary files in a chunked stream in a repository called a “bucket”. These buckets are huge collections of files that can be indexed and retrieved almost instantaneously by storing them in the database instead of the file system. These files can then be uploaded to and recalled to the server by requesting a small identifier string called a Globally Unique Identifier (GUID). When the server receives a binary file upload from the web platform interface, it can process the file (potentially scanning it for viruses among other validation methods) and immediately create a new file in the bucket and upload the file to the database.

```
var fileId = Guid.NewGuid();  
NEDatabase.Buckets.TemplateFiles.UploadFromStream(fileId, fileId.ToString(),  
file.OpenReadStream());
```

Figure 4 Uploading a file from ASP.NET to MongoDB GridFS

Subsequently if the file ever needed to be accessed again, providing the same file GUID to the GridFS system will retrieve the same file back as a stream.

4.7. Sending and Receiving Binary Files from Remote Devices

There are limited options that are provided by Cisco IOS for reading and writing files over the network. Since there will be multiple devices and users connected to the system using it simultaneously, a networking protocol for file transfer is necessary. TFTP provides the features for this solution and none of the bloat as the network will be fully isolated from the outside world and no user authentication is required by the system as outside actors will be unable to get past the airgap.

As the configuration and exam template files are uploaded to the platform to the MongoDB GridFS buckets, this data can be streamed to and from the TFTP server easily by using the paths provided to access data directly instead of through a file browser. For example, issuing a READ request to “/SUBMISSIONS/<ID>” will cause the TFTP server to read the file data from the bucket and transfer it as a full binary file to the requesting device, as per the following example.

```
//Get transfer data from the "file path"
var fileParts = transfer.Filename.Split('/', StringSplitOptions.RemoveEmptyEntries);
var bucketName = fileParts[0];
var fileId = Guid.Parse(fileParts[1]);

//convert the transfer path to the MongoDB bucket
var bucket = GetBucketForName(bucketName) ?? throw new Exception("Invalid bucket.");
...

var bucketStream = bucket.OpenDownloadStream(fileId);
transfer.OnFinished += delegate { bucketStream.Dispose(); };
transfer.Start(bucketStream);
```

Figure 5 Reading data from the buckets using TFTP

Furthermore, this also works for writing files to the buckets over TFTP. This works in much the same way as before, where the TFTP server receives a path to write data to and converts this into a series of identifiers that will select the buckets and process the file before finally writing the data stream into the bucket. This has the advantage of keeping the UGC in memory at all times until it is eventually stored in the database, ensuring high availability as writing to the disk is a slow operation that can increase response times. This will make sure that the file cannot be in a position where it can be executed by the host operating system as an application file.

4.8. Automating Deployment of Configuration Files and Collection of Results

In order to set up and assess testing environments, custom scenarios need to be created in configuration files and distributed to the remote devices. Additionally, the files generated by the students in order to complete the assessment need to be retrieved by the platform at the end of the examination.

Once the SSH and TFTP systems are in place, automating the deployment of files to remote systems and collecting files from them becomes trivial. Firstly, the SSH connection is made to the remote system, giving the platform remote control over the terminal of the device. Once this has been accomplished, the platform can issue copy commands over the SSH connection to send files to the device. The platform will look up the GUIDs of the files that it wishes to send, use these to generate a TFTP path (see section 4.7) that will enable the TFTP server to look up and retrieve the files to be deployed.

When the examination is over the platform will again make an SSH connection to the device, look for the files requested by the lecturers and use TFTP to copy them to the platform and sort them into buckets to be retrieved later by the examiners in a zip file.

4.9. General Problems

4.9.1. Why GNS3 was not an option

The initial idea for testing proposed by the client was to use a piece of software called GNS3. GNS3 is a network simulator that can run virtual devices and network them together on a single computer. It is generally used by networking specialists to model and preview network designs before committing to making the physical network. It was however suggested for use in this project as Cisco hardware is very expensive and the University is unable to allow students to take hardware off-site.

Unfortunately, GNS3 turned out to be incompatible with the version of Cisco IOS that was provided the University, because it was programmed for a custom CPU that is only present in Cisco devices. This required an alternative development method as this approach was untenable.

