



---

# Sensor fusion for training doctors

---

Author:

**Dimitrios Selalmazidis**

Supervisor:

**Ghita Kouadri-Mostefaoui**

MSc Computer Science

September, 2018

This report is submitted as part requirement for the MSc Computer Science degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Department of Computer Science

University College London

# Abstract

My research is about stuff.

It begins with a study of some stuff, and then some other stuff and things.

There is a 300-word limit on your abstract.

# Acknowledgements

Acknowledge all the things!

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background Information and related work</b>	<b>3</b>
2.1	Outline of information sources used for development . . . . .	3
2.2	Literature Review . . . . .	3
2.3	Aims and Project Scope . . . . .	3
2.4	Project Timeline . . . . .	3
2.5	Work Balancing and Process Framework . . . . .	3
<b>3</b>	<b>Requirements and Analysis</b>	<b>4</b>
3.1	Problem Statement . . . . .	4
3.2	Software Requirement Listing and Prioritization . . . . .	4
3.2.1	Functional Requirements . . . . .	5
3.2.2	Non Functional Requirements . . . . .	6
3.3	Domain Modelling . . . . .	7
3.4	Use Case Analysis . . . . .	8
3.5	Use Case Diagram . . . . .	9
3.6	Mock-ups . . . . .	9
<b>4</b>	<b>Design and Implementation</b>	<b>14</b>
4.1	Architecture and System Structure . . . . .	14
4.2	Model-View-Controller Pattern . . . . .	15
4.3	Design Class Diagram . . . . .	17
4.4	Database Design . . . . .	19
4.4.1	Introduction . . . . .	19
4.4.2	Conceptual Schema . . . . .	19
4.4.3	Logical Schema . . . . .	19
4.4.4	Physical schema . . . . .	20

4.5	Pages Implementation . . . . .	20
4.5.1	“Register new operation” Implementation . . . . .	21
4.5.2	“Search for an operation” Implementation . . . . .	25
4.5.3	“Operation Details” Implementation . . . . .	26
<b>5</b>	<b>Debugging, Testing and Results Evaluation</b>	<b>29</b>
5.1	Unit Testing with MSTest . . . . .	29
5.2	Responsiveness Tests-Cross Browser Compatibility . . . . .	31
5.3	System Testing . . . . .	31
<b>6</b>	<b>Conclusion and Project evaluation</b>	<b>34</b>
6.1	Project Goals . . . . .	34
6.2	Fulfilment of Personal Aims . . . . .	35
6.3	Critical Evaluation . . . . .	35
6.4	Future Work . . . . .	36
6.5	Conclusion . . . . .	36
	<b>Appendices</b>	<b>38</b>
<b>A</b>	<b>Source Code</b>	<b>38</b>
A.1	Database creation . . . . .	38
A.1.1	View Creation . . . . .	44
A.2	Home Controller . . . . .	44
A.3	Database Context . . . . .	56
A.4	File Storage Controller . . . . .	76
A.5	Media Utilities Controller . . . . .	82
<b>B</b>	<b>System Manual</b>	<b>94</b>
	<b>Bibliography</b>	<b>95</b>

# List of Figures

3.1	Domain Model . . . . .	7
3.2	Use Case Diagram . . . . .	10
3.3	“Add new Operation” View . . . . .	11
3.4	“Search for an Operation” View . . . . .	12
3.5	“Operation Details” View . . . . .	13
4.1	ASP.NET Structure . . . . .	14
4.2	Application General Structure . . . . .	15
4.3	The Model-View-Controller Pattern of ASP.NET Core . . . . .	16
4.4	Design Class Diagram . . . . .	18
4.5	Conceptual schema . . . . .	20
4.6	Entity-Relationship Diagram . . . . .	21
4.7	Physical database schema . . . . .	22
4.8	NewOperation (HttpGet Request) . . . . .	23
4.9	Register a new operation Page . . . . .	24
4.10	AJAX Call . . . . .	24
4.11	Extracting meta-data from video files . . . . .	26
4.12	Extracting meta-data from audio files . . . . .	27
4.13	SearchOperationViewModel & SingleOperationViewModel . . . . .	28
4.14	Search for an operation Page . . . . .	28
5.1	Unit Test Report . . . . .	32
5.2	Responsiveness Tests . . . . .	33

# List of Tables

3.1	Functional Requirements . . . . .	6
3.2	Non-Functional Requirements . . . . .	6
3.3	Use Case Listing . . . . .	8
3.4	Use Case 01 - Upload a new operation . . . . .	8
3.5	Use Case 02 - Search for an operation . . . . .	9
3.6	Use Case 03 - View operation details . . . . .	9

# Chapter 1

## Introduction

Some stuff about things. [1] Some more things.

Inline citation: Anne Author. Example Journal Paper Title. *Journal of Classic Examples*, 1(1):e1001745+, January 1970

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin.



Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

# **Chapter 2**

## **Background Information and related work**

### **2.1 Outline of information sources used for development**

Jordan De souza

### **2.2 Literature Review**

### **2.3 Aims and Project Scope**

elizabeth

### **2.4 Project Timeline**

elizabeth

### **2.5 Work Balancing and Process Framework**

elizabeth

# Chapter 3

## Requirements and Analysis

This Chapter establishes the requirements and analysis for this work. The developments of the requirements is a crucial part of the analysis for almost all of the software development projects. This phase is extremely useful, as for it's flexibility - if things that are unnecessarily complicated are discovered, or are less important to the client, they can be disregarded or their importance could be downgraded so that they take less time. This is significantly useful for the implementation phase. This is also the easiest time to communicate with the client, while the subject matter is strictly textual, so any major issues or changes to the specifications can be established here.

### 3.1 Problem Statement

Design a web based application, that allows surgeons to register an operation, upload and view all the extracted data coming from the operating theatre's sensors.

### 3.2 Software Requirement Listing and Prioritization

This stage facilitates a comprehensive understanding of the client's needs and to appreciate the potential complexity of the system. Subsection 3.2.1 details the functional requirements of our system. Functional requirements show the behaviours of the system - what the client wants the system to do. Subsection 3.2.2 goes through the non-functional requirements, which affect the system's performance . The requirements were gathered through discussion with the client and also by continually iterating over what the application would be used for, so that additional focus should be given on the user needs.

As mentioned, the requirements have been divided into functional and non-functional [2]. They have also been prioritised according to the MoSCoW prioritisation technique [3], in order to guide the progress of the project and to ensure that a base-level application that achieved the goals of the project. The MoSCoW priorities refer to the order of design and development in order. Must have, Should Have, Could Have, and Won't have requirements.

### 3.2.1 Functional Requirements

ID	Functional Requirements	Priority
<b>Uploading a new Operation</b>		
1	The platform shall support a User Interface that will allow the user to register a new operation	Must
2	The platform shall be able to take as input from the user, the hospital that the operation took place	Must
3	The platform shall be able to take as input from the user, the operating room that the operation took place	Must
4	The platform shall be able to take as input from the user, all the staff that participated in the operation	Must
5	The platform shall be able to take as input from the user, the type of the operation (Neuro-surgery, Hand surgery, Paediatric surgery etc.)	Must
6	The platform shall be able to take as input from the user, the specific patient that has undergone the surgery including the patients unique identification number	Must
7	The platform shall be able to take as input from the user, all the video files that were produced during the operation	Must
8	The platform shall be able to take as input from the user, all the audio files that were produced during the operation	Must
9	The platform shall be able to take as input from the user, the file extracted from the patient's monitoring system	Should
10	The platform shall be able to record, store and distinguish data from different operating theatres	Must
11	The platform shall be able to store all the input data from the user to a relational database	Must
12	The platform shall store all the data coming from an operation in a way that all relevant data of the operation could be extracted	Must
13	The platform shall be able to process the video, audio and patients monitoring files uploaded from the user	Must
14	The platform shall be able to extract all the meta-data from the input files (encoded date, size, duration, file type, file name, full file path)	Must
15	The platform shall be able to store the meta-data extracted from the input files, to the relational database	Must
16	The platform shall be able to store the input files to an Azure Blob Storage Account	Must
17	The platform shall be able to link the files stored in the Azure Blob Storage with the relative operation identification number stored in the relational database	Must
<b>Searching for an Operation</b>		
18	The platform shall support searching capabilities; the user of the platform must be able to search an operation/procedure using relevant criteria	Must

ID	Functional Requirements	Priority
19	The platform shall support filtering capabilities where the user can apply specific filters for an operation (hospital name, operating room number, from/to date, doctors name, patient's name, type)	Must
<b>Details of an Operation</b>		
20	The platform shall be able to display a specific page where the user can see all the relative information and details of a specific operation	Must
21	The platform shall be able to retrieve all the relevant data from the operation (video data, microphone data, patient's monitoring system data etc.)	Must
22	The platform shall be able to convert the data coming from the different sensors to easily handled format ( e.g. convert a variety of video input to .avi)	Could
23	The platform shall support the capability of switching to a specific moment in time and view all the recorded data at that moment	Could
24	The platform shall be able to present to the user all the data recorded from the sensors at the same time. For example it could be able to view the panoramic camera, the light camera, the heart rate, the temperature of the room through the common factor of time.	Could
25	The platform shall provide the media files' URL to the user in order for the user to have access to them and re-play them	Must

**Table 3.1:** Functional Requirements

### 3.2.2 Non Functional Requirements

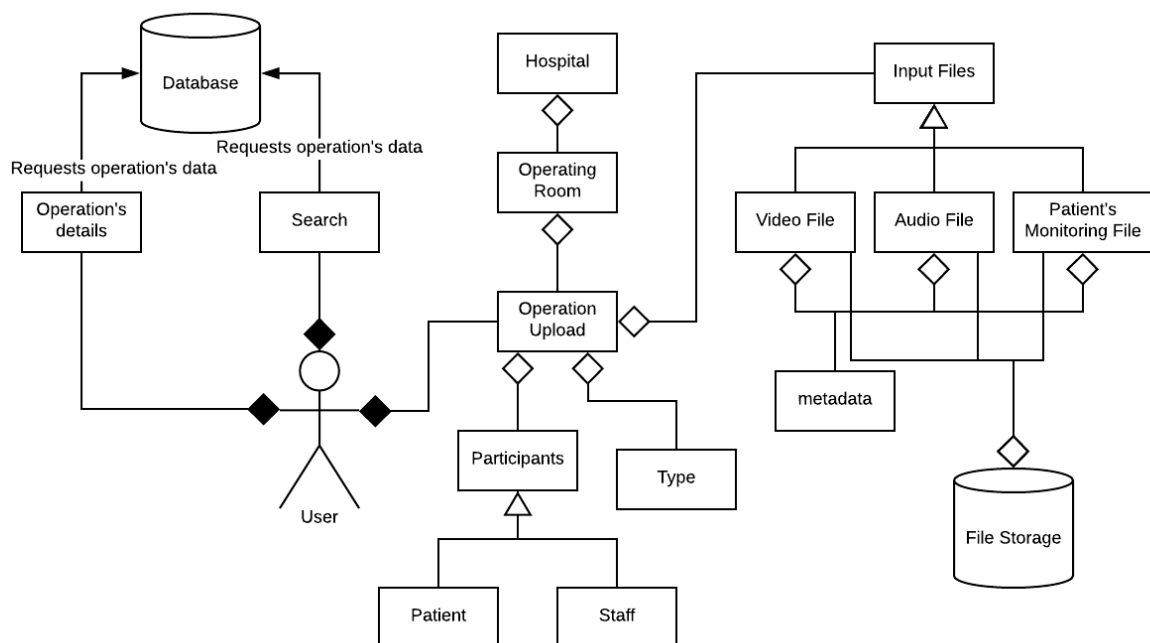
ID	Non-Functional Requirements	Priority
26	The platform shall use web browser as its user interface	Must
27	The platform shall support new sensor adding or should require minimal work for new, unknown sensor adding	Should
28	The platform shall store all the input data in a way that they could be extracted for machine learning purposes (machine learn-able data)	Must
29	The platform shall be independent of the format of the input data	Should
30	The platform shall work on a wide range of operating conditions (screen size, internet connection, performance)	Must
31	The platform shall be a C# web application with an MySQL Database	Should
32	The System shall present search results within 5 seconds	Should

**Table 3.2:** Non-Functional Requirements

### 3.3 Domain Modelling

The domain model looks to identify the objects of a system in terms of the requirements. It provides a conceptual real-world view of the system and enables a simplistic overview of the system's functionalities in terms of the problem domain entities [4]. More specifically, the classes were derived from the requirements list where the identified entities were marked in bold. Therefore, it is now possible to create a simple draft domain model from these entities. Furthermore, objects which were deemed redundant from this iteration were also discarded to refine the list of entities. This initial model was then revised and additional domain objects were added. Similar to previous analyses, the domain model is therefore the result of an iterative process.

Following the refinement process, using the identified entities of the requirements list, the domain model was revised in order to find any unspotted domain entities that weren't already in the requirements list. Figure 3.1 shows the model that resulted after the various iterations. It is a summary of the responsibilities of the system on a general level and was the basis for constructing the use cases in section 3.4.



**Figure 3.1:** Conceptual model of the system's problem domain

### 3.4 Use Case Analysis

Following the domain modelling and the gathering of requirements analysed in sections 3.3 and 3.2 respectively, use cases were developed. The defined domain entities and requirements were used to construct the use cases and their basic relationships. The construction of the uses cases allows the solid organisation of the functional requirements of the project, in order to make sure that they meet the front-end requirements and also to make sure that they are fully encompassing. A use case describes a scenario where a user interacts with the application in order to achieve a specific outcome [5]. Use cases are described from the users point of view rather than a technical point of view. As such, use cases are very effective at visualising and communicating the final product to the client and incorporating the client's voice into the requirements of the project [6].

A complete overview of the identified use cases is shown in Table 3.3 and from Table 3.4 to Table 3.6, all the detailed specification for each use case are shown. Each specification includes details of each use case, the main flows, error flows, post-conditions, pre-conditions and the trigger event.

ID	Use Case
UC01	<b>Upload a new operation</b>
UC02	<b>Search for an operation</b>
UC03	<b>View operation details</b>

**Table 3.3:** List of use cases

Use Case	Upload a new operation
ID	UC01
Brief Description	The user wishes to register the operation to the System
Preconditions	User must have all the necessary files and information of the operation
Main Flow	<ol style="list-style-type: none"><li>1. Home page is displayed</li><li>2. The User selects to register a new operation</li><li>3. The User enters the hospital name</li><li>4. The User enters the operating room number</li><li>5. The User enters all the staff that participated in the operation</li><li>6. The User enters the patient that underwent the surgery</li><li>7. The User enters the type of the operation</li><li>8. The User selects all the relevant video files</li><li>9. The User selects all the relevant audio files</li><li>10. The User selects the patient's monitoring file</li></ol>
Post Conditions	A new operation has been registered to the database and the files have been uploaded to the file storage
Alternative Flows	The User no longer wishes to upload yet the operation to the System and cancels the registration of the operation
Error Flow	<ol style="list-style-type: none"><li>1. The User does not enter any of the 1-7 fields and at least one input file</li><li>2. An error message is displayed</li></ol>
Trigger Event	The user chooses to add new operation

**Table 3.4:** Use Case 01 - Upload a new operation

Use Case	Search for an operation
ID	UC02
Brief Description	The User wishes to find an operation from the database
Preconditions	At least one operation must have been registered to the system
Main Flow	<ol style="list-style-type: none"> <li>1. Home page is displayed</li> <li>2. The 20 most recent uploaded operations are displayed</li> <li>3. The User uses the Filters to input hospital name, operating room number, from/to date and participated staff</li> <li>4. The filtered results are displayed, ordered by date (from most recent to oldest)</li> </ol>
Post Conditions	The User can see the Operations that correspond to the Filters they have applied
Alternative Flows	The Operation that the User searches for and the Filters applied produce no results
Error Flow	None
Trigger Event	The User wishes to Search for an Operation

**Table 3.5:** Use Case 02 - Search for an operation

Use Case	View operation details
ID	UC03
Brief Description	The User can view the details of an Operation
Preconditions	<ol style="list-style-type: none"> <li>1. At least one operation must have been registered</li> <li>2. The User must have searched for an Operation</li> </ol>
Main Flow	<ol style="list-style-type: none"> <li>1. Home page is displayed</li> <li>2. The User searches for an Operation</li> <li>3. The User views the filtered results</li> <li>4. The User selects the desired Operation</li> <li>5. The User views the Operation details</li> </ol>
Post Conditions	The Operation details are displayed to the User
Alternative Flows	The User does not wish to view the Operation details and so goes back to the home-page
Error Flow	System fails to report Operation Details
Trigger Event	The User wishes to view the details of a listed Operation

**Table 3.6:** Use Case 03 - View operation details

## 3.5 Use Case Diagram

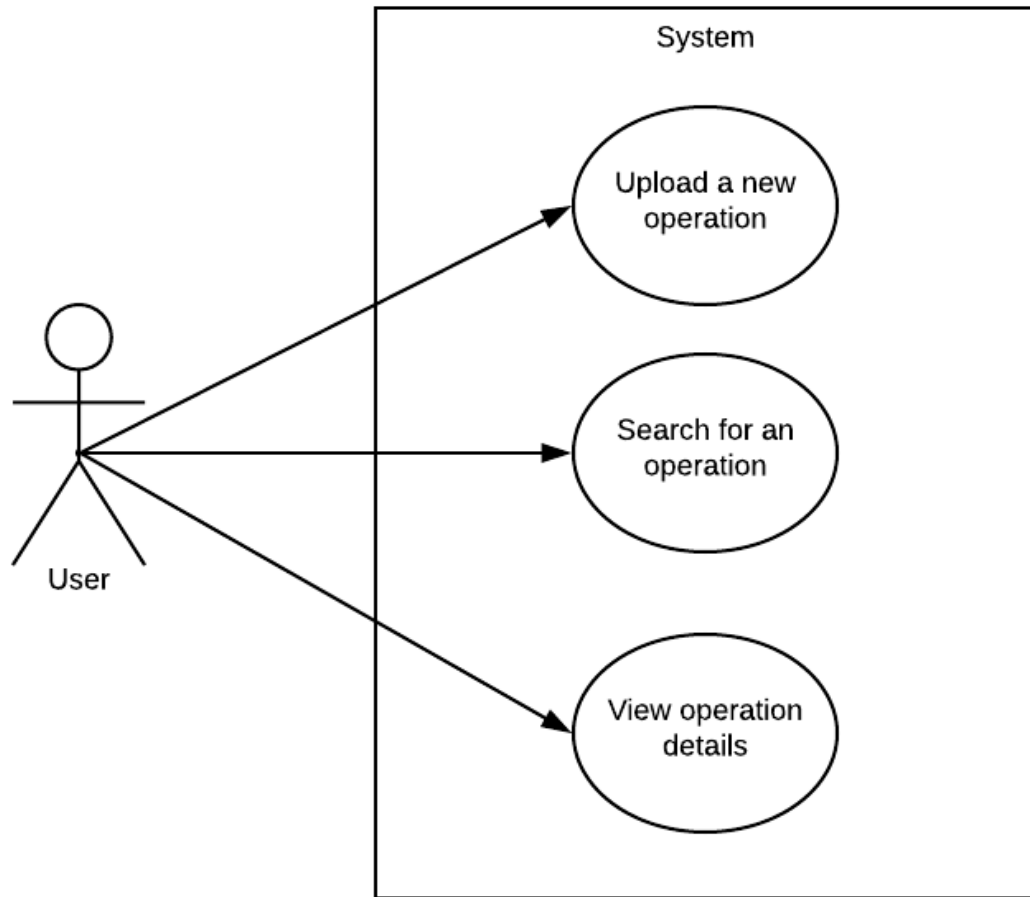
The use case diagram which is depicted in Figure 3.2 is derived from the use case Listing which is basic foundation for the creation of the use case diagram.

## 3.6 Mock-ups

Analysis of the requirements and the use cases resulted in the creation of a series of mock-ups. The reason for creating the mock-ups is to ensure the realization of the requirements and pre-emptively identifying any issues. The mock-ups include only the “must have” and “should have” requirements as the could have requirements entail further investigation for feasibility.

The Use Cases and the analysis of the requirements identify 3 main views that were needed in the project. The first one is the “Add new Operation” View where the user can register a new operation to the system. This means that the user can enter all the details of a specific operation like the





**Figure 3.2:** Use case diagram

hospital and the room that the surgical operation was performed, the patient's name, the participated staff and finally, the user can upload all the files that were produced from the sensors during the operation (video files, audio files & the patient's monitoring file).

The second one is the "Search for an Operation" view which is also the Home Page. This is the first screen that the user sees and it is used to display the 20 most recent uploaded operations, ordered by time. The user can also apply specific filters like hospital, operating room number, specific patient etc., in order to limit the result and find a specific operation of interest.

The third and last view is the "View Operation Details". After the user has selected the desired operation from the "Search for an Operation" page, the operation details page is loaded where the user can see all the relative information of the operation, gathered in one page. The three aforementioned views are presented in the figures below.


Sensor Fusion

Search for an Operation

Add new Operation


Register a new Operation

Select Hospital




Select a hospital... ▼

Operating room




Select a room number... ▼

Select Patient




Select a patient... ▼

Select Staff




Select the staff participated... ▼

Type of Operation



Select the operation type... ▼


Upload the video files



Browse...

No files selected.


Upload the audio files



Browse...


No files selected.

Upload the patient's monitoring file



Browse...

No files selected.

Submit and upload 

**Figure 3.3:** “Add new Operation” View

Sensor Fusion

Search for an Operation

Add new Operation


Hospital

Select a hospital... ▼


Room number

Select a room number... ▼

Operation From

dd/mm/yyyy 

Operation To

dd/mm/yyyy 

Hospital

Select the staff participated... ▼

Patient

Select a patient... ▼

Search

Figure 3.4: “Search for an Operation” View

Sensor Fusion

Search for an Operation

Add new Operation

Operation with Operation ID: xx

Properties	Details
Patient's full name	John Andersson
Hospital	Bernet General Hospital
Operating Room Number	102
Date started	02/08/2018 16:38:04
Duration	45.21 minutes
Type of operation	Endocrine Surgery
Staff participated	Nick Backhouse, Rebecca Anderson
Video source: 1	<a href="https://sensorfusionstorage.blob.core.windows.net/operation1/video1.mp4">https://sensorfusionstorage.blob.core.windows.net/operation1/video1.mp4</a>
Video duration	35.2 minutes
Size	423 Mb
Encoded Date	02/08/2018 16:38:04
Type	Mp4
Video source: 2	<a href="https://sensorfusionstorage.blob.core.windows.net/operation1/video2.mp4">https://sensorfusionstorage.blob.core.windows.net/operation1/video2.mp4</a>
● ● ● ●	
Audio source: 1	<a href="https://sensorfusionstorage.blob.core.windows.net/operation1/audio1.mp3">https://sensorfusionstorage.blob.core.windows.net/operation1/audio1.mp3</a>
● ● ● ●	
Audio source: 2	<a href="https://sensorfusionstorage.blob.core.windows.net/operation1/audio2.mp3">https://sensorfusionstorage.blob.core.windows.net/operation1/audio2.mp3</a>
● ● ● ●	
Patient's monitoring file	<a href="https://sensorfusionstorage.blob.core.windows.net/operation1/monitorFile">https://sensorfusionstorage.blob.core.windows.net/operation1/monitorFile</a>
● ● ● ●	

Figure 3.5: “Operation Details” View

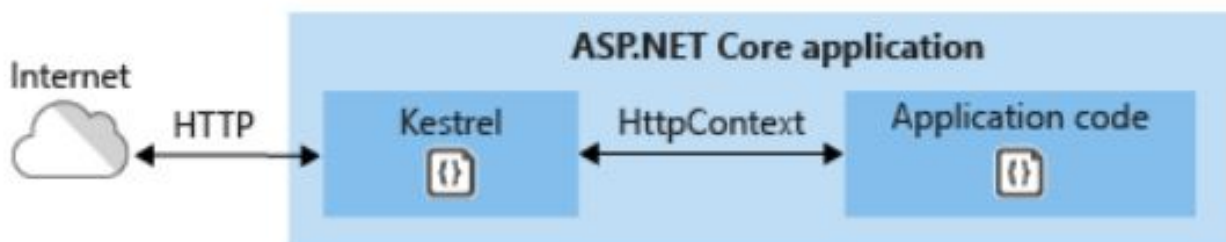
# Chapter 4

## Design and Implementation

The analysis performed in Chapter 3 allows a full outline of the system requirements, the entities of the system and their assigned interactions, and leads onto the development of the full design of the software. The design phase starts with the design and the architecture of the system that is going to be built.

### 4.1 Architecture and System Structure

The system is a typical B/S (Browser/Server) framework with a client browser sending requests and responses and a Kestrel server which is the default cross-platform HTTP server for ASP.NET Core projects. The server is running c# (version 7.0) with a MySQL database. The server implementation listens for HTTP requests and surfaces them to the app as sets of request features composed into an HttpContext [7]. ASP.NET Core communicates with the MySQL database which is hosted in Azure. The structure of ASP.NET Core is shown in figure 4.1.

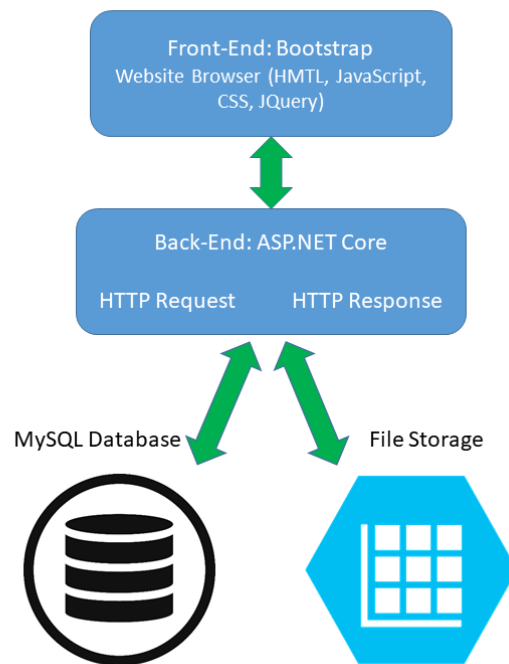


**Figure 4.1:** ASP.NET Structure

The architectural framework is divided into four basic tiers: front-end, back-end, database, and file storage.

1. **Front-End:** the front end is user interface where the user completes a series of operations to control the application which is supported by HTTP request and HTTP response. It contains HTML code, JavaScript, JQuery and CSS)

2. **Back-End** when a static HTTP response is received from the web browser, ASP.NET Core will create an `HttpRequest` object that contains the request data, and invoke the correspondent view to handle this object. After the handle process, it will create and return a new `HttpResponse` object to the front-end view.
3. **Database**: ASP.NET Core controls the MySQL relational database which is hosted in Azure.
4. **File Storage**: The files selected by the user are uploaded to Azure Blob Storage.