4.9.2. Server randomly exits without any debugging trace

This problem was one that was unable to be solved in the time available for this project. Essentially, the server will randomly and unpredictably exit the process without any indication that it was a planned shutdown. The server will not log any exit processes and will exit without the debugger being triggered. If this were to happen during an examination, the server will be stuck in an invalid state upon restarting. Therefore, an administrator would have to be called in to restart the process and clear out the invalid state. Fortunately, this error mostly happens upon first page load after the server is started. Until this problem is resolved, launching the application through a service manager that can notify administrators when an application error has occurred is the only way to ensure uptime.

4.9.3. Unable to have more than one model per Razor page

One of the downsides of using ASP.NET Core for this project is that the WebForms component of the Razor page generation engine does not support more than one data model per page. The “@model” instruction can only appear once and will not update the model for the second time it appears. This was a problem in the initial sign-in page where the sign-in and sign-up forms were adjacent. This was resolved by splitting the forms and any data entry into a dedicated data entry page for that model.

4.9.4. “ArgumentException: Expression ... cannot be used for parameter of x” in a query

When executing some queries, the server will an error that resembles the above. This error prevents the query from executing successfully and causes the server to return an error page to the client. This issue is a repeated error that has come up in several other projects using MongoDB. C# has a feature called “LINQ” which can be used to “transpile” lambdas into other languages through expression trees. The database driver uses LINQ to transpile the lambda into the query language for the database, in this case JavaScript. The transpiler does not know every part of the C# language and cannot transpile custom expressions. The crash occurs when the transpiler is presented with a custom expression that cannot be reduced into a query or is presented with a streaming list from a previous query. The database lookup does not execute in the server, it is sent to the database for it to look up locally instead of streaming irrelevant data back to the server. Therefore, the complex result of a previous query cannot be used in a new query unless the data is “realised”, meaning that the data has been fully streamed into server memory. This was solved by realising the data from a previous query and making the lambda operations performed on the database as simple as possible, mainly relying on standard comparator and arithmetic operations.

4.9.5. Why the Cisco implementation of SSH EXEC in IOS is highly flawed

While the use of SSH greatly improves the efficiency and reliability of *NetExaminer*, there is still one crucial flaw. Cisco has “incorrectly” implemented the SSH EXEC function used by SSH.NET to send commands without an interactive shell. The result of which is that after authenticating with the Cisco device over SSH (a potentially very expensive process that can potentially take upwards of several

seconds on high latency networks), the system is only able to issue one, single command to the device before the session is terminated by IOS. The ramifications of this are threefold:

- Any commands that change session state such as editing individual parts of the configuration are off limits
- Time is wasted re-authenticating with the device after every command
- The system is unable to maintain a true persistent connection with the device and monitor state

Therefore, while SSH is a theoretically perfect choice, alternatives should be considered should the project be continued due to this issue.

4.9.6. POST requests made with the interrupt class were not being reported correctly

After updating the local POST ajax script to support form multipart data, interrupted form submissions that were not using multipart were failing to process on the server as they were being formatted incorrectly. This was solved by disabling jQuery's ability to "auto-sense" what data is being sent to the server and generate headers automatically. Therefore, the fix was to make jQuery only disable the auto-sense when the interrupted form was using multipart data transmission.

5. Evaluation

5.1. Critical Reflection

The project, originally conceived by Roderick Douglas, was the automation of examinations for networking students using a centralised and user-friendly platform. In this regard, the project was mostly a success as it has provided the client with software that fulfils the majority primary goals of the project, as the application was able to accept examination templates, distribute them to groups of students and deploy them on physical hardware without human intervention, thus potentially improving the efficiency and reliability of the department examination processes.

Requirements that were not met included the automated testing capability to ensure that connected hardware was running properly at the time and finishing attempts that have run out of time. Unfortunately, notification of these requirements was only received on 17/04/20, near the completion of the project when there was not enough time to address them. This could have been mitigated through detailed and more structured communication with the client from the outset of the project. Further improvements to the communication could include more detailed note taking of meetings and to understand the importance of agreeing a specific and tangible set of requirements before even considering development.