**Figure 4.2:** Application General Structure

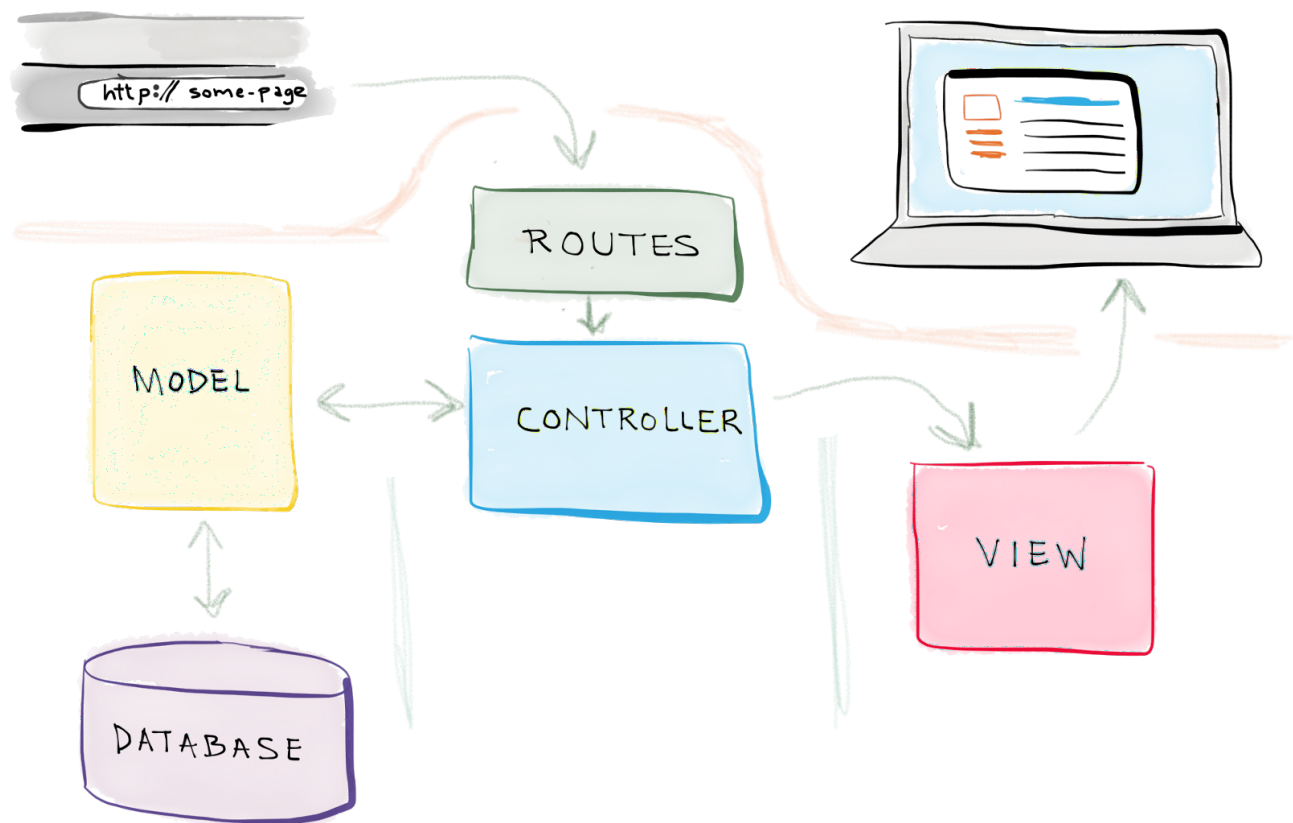
## 4.2 Model-View-Controller Pattern

Due to the time limitations of the project the lack of experience in web designing, it was more appropriate to create a basic user interface and devote the majority of the time to developing a robust back-end. Therefore, it was very important that the architecture of the application made a separation between the user interface and the back-end processing.

MVC framework is one of the most popular design patterns which is motivated by the separation of the UI and the processing performed to generate it. The MVC has been conceptualised for many years, and thus it precedes the inception of web applications and therefore, many efforts at applying the model to web applications through frameworks have been controversial.

Since most of the time has been devoted into developing the back-end than the front-end of the application, it is quite likely that at some point in the future the front-end would be replaced with a more aesthetically pleasing one.

The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns [8]. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires [9].



**Figure 4.3:** The Model-View-Controller Pattern of ASP.NET Core

The server-side MVC (Model-View-Controller) framework (as depicted in figure 4.3 has three core layers:

1. **The Model:** The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it. [9]. This is essentially a library of supporting methods which help the Controller in generating the data to pass to the View. Those methods are especially written to handle and populate data related to the particular View and are based on libraries of more generic functions built to provide helping functions to the Model tier.

2. **The View** Views are responsible for presenting content through the user interface. In ASP.NET Core Views use the Razor view engine which is a compact, expressive and fluid template mark-up language for defining views using embedded C# code. Razor is predominantly used to dynamically generate web content on the server and allows to cleanly mix server code with client side content and code. [9]. Razor view engine is also used for the opposite; embed c# code in HTML mark-up. As a general principle, there should be minimal, if not none, logic within the views, and any logic should be related to presenting the content/model, passed from the Controller.
3. **The Controller:** This is a set of classes that manages the relationship between the View and the Model [10]. It responds to user input, communicates with the Model, and “decided” which view to render and send back to the client side. Essentially, it controls the application logic for a particular unit and is responsible to call all the necessary methods from the Models in order to generate and pass the correct data to the View.

## 4.3 Design Class Diagram

The design class diagram represents a complete overview of the classes within the system, the methods they use and the links between them. One of the main aims of designing the class diagram, is to achieve high cohesion and low coupling. The design class diagram, shown in Figure 4.4 fully details all the internal entities of the system and maps the structure of the components of the software. As explained in subsection 4.2 the controller classes contain methods which are responsible for handling all entities of the system and updating specific states during their life-cycles, depending on the performed action. The notation used for this diagram is based on common industry notation for class diagrams [11] [12].



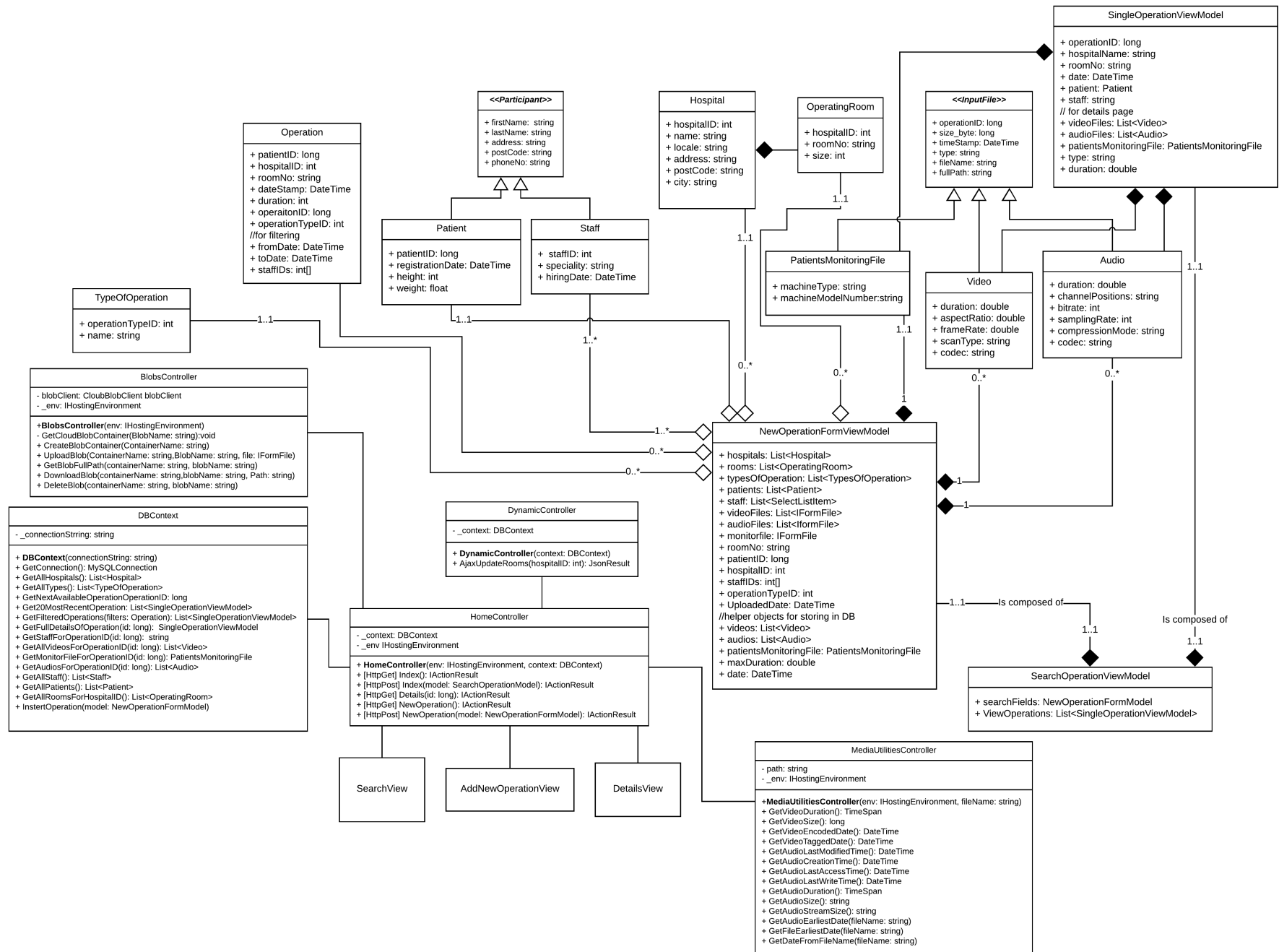


Figure 4.4: Design class diagram.

## **4.4 Database Design**

### **4.4.1 Introduction**

A database is a data repository where the data is managed and stored according to the data structure. It enables data sharing across an organisation, reduces data redundancy and increases data consistency. Database development doesn't have a unique way of implementing, and different data structures require different types of database. In regards to this application, the relational database was selected to represent the data structure. The main purpose of relational databases is to examine how data is related to each other. They translate the complicated data structure and data relationship into simple two-dimensional tables. In the relational model, data and relationships are represented as tables, each of which has a number of columns with a unique name.

Regarding this specific application, it was deemed that a MySQL [13] database is the most suitable relational database management system. MySQL is an easy to use, reliable, scalable and specifically designed and optimized for Web Applications. Although it requires additional configuration when the application is deployed in the server, it is powerful enough to support the extensive complexity that the current application needs.

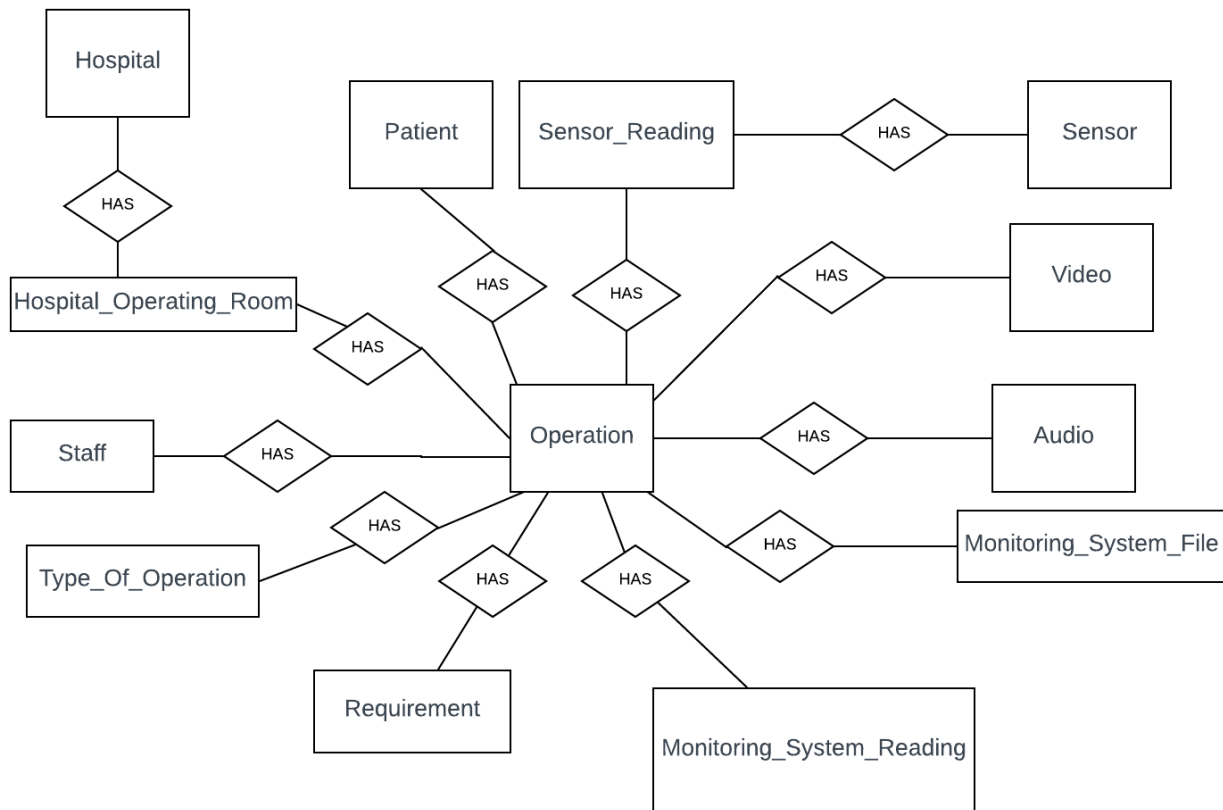
### **4.4.2 Conceptual Schema**

Conceptual modeling or conceptual database design is the process of constructing a model of the information use in an enterprise that is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations. This model is called a conceptual data model [14]. The Conceptual Schema is shown in figure 4.5.

### **4.4.3 Logical Schema**

The logical database design phase maps the conceptual data model on to a logical model, which is influenced by the data model for the target database. The logical data model is a source of information for the physical design phase, providing the physical database designer with a vehicle for making trade-offs that are very important to the design of an efficient database [14]

While the conceptual model is independent of all implementation details, the logical model assumes knowledge of the underlying data model of the target DBMS. The Entity Relationship diagram (Figure 4.6) is created based on the analysis of logical schema. All attributes of every entity have been labelled in the diagram.



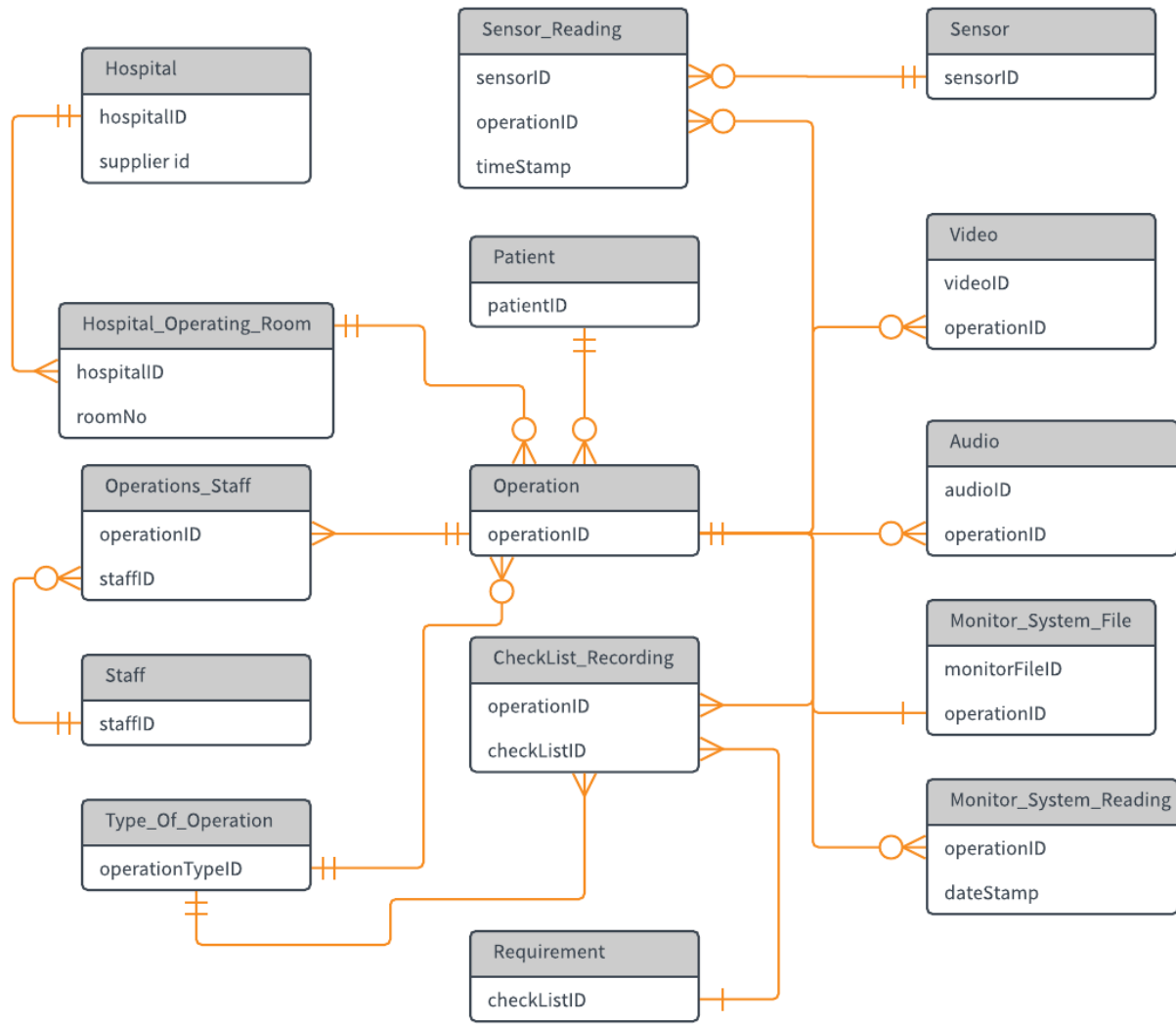
**Figure 4.5:** Conceptual schema

#### 4.4.4 Physical schema

In this final phase of the database design methodology, the logical database design (entities, attributes, relationships, and constraints) has to be translated into a physical database design that can will be implemented using the target DBMS, which in this case in a MySQL database hosted in Azure. Each attribute in the physical schema ( Figure 4.7 ) has constraints which prevent storing invalid data and quickens up the data validation process, requiring no additional code to rewrite the data validation interfaces. The physical database has been especially designed in order to require minimal work for new sensor adding. For example, if a new sensor is added in the future, the schema wouldn't have to be changed and a single row in the "Sensor" table is the only required action. Each sensor reading is recorded to the "Sensor\_Reading" relation.

### 4.5 Pages Implementation

Since, the front-end part of the application is fairly straight-forward, the implementation stage will be split into three subsection which are also the three main views of the application. The first part of the implementation is the page where the user can upload a new operation, the second part is where

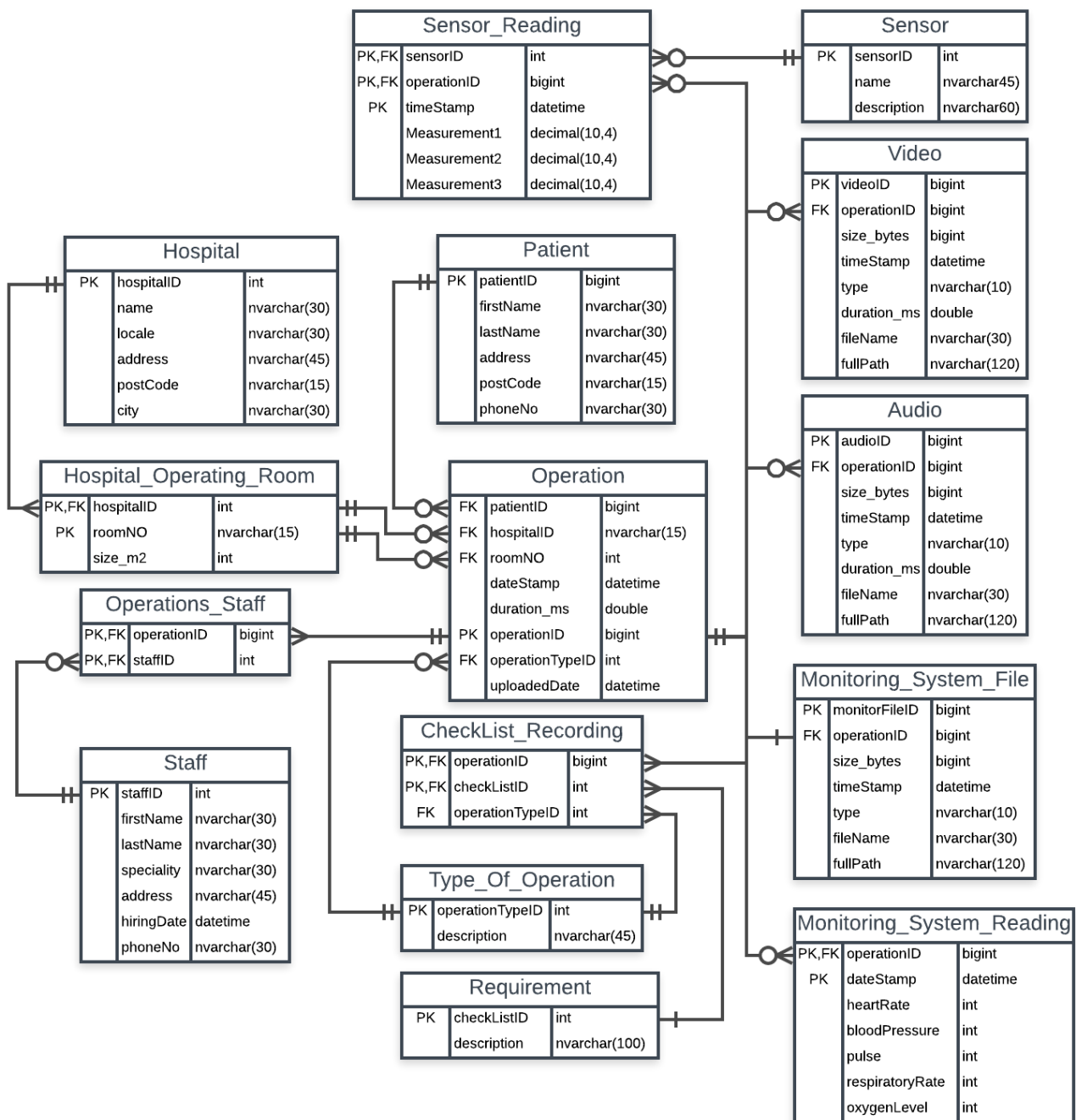


**Figure 4.6:** Entity-Relationship Diagram

the user can make a search for a specific operation stored in the relational database and choose an operation of their choice, and the third page is where the user can see all the details that are related to the specific operation. The application has a single Controller, which as mentioned in 4.2, is an interface between the Model and the View components, processes all the logic and incoming requests, manipulates data using the Model component and interacts with the Views to render the final output. In the HomeController of the application, there are three main methods that correspond to the three main views, which will be analysed in detail in the following subsections.

#### 4.5.1 “Register new operation” Implementation

In this subsection, the process of registering a new operation to the system is explained. The HomeController of the application has two methods with exactly the same names (“New Operation”)



**Figure 4.7:** Physical database schema

that correspond to the `HttpGet` and `HttpPost` request from the client side. When the user selects to load the “Add new Operation” View, the `HttpGet` method is called from the Controller. Due to the fact that this page doesn’t require a already defined model of the application, a new view model had to be created (“`NewOperationViewModel`”). At this stage, the controller instantiates the view model, queries the database to load it and then passes it to the rendered View for display. The `HttpGet` method of the `HomeController` is shown in figure 4.8.

```
[HttpGet]
public IActionResult NewOperation()
{
    var model = new NewOperationFormModel();

    model.typesOfOperation = new SelectList(_context.GetAllTypes().Select(x => new SelectListItem { Value = x.operationTypeID.ToString(), Text = x.name }), "Value", "Text");
    model.staff = new SelectList(_context.GetAllStaff().Select(x => new SelectListItem { Value = x.staffID.ToString(), Text = "ID: " + x.staffID + " " + x.firstName + " " + x.lastName }), "Value", "Text");
    model.hospitals = new SelectList(_context.GetAllHospitals().Select(x => new SelectListItem { Value = x.hospitalID.ToString(), Text = x.name }), "Value", "Text");
    model.patients = new SelectList(_context.GetAllPatients().Select(x => new SelectListItem { Value = x.patientID.ToString(), Text = "ID: " + x.patientID + " " + x.firstName + " " + x.lastName }), "Value", "Text");
    SelectListItem default = new SelectListItem { Text = "Please select a room...", Value = "error", Selected = true };
    List<SelectListItem> defaultSelection = new List<SelectListItem>();
    defaultSelection.Add(default);
    model.rooms = defaultSelection;

    return View(model);
}
```

**Figure 4.8:** NewOperation (HttpGet Request)

After the model is passed to the view, all the information, in regards to the model, will be displayed. The user here, enters all the information related to the specific operation (hospital, operating room, participated staff) and uploads the input files that have been extracted from the sensors. The information entered from the user are passed to the server using the ASP.NET Core MVC Model Binding. Model binding in ASP.NET Core MVC maps data from HTTP requests to action method parameters. When MVC receives an HTTP request, it routes it to a specific action method of a controller. It determines which action method to run based on what is in the route data, then it binds values from the HTTP request to that action method’s parameters [15].

Before the web application posts back to the server, JavaScript validation has to be performed to ensure that the user conforms with the application’s requirements. More specifically, the user has to select a value from all the dropdown menus, and select at least one video file or one audio file or the patient’s monitoring system file. The user interface that the user uploads a new operation is shown in figure 4.9. It is important here to mention that the field “Operating Room” depends on the selection of the specific hospital and cannot be pre-populated. Therefore, a request had to be submitted to the server without posting back and so AJAX had to be used. The section where the client sends a request to the server, in order to query the database and return a JSON object is shown in figure 4.10. When the user has entered all the required fields and selected at least one input file for uploading, a `HttpPost` request is submitted to the server. As previously mentioned, model binding in ASP.NET Core easily binds the data coming from the View to the model, which is then passed back to the controller for manipulation. When the user submits the form, another method with the same name is

SensorFusion
Search for an Operation
Add New Operation
Contact

Register a new Operation

Select Hospital
+
Select a hospital...

Operating Room
🏠
Please select a room...

Select Patient
👤
Select a patient...

Select Staff
👤
Select Staff

Type of Operation
☰
Select the operation type...

Upload the video files
📎
Αναζήτηση...
Δεν επιλέχθηκαν αρχεία.

Upload the audio files
📎
Αναζήτηση...
Δεν επιλέχθηκαν αρχεία.

Upload the patients monitoring system file
📎
Αναζήτηση...
Δεν επιλέχθηκαν αρχεία.

Submit and Upload
➡

© 2018 - SensorFusion

**Figure 4.9:** Register a new operation Page

```

function UpdateRooms(val)
{
    var hospitalID = document.getElementById('HospitalDropDown').value;
    $.ajax({
        type: 'GET',
        dataType: "JSON",
        data: { hospitalID },
        url: 'Url.Action("UpdateRooms", "Dynamic")',
        success: function (result) {
            var dropdown = $('#RoomDropDown');
            dropdown.empty();
            dropdown.append($('

```

**Figure 4.10:** AJAX Call

called (“NewOperation”) which is responsible for the HttpPost requests and takes as input the view model (NewOperationFormViewModel).

After the server-side validation of the model has been successful, the controller must process and manipulate the incoming data, store the input files in the Azure blob storage and insert the new operation to the MySQL relational database. For the purpose of communicating with the database,

an additional class has been created (DBContext). The controller, takes as input the model originated from the View, manipulates the incoming data, and passes the same model to the “DBContext” class for updating the database. This is a very important issue, as the application had to follow the principle of separation of concerns. It is very critical here to mention that only the metadata extracted from the input files were stored to the MySQL database. The files themselves were only stored to a connected file storage account (Microsoft Azure Blob Storage). For this purpose, a new controller had to be created in order to store and retrieve the files from the file storage (BlobsController). In regards to the extraction of the meta-data from the input files, additional NuGet packages and libraries (MediaInfo) had to be installed to the project. Furthermore, a dedicated class for processing, manipulating and extracting metadata from the files had to be created (“MediaUtilities”). This class enables the application to extract information from the input files that would have otherwise been almost impossible to gain. A very small sample of the obtained metadata include the exact starting date, the size of the files, codec used, encoded date, aspect ratio, frame-rate, duration of the media files and many more. The part where the application extracts the metadata from the media files is shown in figures 4.11 and 4.12.