The initial version of the project was created with very limited understanding of the Cisco system architecture and the way that these devices were operated – all that was known about the connectivity of the system for management processes at the time was that there was a serial port for console access. This led the project down a path of using the serial port and a library to control the serial communication through the student's PC. This attempt was ill-informed, and it became obvious that it was extremely inefficient and susceptible to student misuse, therefore prompting further communications with the client about other methods of controlling these devices.

During this conversation, a new way of accessing the devices remotely over a network became apparent. This turned out to be using Virtual Routing and Forwarding (VRF) to provision a specific port away from the control of the students that were accessing the device, connecting it to a separate and fully enclosed network between the device and the platform server. Returning to the client enabled the project to move forward with a fresh understanding of the requirements.

This led to the version two system that would become the final version of the project – using networking protocols to communicate with the devices and transfer commands and files without having to worry about a direct serial communication with the devices. This improved both the ability of the platform to multitask and the speed of the transfers and setup times.

There are many different ways that this project could have been approached – for example the use of an RDBMS instead of the NoSQL solution would have provided a more tried and tested solution for storing the data used by the platform, however the speed and ability to store unstructured data provided by NoSQL far outpaces the benefits of a traditional RDBMS. Additionally, the setup time and internal code required to support such a database would have been orders of magnitude greater for an RDBMS due to the requirement for fully structured and tabulated data. As NoSQL requires no such setup, it provides an instant plug-and-play solution that needs no user intervention.

Other alternatives, such as use of Apache and PHP instead of ASP.NET would have made the project significantly more complex. As .NET is closely tied into the framework of the web-server and provides low level system access on a scale that is not easily achieved with PHP, some features of the project could not be achieved as efficiently as they were in .NET, such as the integration with TFTP and SSH which would have been a lot slower using an interpreted language.

One particularly rough implementation in the deliverable was the web interface and how it linked together – it was very difficult to maintain a clear and repeatable standard throughout the design that made it easy to navigate around in the code. There were several times that this was rewritten however new methods for handling this have recently come to light. This includes the use of Dependency Injection (DI) to reduce the amount of hard to follow code that bloats and increases the complexity of referencing other components of the system. Candidates in the application for use in DI would be the database access, the TFTP server and the SSH remote device hook abstraction. These would have been useful to inject directly into controller classes as they are services that are consumed by the controllers and no other part of the application. This means that the DI injected interfaces could have been more easily updated if there was a change to the way that the system worked without having to rewrite the consumers of the service every time the service was updated.

```
<form asp-controller="Exam" asp-action="Begin" class="interrupt">
    <input type="hidden" name="id" value="@aExam.NEObjId"/>
    <select asp-items="@deviceList" name="resourceId"></select>
    <input type="submit" value="Begin"/>
</form>
```

Figure 6 A use of the .interrupt class

An identified weakness of the application included cases of the application using broken links to navigate the user around the interface after performing some operations such as uploading files or initiation of examinations. This was rectified by the inclusion of the “.interrupt” class on forms and links. This allowed JavaScript to handle the communication with the server, parse the output received from the server and use it to make navigation decisions depending on the output received. This means that while the action will still generate a JSON response for the HTTP API, when used by the internal system for navigation purposes it will still successfully move the user around the site without having to hardcode destinations on the client.

```
/// <summary>
/// Files that are collected by NetExaminer when a student has finished an exam
/// </summary>
public static GridFSBucket<Guid> Submissions
    => _submissions ??=
    new GridFSBucket<Guid>(Database, new GridFSBucketOptions
    {
        BucketName = "(Bucket) Submission Files",
        ChunkSizeBytes = 1048576 * 10, //10mb
        WriteConcern = WriteConcern.WMajority,
        ReadPreference = ReadPreference.Secondary
    });
```

Figure 7 Using GridFS buckets to write files to the database

One particularly effective aspect of this project was the integration of TFTP with the MongoDB GridFS system. This is due to the fact that it was able to serve and receive binary files over the network without having to store them on disk, increasing the security of the platform and making it easier to

access the data once it is stored on the system as no complex file auditing solution, which could result in the loss or disconnection of files from their records in the system over time, and provides a greater and easier collection of data for backup and replication as the files are stored in the same place as the rest of the data to operate the platform.

Another part of the project that was successful and exemplifies quality coding standards is the SSH remote device hook abstraction implementation. It provides a common interface for devices to be configured over SSH without the main application having to worry about the specifics of the remote command syntax. As an example, as far as the main platform server is aware, it is simply calling for a copy operation but the abstraction is converting that call into a SSH command that can be remotely executed in a different syntax depending on the type of device that is being connected to.

An important concept raised by this project is the ability for remote examination. As the platform is entirely web based, improvements could be easily made to the infrastructure to enable remote working. This would have been incredibly useful during the 2020 Coronavirus pandemic as it would enable complex examinations requiring specialist hardware to be completed remotely without substantial interruption.

5.2. Professional, Ethical Concerns and Risks

There are several risks in deploying and creating the project as it currently stands. These can mostly be worked on to improve compliance of the platform to regulations and common safety standards.

5.2.1. GDPR and Data Security

In today's world, data security and user privacy are, as they should, becoming more and more prevalent and in the forefront of the minds of the general population. To that end software and data storage is required to meet with more stringent law in order to prevent data misuse. To this end, *NetExaminer* attempts to comply with these regulations where appropriate. Because *NetExaminer* runs behind the university and could currently only be run by the university, the personal and identifying information that is required by *NetExaminer* to operate has already been collected by the university and can be used as wished. Furthermore, *NetExaminer* will only retain data that is necessary for continued operation. *NetExaminer* will sometimes only flag a piece of data as "deleted" instead of fully removing the data. This is either for auditing purposes or for operational reasons, where the data is required for the continued functionality of the platform.

Additionally, *NetExaminer* employs no user identification techniques further than the authorisation token used to log in and access the token. This is considered a functional token that is required for the service to work and cannot be disabled. No other identifiers are placed on the user's device.

Unfortunately, the current version of *NetExaminer* is not configured for database security. Due to this it accesses the MongoDB database over an unsecured channel and without authentication. This is a development measure in order to quickly use a local version of the database, however in order to comply with data protection regulations the software cannot be deployed in a commercial environment without modification to the database connection script.

5.2.2. User-Generated Content

UGC poses a threat to the integrity of the platform due to the vulnerabilities that can be exposed by processing, storing, and transmitting unstructured and unverifiable data provided by users. Should the application become compromised by the inclusion of a malicious payload into the system, measures have been taken to mitigate any potential damages. For example, the server will never allow UGC to be executed as program code on the file system and direct file uploads are restricted to authorised staff members only. However, should a malicious payload be deployed into the system, the system has no way of knowing that it is distributing a malicious file. Therefore, care should be taken to ensure that files retrieved from users are safe.

5.2.3. Potential for Abuse

As *NetExaminer* has direct access to administrate multiple expensive and powerful networking devices on a university network, care must be taken to ensure that the system cannot be used for nefarious purposes. This could include the use of *NetExaminer* to compromise connected systems, remove critical data or other such attacks. Even though *NetExaminer* will never knowingly allow remote code execution on remote devices, care should be taken to ensure that *NetExaminer* is not used by trusted personnel for undesirable reasons.

5.2.4. Stability and Interruption of Business

As *NetExaminer* is inserting itself directly into the path of “Business Critical” operations provided by the university, the stability and reliability of the platform must be called into question. For the time being the platform is not tested or feature complete enough to ensure continuation of business-critical operations. Furthermore, the platform has not been fully configured with a unit test framework in order to ensure that operations performed on the application meet QA testing standards. Further development in stability and reliability should be completed before a production-level deployment can be recommended. This, however, is intentional, as the project was only ever designed at this stage as a proof of concept.

5.3. Next Steps

Despite the successes of the project in providing a basic implementation of the requirements, there are many possible improvements, optimisations and new features that should be added to the application should there be future development. These features were omitted from the current project due to time constraints, technical or ethical limitations or that they became apparent due to experiences gained while creating the project.

Using Active Directory to Authenticate Users

Currently *NetExaminer* will authenticate users using an internal authentication scheme. This scheme works by storing session cookies in the database and using these to identify users when they return an authorisation bearer token in a request header or the “NE-Auth” cookie. ASP.NET provides a login system that can be backed with Active Directory which is much more secure than a custom solution that was written in a few minutes for this project. Additionally, the University is likely to use Active Directory for authorising users into their IT hardware and provisioning user accounts onto university hardware. *NetExaminer* can be attached to this system to share user authentication data with the rest

of the University systems and keeping extremely sensitive information out of the *NetExaminer* database.