## **4.5.2 “Search for an operation” Implementation**

Following the registration of a new operation, the user can navigate to the home page, where they can search for an operation stored in the relational database. As mentioned in subsection 4.5.1, two methods with the same name were created in the HomeController, one for the HttpGet and one for the HttpPost request. Since no model is sufficient to display the information needed in this page, two more view models had to be created (SearchOperationViewModel & SingleOperationViewModel) which are shown in figure 4.13. The rationale for this implementation decision is that the view had to display two completely different elements; all the filters and a number of operation as a list. Therefore the controller had to pass to the view a single view model that would incorporate two view models as objects. For displaying the filters section, there was no need to create a new view model and thus the pre-existing NewOperationFormViewModel was used.

As implemented in the “add new operation” page, AJAX had to be used for querying the database without posting back to the server. This page, which is also the home page, initially shows the 20 most recently uploaded operations. The user can apply certain filters, such as date, hospital, operating room number, patient, staff, etc. in order to reduce the results and find the specific operation of their choice. As a consistent implementation pattern, when the user clicks the search button, a HttpPost request is submitted to the same method. Then, the “NewOperationFormViewModel” is examined



```

public class VideoInfo
{
    public string Codec { get; private set; }
    public int Width { get; private set; }
    public int Height { get; private set; }
    public double FrameRate { get; private set; }
    public string FrameRateMode { get; private set; }
    public string ScanType { get; private set; }
    public TimeSpan Duration { get; private set; }
    public int Bitrate { get; private set; }
    public string AspectRatioMode { get; private set; }
    public double AspectRatio { get; private set; }
    public string TaggedDate { get; private set; }
    public string EncodedDate { get; private set; }
    public long FileSize { get; private set; }

    public VideoInfo(MediaInfo mi)
    {
        Codec = mi.Get(StreamKind.Video, 0, "Format");
        Width = int.Parse(mi.Get(StreamKind.Video, 0, "Width"));
        Height = int.Parse(mi.Get(StreamKind.Video, 0, "Height"));
        Duration = TimeSpan.FromMilliseconds(int.Parse(mi.Get(StreamKind.Video, 0, "Duration")));
        Bitrate = int.Parse(mi.Get(StreamKind.Video, 0, "BitRate"));
        AspectRatioMode = mi.Get(StreamKind.Video, 0, "AspectRatio/String"); //as formatted string
        AspectRatio = double.Parse(mi.Get(StreamKind.Video, 0, "AspectRatio"));
        FrameRate = double.Parse(mi.Get(StreamKind.Video, 0, "FrameRate"));
        FrameRateMode = mi.Get(StreamKind.Video, 0, "FrameRate_Mode");
        ScanType = mi.Get(StreamKind.Video, 0, "ScanType");
        TaggedDate = mi.Get(StreamKind.Video, 0, "Tagged_Date");
        EncodedDate = mi.Get(StreamKind.General, 0, "Encoded_Date");
        FileSize = Int64.Parse(mi.Get(StreamKind.General, 0, "FileSize"));
    }
}

```

**Figure 4.11:** Extracting meta-data from video files

and processed, in order to ascertain whether the user has filled any of the filter fields. If that is the case, the view model is used to pass it to the database controller (DbContext) and return the results from the database, according to the filters of the user. Finally, the HttpPost method load the same view model (SearchOperationViewModel) which is in turn passed to the view for displaying. The search page of the application is shown in figure 4.14.

### 4.5.3 “Operation Details” Implementation

The last part of the application’s user interface is about displaying all the relevant information of an operation. From the home page, the user clicks on any listed operation and the details page will be loaded. It is important here to highlight that another controller method was deemed appropriate to be called, using the ASP.NET Core routing. Routing in ASP.NET Core MVC is the mechanism through which incoming requests are mapped to controllers and their actions. This is achieved by adding

```

public class AudioInfo
{
    public string Codec { get; private set; }
    public string CompressionMode { get; private set; }
    public string ChannelPositions { get; private set; }
    public TimeSpan Duration { get; private set; }
    public int Bitrate { get; private set; }
    public string BitrateMode { get; private set; }
    public int SamplingRate { get; private set; }
    public string TaggedDate { get; private set; }
    public string EncodedDate { get; private set; }
    public string FileSize { get; set; }
    public string StreamSize { get; set; }

    public AudioInfo(MediaInfo mi)
    {
        Codec = mi.Get(StreamKind.Audio, 0, "Format");
        Duration = TimeSpan.FromMilliseconds(int.Parse(mi.Get(StreamKind.Audio, 0, "Duration")));
        Bitrate = int.Parse(mi.Get(StreamKind.Audio, 0, "BitRate"));
        BitrateMode = mi.Get(StreamKind.Audio, 0, "BitRate_Mode");
        CompressionMode = mi.Get(StreamKind.Audio, 0, "Compression_Mode");
        ChannelPositions = mi.Get(StreamKind.Audio, 0, "ChannelPositions");
        SamplingRate = int.Parse(mi.Get(StreamKind.Audio, 0, "SamplingRate"));
        TaggedDate = mi.Get(StreamKind.General, 0, "Tagged_Date");
        EncodedDate = mi.Get(StreamKind.General, 0, "Encoded_Date");
        FileSize = mi.Get(StreamKind.General, 0, "FileSize");
        StreamSize = mi.Get(StreamKind.Audio, 0, "StreamSize");
    }
}

```

**Figure 4.12:** Extracting meta-data from audio files

Routing middleware to the pipeline and using `IRouteBuilder` to map URL pattern to a controller and action [16].

Therefore, using the routing functionality of ASP.NET Core, the `Details` method is called from the `HomeController`, which takes as input the specific operation id that the user has selected. The method then passes the id to the database controller (`DBContext`) which queries the database and returns all the stored informations related to the specific operation. As in all previous methods, a view model is loaded (in this case the “`SingleOperationViewModel`”) and then passed to the view for displaying.

```

public class SearchOperationViewModel
{
    public NewOperationFormViewModel searchFields { get; set; }
    public IEnumerable<SingleOperationViewModel> ViewOperations { get; set; }
}

public class SingleOperationViewModel
{
    public long operationID { get; set; }
    public string hospitalName { get; set; }
    public string roomNO { get; set; }
    public DateTime date { get; set; }
    public Patient patient { get; set; }
    public string staff { get; set; }
    public List<Video> videoFiles { get; set; }
    public List<Audio> audioFiles { get; set; }
    public PatientsMonitoringFile patientsMonitoringFile { get; set; }
    public string type { get; set; }
    public double duration { get; set; }
}

```

**Figure 4.13:** SearchOperationViewModel & SingleOperationViewModel

SensorFusion
Search for an Operation
Add New Operation
Contact

Hospital  
Select a hospital...

Room Number  
Please select a room...

Operation From  
2018 / 08 / 16

Operation To  
2018 / 08 / 16

Staff  
Select Staff

Patient  
Select a patient...

Search

HOSPITAL	OR NUMBER	PATIENT	DATE	STAFF
University College Hospital	502	Mark Jones	2018-08-16 21:49:00	Mark Courtley, Bethany Johnson
Great Ormond Street Hospital	402	Ege Scott	2018-08-16 16:46:43	Rebecca Anderson
Gordon Hospital	303	Jannis Roberts	2018-08-15 13:18:15	Jason Lorens, Bryan Jackman
Capio Nightingale Hospital	202	Nick James	2018-08-10 13:57:49	Rebecca Anderson, Bethany Johnson
Bernet General Hospital	101	John Andersson	2018-08-01 21:33:15	John Adams, Mark Courtley

**Figure 4.14:** Search for an operation Page

# Chapter 5

## Debugging, Testing and Results Evaluation

In order to ensure that the functionalities of the application have been achieved and abnormal/erroneous interactions have been eliminated, it is very important to debug and test a system as a whole and its individual parts during and thereafter the implementation. Over the development of the project, for testing the correctness, completeness and quality of the application, the test process was divided into three main phases: unit testing [17], responsiveness tests and system testing. Unit testing was performed throughout the development of the application while responsiveness testing and system testing were performed towards the completion of the development stage. In this chapter, the outcomes of all testing procedures will be discussed.

### 5.1 Unit Testing with MSTest

Unit tests are used to check and verify the smallest possible part and component of the application. This component usually is the smallest testable part of the application and it usually has one or a few inputs and a single output. They allow programmers to examine the viability of the code as it is being developed. For example, a test could assert the outcome of an object's instance variables after it has been initialised, or after some other methods have been executed.

As mentioned, unit testing, was performed during the implementation of each of the components of the application and as additional functionalities were added, their individual performance was examined with small unit tests. The specific functionality that was just added, was then tested again, after it has been integrated with the rest of the application, in order to make sure that they functioned properly and didn't obstruct the functionality of any other component.

Microsoft and ASP.NET Core recognizes the big usefulness and importance of unit testing and thus Visual Studio Community 2017 ships with its own unit testing framework, known as MSTest or officially "Visual Studio Unit Testing Framework" which was used for this project. It comes with its own test suite, which provides a professional test guide to help developers write test code for unit testing. In order to facilitate the unit testing of the application, three separate classes were created:

1. **HomeControllerTests.cs:** This class test the functionality of the HomeController, which is the single controller of the application that handles the http requests coming from the browser. The proper functionality of this class was certainly of paramount importance, since the home controller is responsible for instantiating and manipulating the appropriate models and view models. This class has in total five methods and thus the testing class has as well an equal amount of methods. It was very essential for the application that both the HttpGet and HttpPost methods were rigorously tested since their functionality is substantially different.
2. **DBContextTests.cs:** In this class the proper functionality of the database was tested in order to ensure security and the quality of the database. Since the home controller passes object to the DBContext for manipulation the accurate loading of the object had to be precisely ensured. The DBContext has in total 14 submethods and therefore the testing class has an equal amount of test methods.
3. **MediaUtilitiesTests.cs:** This class was created to help the application extract the metadata from the input files and therefore its absolute proper functionality was of high importance. It must have been ensured that this class could provide the right metadata of the media files, at any cases and under all extreme conditions. The bulk of the tests was done on this specific test class.

Following the pattern of true test-driven development (TDD) [18] [19], a developer should follow three rules [20] :

1. “You are not allowed to write any production code until you have first written a failing unit test”
2. “You are not allowed to write more of a unit test than is sufficient to fail-and not compiling is failing”
3. “You are not allowed to write more production code that is sufficient to pass the currently failing unit test.”

Being in line with the aforementioned approach, every time a single test was developed, the testing collection was run to ensure that all the previously created tests were successful and that the newly created test was failing. So, essentially, the tests for each method was created first and then the method in question was implemented. Once the newly implemented method was written, the

collection of tests was run again in order to ensure not only that the specific test passed, but that no other test in the test suite was affected by the newly added method. At the end of the development stage, the tests suite was executed one more time in order to make sure that all written tests successfully passed, which they did. In the figure 5.1 the result of each individual test is shown upon the completion of the test suite.

## **5.2 Responsiveness Tests-Cross Browser Compatibility**

The application aims to be accessible to all doctors/users and all the NHS computers regardless of the browser installed on each computer. Therefore, the system must be responsive on a variety of client browsers. The application was tested for responsiveness on 5 browsers including Google Chrome, Mozilla Firefox, Opera, Internet Explorer and Microsoft Edge.

All pages of the application were tested in this session, one example can be seen in Figure 5.2 which shows how one of the pages (the home page) tested performed on all of the aforementioned browsers. Part of the responsiveness test was also the display functionality such as the “chosen” dropdown menus, multi-select dropdown menus, date-pickers etc. The responsive Bootstrap/CSS styled display was evaluated with acceptable performance on all browsers and only slight visual changes were made during this testing session so that a better user view for all users was possible.

## **5.3 System Testing**

The aim of system testing is to establish that the system meets the functional and non functional requirements defined in Chapter 3 [21]. In the test plan that was followed, the test object in system testing was the whole application. At this phase, the testing method was to invite three volunteers to participate in the test and ask them to finish a sequence of operations according to different scenarios.