Proper REST API and More Streamlined UI

The current REST API is incomplete and not fully standardised. This results in a confusing jumble that makes it hard to support and keep the application running smoothly. Therefore, it is best that the REST API is separated from the rest of the application (so that other applications can interface with *NetExaminer*) and the application itself will use an internal API that can handle the page navigation. This is currently accomplished by setting parameters of the “RESTResponse” class that instruct the JavaScript on the page to redirect. This same behaviour can be accomplished using a separate internal API that returns HTTP redirect responses allowing ASP to keep track of linked pages. A big problem that the current approach caused was during the initiation of exams, the endpoint that starts the exam for the student will return a piece of JSON that states that the exam was started instead of sending the user to the exam monitoring page. This could be better handled by an internal API.

Support for More Varied and Virtualised Hardware

Currently the *NetExaminer* platform only supports interaction with a limited number of Cisco devices. There is currently framework in place that could support other devices such as a Linux server and more. In future versions *NetExaminer* could interface with all manner of networking devices to configure and enable examinations on them. Furthermore, the networking department uses virtualisation software that can create virtual networks of devices that still believe they are physical machines (virtualisation). *NetExaminer* currently has no way of supporting these devices and there are APIs for these pieces of software that can enable them to interact directly with *NetExaminer* through the simulation instead of treating them as physical hardware.

Multi-Device examinations

Currently *NetExaminer* only supports one device per examination for technical reasons. This is because configuration files cannot be assigned to specific devices. In the future, *NetExaminer* should allow staff members to tag devices with specific properties that identify features of the device. When creating a new examination template, they can upload individual configuration files that can only be applied to devices that match the tags required by the examination template. This enables multiple devices and repeated devices in a single examination while allowing the physical hardware to change without changing the examination templates.

Device Provisioning and Booking Out devices

Currently *NetExaminer* assumes that all devices are available all the time. This can obviously cause conflicts if multiple students are attempting to access the same device at the same time. Therefore, work should be undertaken to allow *NetExaminer* to identify devices that are currently in-use or disabled by the staff and prevent students from using these devices.

Online Portal including tools to access remote devices through the web interface

Currently the devices must be networked to the student for them to interface with it. This means that the students cannot practice examinations at home or be given remote examinations in a time of crisis, such as the 2020 Coronavirus pandemic. In such circumstances *NetExaminer* should use an in-browser application, networking tunnel or other solution to ensure the devices are accessible to users who are not in physical proximity.

Bulk Hardware Management for University Staff

NetExaminer can perform automated remote actions on connected devices. This functionality could be leveraged by the platform and the networking staff to provide bulk actions on devices such as to update software, reset devices or other actions that will help the staff to perform tasks more efficiently. Administration should be able to issue bulk actions on connected devices or access a command prompt to enable remote access without having to visit each individual machine.

Emailing students and staff for system notifications

When students have a new exam available on *NetExaminer* they currently must be informed of it through other means. This may include being in a lecture or being emailed by the lecturer that set the exam. In situations that makes these methods ineffective, such as illness or human error, the *NetExaminer* system should be able to send email reminders to staff and students based on actions taken on the platform. This could reduce staff workload and reduce human error on the part of the staff members using the system. It will also assist students that are not as organised as others and prevent them from missing important examinations.

Automated test harnesses

In order to improve testing capacity and the usefulness of the application, software test harnesses should be definable that can be run against hardware and software applications to verify the status and operability of the solution provided by the student. Usually staff members would have to do this manually, and an automated system to provide this facility would be a significant time saving feature, as well as reducing the chance for human error. Harnesses could be uploaded and linked to the examination template through the web interface in order to reduce the work required to set new examinations.

Container Orchestration Support

The current solution is a monolithic single process application that can be easily overloaded should it encounter high load. Further development of this project should focus on reworking the application into a microservice architecture using container orchestration for automatic load balancing and failover. This would improve the stability of the application as well as reliability and uptime should the central server become overloaded. Furthermore, distribution of the application as a container image would smooth over many potential installation problems caused by dependencies and other 3rd-party software. A good framework for doing this is Microsoft Service Fabric as it already has integration with the other frameworks in use by *NetExaminer*, such as ASP.NET Core. Other frameworks that could be considered are Docker containers and Kubernetes.

Rewrite into a Dependency Injection format

The current system uses a large amount of static methods and very strictly structured code flows. This makes the program hard to test and hard to debug should something go wrong. Additionally, it vastly increases the amount of code required to interact with core parts of the application. Therefore, the application frequently refers to static classes such as “NEDatabase” which could be accessed in a much safer and more efficient way. Using Dependency Injection (DI) to inject components into parts of the system where they are needed is the recommended approach for ASP.NET applications. Instead of the “monolithic” “NEDatabase” class to store database access handles and database logic being stored in the application controllers (which makes it hard to update and test), the use of DI to inject services that handle these requests will improve the scalability of the application. For example, instead of using “NEDatabase” to access the “Submission Files” bucket, the DI application would expose a “Submission Service” that can be injected into controllers that need it. The Submission Service would expose several functions that can manipulate submission files and related functionality. If the application were to have a functioning unit test (this was attempted but never completed), the Submission Service implementation could be replaced with a version that returns immutable data that will remain the same enabling an expected output to be checked without interfacing with mutable data.

6. Conclusion

The general success of project depended on 3 identifiable core aims. Firstly, to create a reliable and repeatable examination context. Secondly, to reduce the complexity of the examination process for both staff and students. Thirdly, to maintain flexibility of a manual testing environment while still providing a substantial improvement to the testing capability.

The first aim was largely successful through the ability to store and automatically deploy configuration files based on examination context. Improvements could be made to the mechanism for uploading the examination to the server to allow for more complex setups and to reduce burden on the end user.

The second aim was met using HTML5 and related modern web technologies to create a user friendly and non-technical interface to a very technical problem, reducing the burden on the user. This was a successful aspect of the project, however there is room for improvement through more intuitive interface design.

It was a struggle to meet the third aim due to communication issues and a misunderstanding of the problem. Currently, the platform has limited application because it can only handle one configuration file per examination context. In the future it is essential that this functionality is included as it represents a core requirement of the application.

During this project, some essential lessons were learned. These include an improvement to timekeeping strategies, ensuring that requirements are well defined and communicated at the outset of the project, as well as an introduction to several unfamiliar and interesting technologies such as the power of NoSQL, and a further understanding of how advanced features in ASP.NET Core can be applied to a wider range of projects.

Overall, this project has provided opportunities to understand the business protocols for software development and why they are important. These protocols ensure that the focus of the software does not deviate from the vision of the client without good reason. Ensuring that the criteria is established early on and definitively is the most important take-away from this project.

7. References

- Bootstrap*. (n.d.). Retrieved February 18, 2020, from <https://getbootstrap.com/>
- Callisto82/tftp.net*. (n.d.). Retrieved February 18, 2020, from <https://github.com/Callisto82/tftp.net>
- Edward, S. G., & Sabharwal, N. (2015). *Practical MongoDB: Architecting, Developing, and Administering MongoDB*.
- Esposito, D. (2018). *Programming ASP.NET Core* (First Edit). Microsoft Press.
- GridFS — MongoDB Manual*. (n.d.). Retrieved February 18, 2020, from <https://docs.mongodb.com/manual/core/gridfs/>
- Sollins, K. R. (1992). The TFTP Protocol (Revision 2) RFC 1350. *IAB Official Protocol Standards*, 1–11. <https://tools.ietf.org/html/rfc1350>
- Spector-Zabusky, A. (2010). *Is it safe to assume a GUID will always be unique?* <https://stackoverflow.com/a/2977648>
- sshnet/SSH.NET*. (n.d.). Retrieved February 18, 2020, from <https://github.com/sshnet/SSH.NET>
- The MongoDB 4.2 Manual — MongoDB Manual*. (n.d.). Retrieved February 18, 2020, from <https://docs.mongodb.com/manual/>
- XMODEM Protocol Overview*. (n.d.). Retrieved January 22, 2020, from <http://techheap.packetizer.com/communication/modems/xmodem.html>
- Ylonen, T., & Lonvick, C. (2006). RFC 4253: The secure shell (SSH) transport layer protocol. *The Internet Society*, 1–32. <https://tools.ietf.org/html/rfc4253>