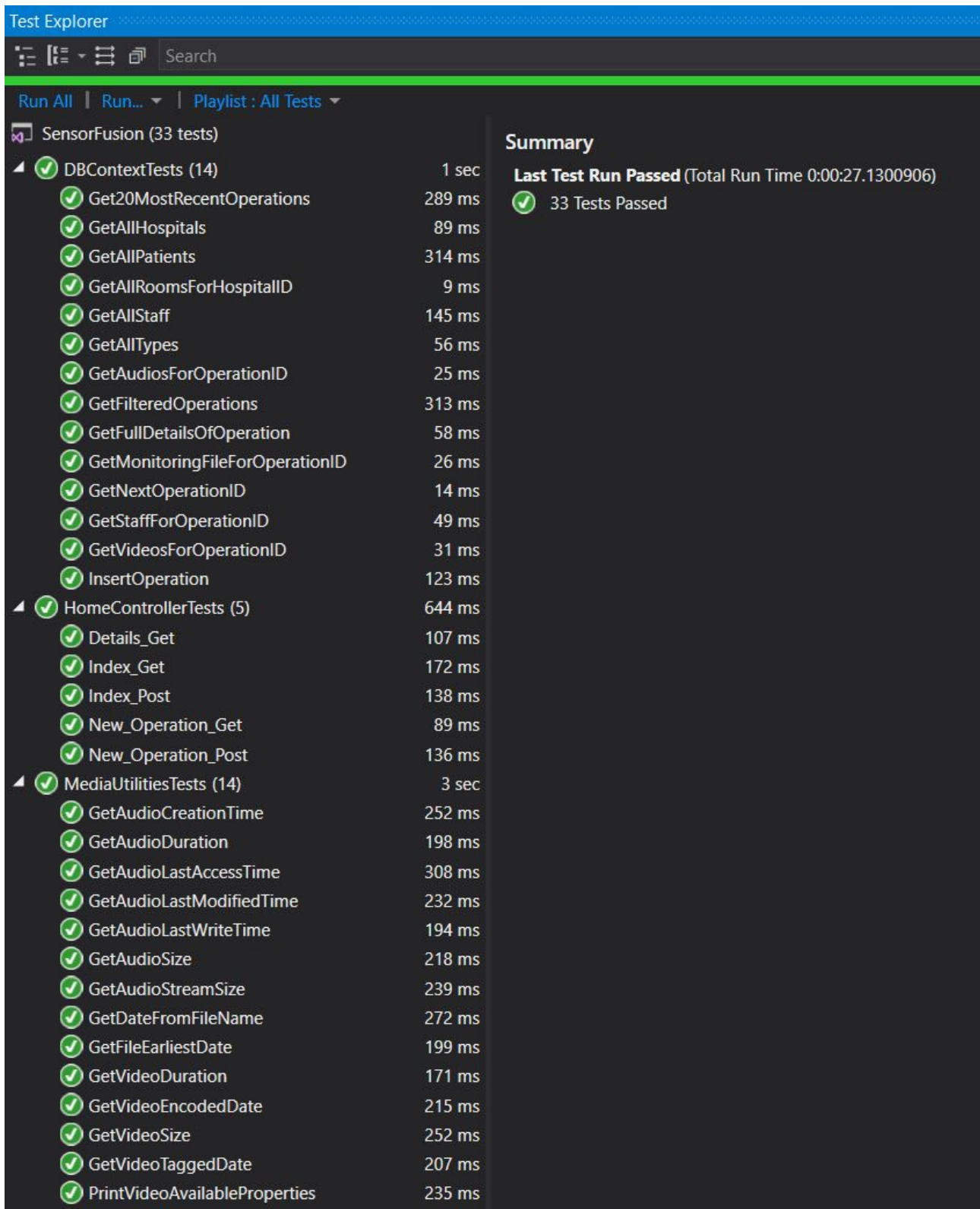
- **Scenario one:**

The volunteers are initially provided with sample operation’s informations and files and are asked to upload the operation to the system. The volunteers can operate in their own way without any constrains or interference.

- **Scenario two:**

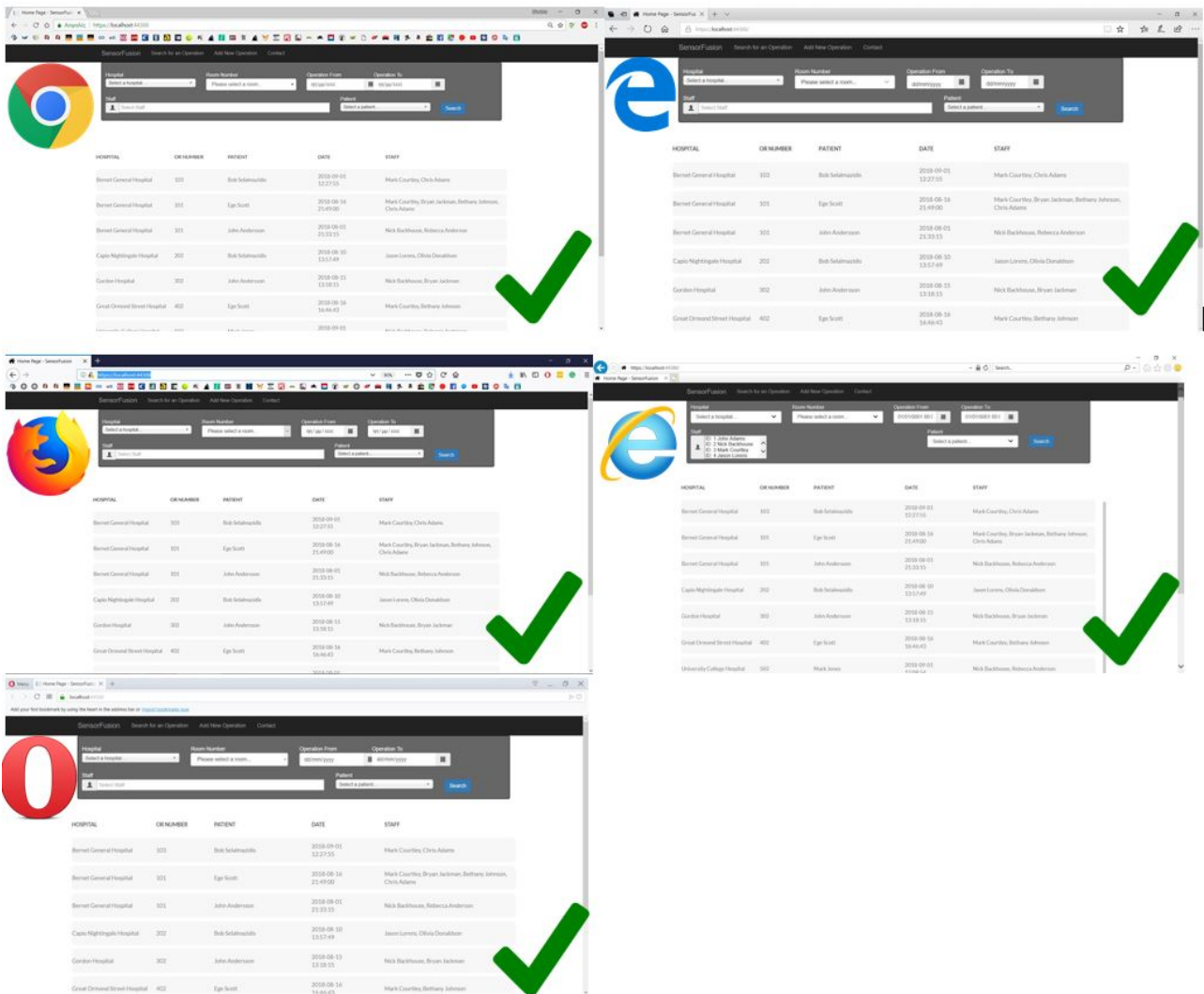
The volunteers are asked to navigate to the home page and start searching for a specific operation using the pre-defined filters. The volunteers are then provided with all the information related to a specific operation and are asked to use the filters in order to find the





**Figure 5.1:** Summarised result from executing the entire test suite for the whole project

stored operation in the database. The volunteers are then asked to load all the relevant information of the operation and view the details of the operation in question.



**Figure 5.2:** Screen shot of how the home page of the application is displayed on different browsers



# Chapter 6

## Conclusion and Project evaluation

This chapter completes the project by revising the project goals, objectives and personal aims in order to assess whether they were met. An evaluation of the project examines the processes and procedures that were implemented to develop and deliver the project. Finally, future work and concluding remarks are presented.

### 6.1 Project Goals

The introductory goals of the project have been defined quite broadly on purpose since there was substantial uncertainty in the initial stage of the project about how quickly the development of the application would proceed. The author is generally satisfied with the results, although some components were more successful than the rest. The first project goal was to develop and deliver a web application with a clear and usable interface that will enable the user to register and upload all the relevant information of an operation, including the generated from the operation media files. Sensor Fusion Web Application has been successfully deployed and tested at <https://sensorfusion.azurewebsites.net/> and all the aforementioned functionalities have been included. The second goal was to build the back-end part of the web application, that will make it able to extract all the meta-data from the selected media files, store the media files in a private file storage and store all the information of the operation in a relational database. That was arguably the hardest part of the application, but the application now fully supports the extraction, storing and retrieval of all the metadata related to the media files. The second goal was to develop a combination of front-end and back-end components that will allow the user to search for an operation stored in the relational database, apply certain filters to reduce the displayed results in order to find an operation of their choice. This goal has also been achieved, as the home page of the application demonstrates the listing and searching capabilities of the web application. The third and final goal of the application was to design a distinct web page in the application where the user is able to view all the aggregated information that is related to a specific operation. The web

application includes a separate “details” page where the user can all the relevant information of an operation, including the media files’ URL and metadata information. Overall, the author’s opinion is that the development of the application went very well. During the development process of the server side, there was never an occasion where where a part of the application had to be restructured or reconstructed in order to make another part of the application work properly, which by itself was a strong sign of separation of concerns.

## **6.2 Fulfilment of Personal Aims**

The personal aims of this project were mostly focused on learning more and becoming proficient in building and developing dynamic web applications. In regards to the server side language, the programming language of choice was C# and ASP.NET Core framework, as described throughout the current report. During the development phase and after the first weeks of learning C# and ASP.NET, it was only able to build static pages and some basic database manipulation. Nonetheless, after a intense period, the author was able to realise the true potential of the framework and from then on the development stage progressed at a rapid pace. Overall, the learning experience was very intensive but at the same time very rewarding, and thus this particular stage can be regarded as one of the most significant personal achievements of the whole project. Feeling confident to develop robust web application in a new programming language like C# and being able to build an application from scratch in the ASP.NET framework was certainly a fulfilment of one of my personal aims. By undertaking this particular project, the front-end programming skills of the author have been significantly improved. After the completion of the project, the author gained a significant amount of knowledge on creating attractive user interfaces and is comfortable to a certain level to understand the proper use of JavaScript and HTML5. Furthermore, the use of jQuery and Ajax methods helped with the development of a firm understanding of how JavaScript can be used in order to create a rich and interactive website. Finally, the last achieved personal aim was to build onn previous knowledge of MySQL database. By creating the web application, it provided the opportunity to further improve into a very advanced level the knowledge that had been gained through the GC04 and COMPGC06 database modules. The application helped develop and refine the skills needed to create arguable the most elaborate, robust and stable database schema explained in detail in Chapter 3.

## **6.3 Critical Evaluation**

This section intends to summarise the successes and failures of the project. Being able to analyse the process after its completion reveals many areas that will be changed in future ASP.NET projects.

First of all, the true importance of unit testing has been fully understood only in the middle of the development process and therefore in a future project it will be performed in a larger scale. In regards to the development stage, there was a significant amount of time spent on .NET and C# tutorials that weren't necessarily related to the project. In future projects, taking into account the substantial knowledge that the author has built in the said technologies, the author will devote considerably more time in the development process rather than learning the technologies. Since the application was predominantly back-end based, the author would start developing the front-end at a much later stage in order to have a clearer idea of what information needs to be displayed before creating the view pages. Finally, on the front end side of the application, extensive research would have been made before using front-end technologies (HTML, JavaScript, Ajax, CSS), something that would have saved a considerable amount of time, whilst probably producing an adequate end result. There should have been a lot greater use of AJAX in order to make the pages load quicker and be more responsive.

## **6.4 Future Work**

Taking into account the initial requirements set for the application, the main functionality that is missing is to present the stored data to the user in a meaningful way. This would include developing a media player that would be able to play all the input files synchronised over time. The user would be able to select a moment and the application would synchronise all the media files and present all the relevant information of the operation at the exact selected moment in time. For example, the user would choose to see all the information at 14.25pm and so the application would tune the media files to this exact moment. The second feature that could be implemented is the ability of the application to extract all the relevant information of the operation automatically from the input files (video files, audio files, patient's monitoring system). That means that the user would only have to upload the generated input files to the system, and all the other information would be extracted from the input files. That would also eliminate the possibility of wrong user input, since the user wouldn't have to manually fill the information (hospital name, operating room, patient etc.).

## **6.5 Conclusion**

In order to deliver a web application that will manipulate media files generated from a surgery room and synchronise them over time is a challenging task. Many expected and unexpected difficulties arose during the project, but the fact that eliminating these difficulties remained exciting and fun rather than frustrating, is certainly a sign for a well chosen topic. Apart from that, the project

produced an application that is simple to navigate as well as aesthetically pleasing. The successes and failures were discussed in the previous section of this chapter and both were equally valuable for the future. Since the project has met most of the requirements and left the researcher with significantly more confidence in developing web applications in the future, it can be regarded as success.

# Appendix A

## Source Code

The following pages include a significant port of the project source code. Nonetheless, all the application's code could not fit in the current report because of its quantity and thus the rest of the code can be found on github at:

<https://github.com/jim1924/Summer-Project-2018-Sensor-Fusion> or in the zip files accompanying this report. The code contained in this chapter, represents the post complicated programming features that have been implemented. Finally, any code that has been referenced throughout the writing of this report is provided in full here. The code listing contains files for each of the main four languages used, which are MySQL, C#, JavaScript and HTML.

### A.1 Database creation

```
drop database Sensor_FusionV1;
CREATE DATABASE Sensor_FusionV1;
USE Sensor_FusionV1;
CREATE TABLE 'Hospital' (
    'hospitalID' int auto_increment,
    'name' nvarchar(30),
    'locale' nvarchar(30),
    'address' nvarchar(45),
    'postCode' nvarchar(15),
    'city' nvarchar(30),
    PRIMARY KEY ('hospitalID')
);
```

#Table structure

```
CREATE TABLE 'Sensor' (
```

```

        'sensorID' int auto_increment ,
        'name' nvarchar(45),
        'description' nvarchar(60),
        PRIMARY KEY ('sensorID')
    );

```

```

CREATE TABLE 'Requirement' (
    'checkListID' int auto_increment ,
    'description' nvarchar(100),
    PRIMARY KEY ('checkListID')
);

```

```

CREATE TABLE 'Staff' (
    'staffID' int auto_increment ,
    'firstName' nvarchar(30),
    'lastName' nvarchar(30),
    'type' nvarchar(15),
    'speciality' nvarchar(30),
    'hiringDate' datetime ,
    'address' nvarchar(45),
    'phoneNo' nvarchar(30),
    PRIMARY KEY ('staffID')
);

```

```

CREATE TABLE 'Type_Of_Operation' (
    'operationTypeID' int auto_increment ,
    'description' nvarchar(45),
    PRIMARY KEY ('operationTypeID')
);

```

```

CREATE TABLE 'Patient' (
    'patientID' bigint auto_increment ,
    'firstName' nvarchar(30),

```

```

        'lastName' nvarchar(30),
        'address' nvarchar(45),
        'postCode' nvarchar(15),
        'phoneNo' nvarchar(30),
        PRIMARY KEY ('patientID')
    );

```

```

CREATE TABLE 'Hospital_Operating_Room' (
    'hospitalID' int,
    'roomNO' nvarchar(15) NOT NULL,
    'size_m2' int,
    PRIMARY KEY ('roomNO', 'hospitalID'),
    FOREIGN KEY (hospitalID) REFERENCES Hospital(hospitalID)
    ON DELETE CASCADE
);

```

```

CREATE TABLE 'Operation' (
    'patientID' bigint,
    'hospitalID' int,
    'roomNO' nvarchar(15),
    'dateStamp' datetime,
    'duration_ms' double,
    'operationID' bigint AUTO_INCREMENT NOT NULL,
    'operationTypeID' int,
    'uploadedDate' datetime,
    PRIMARY KEY ('operationID'),
    FOREIGN KEY (hospitalID)
    REFERENCES Hospital(hospitalID) ON DELETE SET NULL,
    FOREIGN KEY (patientID)
    REFERENCES Patient(patientID) ON DELETE SET NULL,
    FOREIGN KEY (operationTypeID)
    REFERENCES Type_Of_Operation(operationTypeID) ON DELETE SET NULL,

```

```

FOREIGN KEY (roomNO)
REFERENCES Hospital_Operating_Room(roomNO)ON DELETE SET NULL
);

```

```

CREATE TABLE 'Video' (
    'videoID' bigint auto_increment ,
    'operationID' bigint ,
    'size_bytes' int ,
    'type' nvarchar(50),
    'duration_ms' double ,
    'timeStamp' datetime ,
    'fileName' nvarchar(30),
    'fullPath' nvarchar(120),
    PRIMARY KEY ('videoID'),
    FOREIGN KEY (operationID)
    REFERENCES Operation(operationID) ON DELETE SET NULL
);

```

```

CREATE TABLE 'Monitor_System_File' (
    'monitorFileID' bigint auto_increment ,
    'operationID' bigint ,
    'size_bytes' int ,
    'type' nvarchar(50),
    'timeStamp' datetime ,
    'fileName' nvarchar(30),
    'fullPath' nvarchar(120),
    PRIMARY KEY ('monitorFileID'),
    FOREIGN KEY (operationID)
    REFERENCES Operation(operationID) ON DELETE SET NULL
);

```

```

CREATE TABLE 'Audio' (

```



```

        'audioID' bigint auto_increment ,
        'operationID' bigint ,
        'size_bytes' int ,
        'type' nvarchar(50),
        'duration_ms' double ,
        'timeStamp' datetime ,
        'fileName' nvarchar(30),
        'fullPath' nvarchar(120),
PRIMARY KEY ('audioID'),
FOREIGN KEY (operationID)
REFERENCES Operation(operationID) ON DELETE SET NULL
);

```

```

CREATE TABLE 'Sensor_Reading' (
    'sensorID' int AUTO_INCREMENT,
    'operationID' bigint ,
    'timeStamp' datetime ,
    'Measurement1' decimal(10,4),
    'Measurement2' decimal(10,4),
    'Measurement3' decimal(10,4),
PRIMARY KEY ('sensorID', 'timeStamp', 'operationID'),
FOREIGN KEY (sensorID)
REFERENCES Sensor(sensorID) ON DELETE cascade ,
FOREIGN KEY (operationID)
REFERENCES Operation(operationID)ON delete cascade
);

```

```

CREATE TABLE 'CheckList_Recording' (
    'operationID' bigint ,
    'checkListID' int ,
    'operationTypeID' int ,
PRIMARY KEY ('operationID', 'checkListID'),

```

```

FOREIGN KEY (operationTypeID)
REFERENCES Type_Of_Operation(operationTypeID)ON DELETE SET NULL,
FOREIGN KEY (checkListID)
REFERENCES Requirement(checkListID)ON DELETE cascade ,
FOREIGN KEY (operationID)
REFERENCES Operation(operationID)ON DELETE cascade
);

```

```

CREATE TABLE 'Operations_Staff' (
    'operationID' bigint ,
    'staffID' int ,
    PRIMARY KEY ('operationID' , 'staffID' ),
    FOREIGN KEY (staffID)
    REFERENCES Staff(staffID)ON DELETE cascade ,
    FOREIGN KEY (operationID)
    REFERENCES Operation(operationID)ON DELETE cascade
);

```

```

CREATE TABLE 'Monitoring_System_Reading' (
    'operationID' bigint auto_increment ,
    'dateStamp' datetime ,
    'heartRate' int ,
    'bloodPressure' int ,
    'pulse' int ,
    'respiratoryRate' int ,
    'oxygenLevel' int ,
    PRIMARY KEY ('operationID' , 'dateStamp' ),
    FOREIGN KEY (operationID)
    REFERENCES Operation(operationID)ON DELETE cascade
);

```

```

CREATE TABLE 'Requirements_Per_Type' (

```

```

    'operationTypeID ' int ,
    'checkListID ' int ,
PRIMARY KEY ( 'operationTypeID ' , 'checkListID ' ) ,
FOREIGN KEY ( operationTypeID )
REFERENCES Type_Of_Operation ( operationTypeID ) ON DELETE cascade ,
FOREIGN KEY ( checkListID )
REFERENCES CheckList_Recording ( checkListID ) ON DELETE cascade
);

```

### **A.1.1 View Creation**

```

CREATE VIEW twentyOperations
AS SELECT *
FROM operation
order by operationID desc
Limit 20;

```

## **A.2 Home Controller**

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.AspNetCore.Mvc.Rendering;
using SensorFusion.Models;
using SensorFusion.ViewModels;

namespace SensorFusion.Controllers

```

```

{
public class HomeController : Controller
{
DBContext _context;
private IHostingEnvironment _hostingEnvironment;

public HomeController(DBContext context, IHostingEnvironment environment)
{
_context = context;
_hostingEnvironment = environment;
}

[HttpGet]
public IActionResult Index()
{
var searchOperationModel = new SearchOperationViewModel();
var newOperationModel = new NewOperationFormViewModel();

newOperationModel.typesOfOperation = new SelectList(_context.GetAllTypes()
    .Select(x => new SelectListItem {
        Value = x.operationTypeID.ToString(), Text = x.name
    })), "Value", "Text");

newOperationModel.staff = new SelectList(_context.GetAllStaff()
    .Select(x => new SelectListItem {
        Value = x.staffID.ToString(),
        Text = "ID: " + x.staffID + " " + x.firstName + " " + x.lastName
    })), "Value", "Text");

newOperationModel.hospitals = new SelectList(_context.GetAllHospitals())

```

```

.Select(x => new SelectListItem {
Value = x.hospitalID.ToString(), Text = x.name
}), "Value", "Text");

newOperationModel.patients = new SelectList(_context.GetAllPatients()
.Select(x => new SelectListItem {
Value = x.patientID.ToString(),
Text = "ID: " + x.patientID + " " + x.firstName + " " + x.lastName
}), "Value", "Text");

SelectListItem defau = new SelectListItem {
Text = "Please select a room...", Value = "error", Selected = true };

List<SelectListItem> defaultSelection = new List<SelectListItem>();
defaultSelection.Add(defau);
newOperationModel.rooms = defaultSelection;
searchOperationModel.ViewOperations = _context.Get20MostRecentOperations();
searchOperationModel.searchFields = newOperationModel;
return View(searchOperationModel);
}

[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Index(SearchOperationViewModel model)
{
bool hospitalSelected = model.searchFields.hospitalID != 0;
bool roomSelected = !model.searchFields.roomNo.Equals("error");
bool fromDateSelected = !(model.searchFields.fromDate == new DateTime());
bool toDateSelected = !(model.searchFields.toDate == new DateTime());

```

```

bool staffSelected = model.searchFields.staffIDs != null;
bool patientSelected = model.searchFields.patientID != 0;

var searchOperationModel = new SearchOperationViewModel();

var condition1=hospitalSelected || roomSelected || fromDateSelected;
var condition2= toDateSelected || staffSelected || patientSelected;

if (condition1 || condition2)
{
    Operation filters = new Operation();
    filters.fromDate = model.searchFields.fromDate;
    filters.toDate = model.searchFields.toDate;
    if (hospitalSelected)
    {
        filters.hospitalID = model.searchFields.hospitalID;
    }
    if (roomSelected)
    {
        filters.roomNO = model.searchFields.roomNo;
    }
    if (staffSelected)
    {
        filters.staffIDs = model.searchFields.staffIDs;
    }
    if (patientSelected)
    {
        filters.patientID = model.searchFields.patientID;
    }
}

```

```

var newOperationModel = new NewOperationFormViewModel();

newOperationModel.typesOfOperation = new SelectList(_context.GetAllTypes()
.Select(x => new SelectListItem {
Value = x.operationTypeID.ToString(), Text = x.name })), "Value", "Text");

newOperationModel.staff = new SelectList(_context.GetAllStaff()
.Select(x => new SelectListItem { Value = x.staffID.ToString(),
Text = "ID: " + x.staffID + " " + x.firstName + " " + x.lastName
})), "Value", "Text");

newOperationModel.hospitals = new SelectList(_context.GetAllHospitals()
.Select(x => new SelectListItem {
Value = x.hospitalID.ToString(), Text =x.name })), "Value", "Text");

newOperationModel.patients = new SelectList(_context.GetAllPatients()
.Select(x => new SelectListItem {
Value = x.patientID.ToString(),
Text = "ID: " + x.patientID + " " + x.firstName + " " + x.lastName
})), "Value", "Text");

SelectListItem defau = new SelectListItem {
Text="Please select a room...", Value = "error", Selected =true };

List<SelectListItem> defaultSelection = new List<SelectListItem>();
defaultSelection.Add(defau);
newOperationModel.rooms = defaultSelection;
searchOperationModel.searchFields = newOperationModel;

```

```

searchOperationModel.ViewOperations = _context.Get20MostRecentOperations();

searchOperationModel.ViewOperations=_context.GetFilteredOperations(filters)

return View(searchOperationModel);
}

```

```

return RedirectToAction("Index");
}

```

```

[HttpGet]
public IActionResult Details(long id)
{
    SingleOperationViewModel model = _context.GetFullDetailsOfOperation(id);

    if (model == null)
    {
        return RedirectToAction("Index");
    }
    return View(model);
}

```

```

[HttpGet]
public IActionResult NewOperation()
{
    var model = new NewOperationFormViewModel();
}

```



```

model.typesOfOperation = new SelectList(_context.GetAllTypes()
    .Select(x=>new SelectListItem { Value =x.operationTypeID.ToString(),
    Text = x.name })), "Value", "Text");

model.staff = new SelectList(_context.GetAllStaff()
    .Select(x => new SelectListItem { Value = x.staffID.ToString(),
    Text ="ID: "+ x.staffID+ " "+x.firstName +" "+ x.lastName
    })), "Value", "Text");

model.hospitals = new SelectList(_context.GetAllHospitals()
    .Select(x => new SelectListItem { Value = x.hospitalID.ToString(),
    Text = x.name })), "Value", "Text");

model.patients = new SelectList(_context.GetAllPatients()
    .Select(x => new SelectListItem { Value = x.patientID.ToString(),
    Text ="ID: "+x.patientID+" "+ x.firstName+" "+ x.lastName
    })), "Value", "Text");

SelectListItem defau = new SelectListItem {
    Text = "Please select a room...", Value ="error",Selected=true };
List<SelectListItem> defaultSelection=new List<SelectListItem>();
defaultSelection.Add(defau);
model.rooms = defaultSelection;

return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]

```

```

public async Task<IActionResult>NewOperation(NewOperationFormViewModel model)
{

    BlobsController storage = new BlobsController(_hostingEnvironment);
    var path = _hostingEnvironment.WebRootPath;
    long nextID = _context.GetNextOperationID();
    string containerName = "operation" + nextID;
    model.maxDuration = 0;
    model.date = new DateTime(9000, 1, 1);
    if (model.videoFiles != null)
    {
        int i = 1;
        model.videos = new List<Video>();
        foreach (var VideoFile in model.videoFiles)
        {
            if (VideoFile.Length > 0)
            {
                string[] type = VideoFile.ContentType.ToString().Split('/');
                if (!type[0].Equals("video"))
                {
                    continue;
                }
                string videoName = "video" + i + "." + type[1];
                await storage.UploadBlob(containerName, videoName, VideoFile);
                var uploads = Path.Combine(_hostingEnvironment.WebRootPath, "TempFiles");
                var filePath = Path.Combine(uploads, videoName);
                MediaUtilities mediaUtil;
                mediaUtil=new MediaUtilities(_hostingEnvironment,videoName);
                using (var fileStream = new FileStream(filePath, FileMode.Create))
                {
                    await VideoFile.CopyToAsync(fileStream);
                }
            }
        }
    }
}

```

```

string fullVideoPath = storage.GetBlobFullPath(containerName, videoName);

model.videos.Add(new Video() {
    OperationID = nextID,
    size_bytes = mediaUtil.GetVideoSize(),
    timeStamp = mediaUtil.GetVideoEncodedDate(),
    type = type[1],
    duration = mediaUtil.GetVideoDuration().TotalMilliseconds,
    fileName = videoName,
    fullPath = fullVideoPath

});

```

```

if (mediaUtil.GetVideoDuration().TotalMilliseconds > model.maxDuration)
{
    model.maxDuration = mediaUtil.GetVideoDuration().TotalMilliseconds;
}
if (model.date.CompareTo(mediaUtil.GetVideoEncodedDate()) > 0)
{
    model.date = mediaUtil.GetVideoEncodedDate();
}
i++;
}
}
}

```