8. Appendices

Appendix 1 – The RESTResponse class

```
/// <summary>
/// Common REST response template from the server API.
/// </summary>
[PublicAPI]
public class RESTResponse
{
    /// <summary>
    /// Broadly determines if the request did what it was supposed to do internally.
    /// Separate to HTTP errors, this describes the task
    /// not the connection or server state.
    /// </summary>
    public bool Success { get; set; }

    /// <summary>
    /// A human understandable representation of what the response means (mostly used
    if <see cref="Success"/> is <see langword="false"/>).
    /// </summary>
    public string Message { get; set; }

    /// <summary>
    /// A generic object payload that is serialised into JSON for the client to read.
    This object is fully dynamic.
    /// Do <b>NOT</b> use this class to deserialise data received from the client or
    it could result in a serialisation
    /// injection attack. This is most commonly exploited by deserialising an object
    that contains a predicate, such as
    /// <see cref="SortedList{TKey,TValue}"/>, that runs after deserialisation and
    enables remote code execution.
    /// </summary>
    public object Payload { get; set; }

    /// <summary>
    /// A flag to cause the AJAX handler on the main site to force the local page to
    refresh.
    /// </summary>
    public bool RefreshLocalPage { get; set; }

    /// <summary>
    /// A flag to cause the AJAX handler to move the current page to this location.
    /// </summary>
    public string RedirectTo { get; set; }

    /// <summary>
    /// The timestamp on the server that the response was created.
    /// </summary>
    public DateTime Timestamp => DateTime.UtcNow;

    /// <summary>
    /// Internal storage field to maintain the response ID.
    /// </summary>
    private Guid _responseId = Guid.Empty;

    /// <summary>
    /// A random "nonce" ID that sort of helps identify a unique response.
    /// </summary>
    public Guid ResponseId => _responseId == Guid.Empty ? _responseId = Guid.NewGuid()
: _responseId;
}
```


This also introduces a difficulty for marking of the assessment. It is a reasonably simple process to check whether the Branch Server is available, but if it is no, it could be either the OSPF routing problem or the GRE Tunnel (or both) which the student has failed to resolve.

Desired Solution

A solution is required which has the following characteristics:

1. Administrators are able to pre-configure a number of problems which can be automatically introduced into the system.
2. Students should be able to select a problem to resolve, and allow the system to be configured with the faults.
3. Students should then be able to attempt troubleshooting and fixing the issue.
4. When students are satisfied with the results or have decided to try another problem, they should be able to signal the end of their attempt.
5. The system should then automatically perform some tests which check whether the problem has been resolved. In addition the configuration files of the relevant devices should be saved.
6. The system should then be returned to its initial state to allow the student to choose a different problem.
7. After a student has attempted all problems or decided to stop or run out of time, the system should save the relevant details for the student.

The system should authenticate users, and only allow users to access the particular setups required for their set of problems.

The system should be able to be modified by an administrator, who should be able to allocate problems to students.

If possible, the system should be able to grade student performance against a clear set of criteria such as

- the ability to ping from one system to another
- presence of given items in configuration files
- results from show commands such as “show ip route”

Acknowledgements

Throughout the creation of this project I have received a great deal of support and assistance from many. I would not have been able to get this far otherwise.

I would first like to thank my supervisor, Martin Cooper, for helping me with organisation and helping to set up meetings with those important to the project.

I would additionally like to thank Roderick Douglas, the client for this project, for his great assistance in providing me with a great crash course in interfacing with Cisco devices (despite not being a student in the networking course) and frequent help with my many questions about networking topics that I did not understand.

Furthermore, I would like to thank the other networking department lecturers that collectively helped me understand what I was doing by huddling around my computer at 6pm in the evening despite the fact that my lack of ability to concentrate clearly meant I should have gone home hours before.

Finally, I would like to thank my disability advisor Adele Beeson for the great help of gently reminding me not to waste time with this project, help with staying on track with my work and assistance in research and academic writing despite my many attempts to procrastinate and do everything but work on the project.