```

if (model.audioFiles != null)

```

```

{
    int i = 1;
    model.audios = new List<Audio>();
    foreach (var audioFile in model.audioFiles)
    {
        if (audioFile.Length > 0)
        {

            string[] type = audioFile.ContentType.ToString().Split('/');
            if (!type[0].Equals("audio"))
            {
                continue;
            }
            string audioName = "audio" + i + "." + type[1];
            await storage.UploadBlob(containerName, audioName, audioFile);
            var uploads=Path.Combine(_hostingEnvironment.WebRootPath,"TempFiles");
            var filePath = Path.Combine(uploads, audioName);
            MediaUtilities mediaUtil=new MediaUtilities(_hostingEnvironment, audioName);
            using (var fileStream = new FileStream(filePath, FileMode.Create))
            {
                await audioFile.CopyToAsync(fileStream);
            }

            string fullAudioPath = storage.GetBlobFullPath(containerName, audioName);

            model.audios.Add(new Audio()
            {
                OperationID = nextID,
                size_bytes = audioFile.Length,
                timeStamp = mediaUtil.GetAudioEarliestDate(audioFile.FileName),
                type = type[1],
            }
        }
    }
}

```

```

duration = mediaUtil.GetAudioDuration().TotalMilliseconds,
fileName = audioName,
fullPath=fullAudioPath
});
mediaUtil.PrintAudioAvailableProperties();

```

```

if (mediaUtil.GetAudioDuration().TotalMilliseconds > model.maxDuration)
{
model.maxDuration = mediaUtil.GetAudioDuration().TotalMilliseconds;
}
var AudioEarliestDate=mediaUtil.GetAudioEarliestDate(audioFile.FileName);
if (model.date.CompareTo(AudioEarliestDate)>0)
{
model.date = mediaUtil.GetAudioEarliestDate(audioFile.FileName);
}
i++;
}
}
}

```

```

if (model.monitorFile != null)
{
if (model.monitorFile.Length > 0)
{
model.patientsMonitoringFile = new PatientsMonitoringFile();
string[] name = model.monitorFile.FileName.Split('.');
Console.WriteLine("The fucking name is ");
name.ToList().ForEach(Console.WriteLine);
string suffix = name[name.Length - 1];

```

```

string type = model.monitorFile.ContentType.ToString();

string fileMonitorName = "patients-monitoring-file" + "." + suffix;
await storage.UploadBlob(containerName, fileMonitorName, model.monitorFile);
var uploads = Path.Combine(_hostingEnvironment.WebRootPath, "TempFiles");
var filePath = Path.Combine(uploads, fileMonitorName);
MediaUtilities mediaUtil;
mediaUtil = new MediaUtilities(_hostingEnvironment, fileMonitorName);
using (var fileStream = new FileStream(filePath, FileMode.Create))
{
    await model.monitorFile.CopyToAsync(fileStream);
}

string fullFilePath;
fullFilePath = storage.GetBlobFullPath(containerName, fileMonitorName);

model.patientsMonitoringFile.OperationID = nextID;
model.patientsMonitoringFile.size_bytes = model.monitorFile.Length;
var value = mediaUtil.GetFileEarliestDate(model.monitorFile.FileName);
model.patientsMonitoringFile.timeStamp = value;
model.patientsMonitoringFile.type = type;
model.patientsMonitoringFile.fileName = fileMonitorName;
model.patientsMonitoringFile.fullPath = fullFilePath;

}

}

Console.WriteLine("the earliest recorded date is:" + model.date);

MediaUtilities.CleanTempFolder(_hostingEnvironment);

```

```

_context.InsertOperation(model);

string msg="<script>alert('Operation uploaded successfully');</script>";
TempData["msg"] = msg;

return RedirectToAction(nameof(Index));

}

```

```

public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    return View();
}

```

```

public IActionResult Privacy()
{
    return View();
}

```

```

}

}

```

## A.3 Database Context

```

using MySql.Data.MySqlClient;
using SensorFusion.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace SensorFusion.Models
{

    //this class handles the interactin between
    // the application and the relational database
    public class DBContext
    {
        public string _ConnectionString { get; set; }

        //gets the connection string
        public DBContext(string connectionString)
        {
            this._ConnectionString = connectionString;
        }
        private MySqlConnection GetConnection()
        {
            return new MySqlConnection(_ConnectionString);
        }
        //this method returns all the hospitals
        //stored in the database
        public IEnumerable<Hospital> GetAllHospitals()
        {
            List<Hospital> list = new List<Hospital>();

```



```

using (MySqlConnection conn = GetConnection())
{
    conn.Open();
    MySqlCommand cmd=new MySqlCommand("SELECT * FROM hospital",conn);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            list.Add(new Hospital()
            {
                hospitalID = reader.GetInt32("hospitalID"),
                name = reader.GetString("name"),
                address = reader.GetString("address"),
                postCode = reader.GetString("postCode"),
                city = reader.GetString("city")
            });
        }
    }
}

return list;
}

```

```

//this method returns all the available types of
//operatins that are stored in the database
public IEnumerable<TypeOfOperation> GetAllTypes()
{
    List<TypeOfOperation> list = new List<TypeOfOperation>();

```

```

using (MySqlConnection conn = GetConnection())
{

```

```

conn.Open();
MySQLCommand cmd =new MySQLCommand("SELECT * FROM Type_Of_Operation",conn);
using (MySQLDataReader reader = cmd.ExecuteReader())
{
    while (reader.Read())
    {
        list.Add(new TypeOfOperation()
        {
            operationTypeID = reader.GetInt32("operationTypeID"),
            name = reader.GetString("description")
        });
    }
}

return list;

}

public long GetNextOperationID()
{
    long next = 0;
    using (MySQLConnection conn = GetConnection())
    {
        conn.Open();
        string autoInc="SELECT Auto_increment "+
        "FROM information_schema.tables WHERE table_name='operation '"
        MySQLCommand cmd = new MySQLCommand(autoInc , conn);
        using (MySQLDataReader reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {

```

```

next = reader.GetInt64("Auto_increment");

}

}

}

return next;

}

//this method returns the 20 most recend operations
// from the database
public IEnumerable<SingleOperationViewModel> Get20MostRecentOperations()
{

List<SingleOperationViewModel> list=new List<SingleOperationViewModel>();

using (MySqlConnection conn = GetConnection())
{
conn.Open();
MySQLCommand cmd = new MySQLCommand(
"select twentyoperations.operationID ,hospital.name AS 'Hospital Name' "
" ,hospital_operating_room.roomNO,"+
"twentyoperations.dateStamp ,patient.firstName"+
" AS 'Patients first name',patient.lastName AS "+
"'Patients last name',patient.patientID " +
" from twentyoperations inner join hospital "+
" ON twentyoperations.hospitalID = hospital.hospitalID " +
" inner join hospital_operating_room "+
" ON twentyoperations.roomNO = hospital_operating_room.roomNO " +
"inner join patient ON twentyoperations.patientID=patient.patientID ",conn)

```

```

using (MySqlDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
{
SingleOperationViewModel operation = new SingleOperationViewModel();
operation.date = (DateTime) reader.GetMySqlDateTime("dateStamp");
operation.hospitalName = reader.GetString("Hospital Name");
operation.operationID = reader.GetInt64("operationID");
operation.patient = new Patient();
operation.patient.firstName = reader.GetString("Patients first name");
operation.patient.lastName = reader.GetString("Patients last name");
operation.patient.patientID=reader.GetInt64("patientID");
operation.roomNO = reader.GetString("roomNO");

operation.staff = GetStaffForOperationID(operation.operationID);
list.Add(operation);
}
}
}

return list;
}

//this method takes as input the filters entered from the user
//and returns the results that comply to the user's criteria
public IEnumerable<SingleOperationViewModel>GetFilteredOperations(Operation
{
List<SingleOperationViewModel> list = new List<SingleOperationViewModel>();

using (MySqlConnection conn = GetConnection())
{
conn.Open();

```

```

MySQLCommand cmd = conn.CreateCommand();
string staffQuery = "";
if (filters.staffIDs != null)
{
    string staffNumbers = filters.staffIDs[0].ToString();
    for (int i = 1; i < filters.staffIDs.Count(); i++)
    {
        staffNumbers = staffNumbers + "," + filters.staffIDs[i].ToString();
    }
    staffNumbers = "(" + staffNumbers + ")";
    staffQuery = "AND operation.operationID in "+
    "( select operations_staff.operationID FROM operations_staff "+
    "WHERE operations_staff.staffID in " + staffNumbers + " "+
    "group by operations_staff.operationID  having "+
    " count(operation.operationID)="+filters.staffIDs.Count().ToString()+")";
}

```

```

cmd.CommandText =
"select operation.operationID ,hospital.name AS 'Hospital Name', "+
" hospital_operating_room.roomNO,operation.dateStamp,"+
"patient.firstName AS 'Patients first name'," +
"patient.lastName AS 'Patients last name',patient.patientID" +
" from operation inner join hospital "+
" ON operation.hospitalID = hospital.hospitalID" +
" inner join hospital_operating_room ON "+
" operation.roomNO = hospital_operating_room.roomNO" +
" inner join patient ON operation.patientID = patient.patientID" +
" where (operation.hospitalID=?hospitalID OR ?hospitalID=0) AND "+
"(operation.roomNO=?roomNO OR ?roomNO IS NULL) AND "+
"(operation.dateStamp>?fromDate OR ?fromDate IS NULL)" +
" AND (operation.dateStamp<?toDate OR ?toDate IS NULL) AND "+

```

```

"(operation.patientID=?patientID OR ?patientID=0) "+ staffQuery;

var value1=(filters.hospitalID != 0) ? filters.hospitalID : 0;
var value2=(filters.roomNO != null) ? filters.roomNO : null;
var bool1=filters.fromDate != new DateTime();
var bool2=filters.toDate != new DateTime();
var par1=(bool1) ? filters.fromDate.ToString("yyyy-MM-dd HH:mm:ss") : null;
var par2=(bool2) ? filters.toDate.ToString("yyyy-MM-dd HH:mm:ss") : null

var par3=(filters.patientID != 0) ? filters.patientID : 0;

cmd.Parameters.AddWithValue("?hospitalID", value1);
cmd.Parameters.AddWithValue("?roomNO", value2);
cmd.Parameters.AddWithValue("?fromDate", par1);
cmd.Parameters.AddWithValue("?toDate", par2 );
cmd.Parameters.AddWithValue("?patientID", par3 );

using (MySqlDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
{
SingleOperationViewModel operation = new SingleOperationViewModel();
operation.date = (DateTime)reader.GetMySqlDateTime("dateStamp");
operation.hospitalName = reader.GetString("Hospital Name");
operation.operationID = reader.GetInt64("operationID");
operation.patient = new Patient();
operation.patient.firstName = reader.GetString("Patients first name");
operation.patient.lastName = reader.GetString("Patients last name");

```

```

operation.patient.patientID = reader.GetInt64("patientID");
operation.roomNO = reader.GetString("roomNO");
operation.staff = GetStaffForOperationID(operation.operationID);
list.Add(operation);

}

}

}

return list;

}

//this method takes as input the operation id and
//returns all the available information of an operation
public SingleOperationViewModel GetFullDetailsOfOperation(long id)
{
SingleOperationViewModel operation = new SingleOperationViewModel();
using (MySQLConnection conn = GetConnection())
{
conn.Open();
MySQLCommand cmd = new MySQLCommand(
"select operation.operationID , operation.duration_ms ,"+
" hospital.name AS 'Hospital Name', hospital_operating_room.roomNO, "+
"operation.dateStamp, patient.firstName AS 'Patients first name', "+
" patient.lastName AS 'Patients last name',"+
" patient.patientID , type_of_operation.description" +
" from operation inner join hospital ON "+
"operation.hospitalID = hospital.hospitalID" +
" inner join hospital_operating_room ON"+
" operation.roomNO = hospital_operating_room.roomNO" +

```

```

" inner join patient ON operation.patientID = patient.patientID" +
" inner join type_of_operation ON "+
"operation.operationTypeID = type_of_operation.operationTypeID" +
" where operation.operationID='" +id+"'", conn);
using (MySqlDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
{
operation.audioFiles = GetAudiosForOperationID(id);
operation.staff = GetStaffForOperationID(id);
operation.videoFiles = GetVideosForOperationID(id);
operation.patientsMonitoringFile = GetMonitoringFileForOperationID(id);
operation.date = (DateTime)reader.GetMySqlDateTime("dateStamp");
operation.hospitalName = reader.GetString("Hospital Name");
operation.operationID = reader.GetInt64("operationID");
operation.patient = new Patient();
operation.patient.firstName = reader.GetString("Patients first name");
operation.patient.lastName = reader.GetString("Patients last name");
operation.patient.patientID = reader.GetInt64("patientID");
operation.roomNO = reader.GetString("roomNO");
operation.type = reader.GetString("description");
operation.duration = (double)reader.GetInt64("duration_ms") / 1000 / 60;

}
}
}

return operation;

```



```

}
//this method returns all the participated
//staff of a specific operation
public string GetStaffForOperationID(long id)
{
    List<Staff> list = new List<Staff>();

    using ( MySqlConnection conn = GetConnection() )
    {
        conn.Open();
        MySqlCommand cmd = new MySqlCommand(
            "select * from operations_staff inner join staff on "+
            "operations_staff.staffID=staff.staffID "+
            "WHERE operations_staff.operationID='" + id + "'", conn);
        using ( MySqlDataReader reader = cmd.ExecuteReader() )
        {
            while ( reader.Read() )
            {
                Staff objectstaff = new Staff();
                objectstaff.staffID = reader.GetInt32("staffID");
                objectstaff.firstName = reader.GetString("firstName");
                objectstaff.lastName = reader.GetString("lastName");
                list.Add(objectstaff);
            }
        }
        string staff= list[0].firstName + " " + list[0].lastName;
        for (int i = 1; i < list.Count; i++)
        {
            staff = staff + ", " + list[i].firstName+" "+ list[i].lastName;
        }
    }
}

```

```

}
return staff;

}
//this method returns all the videos that correspond
//to a specific operaiton
public List<Video> GetVideosForOperationID(long id)
{
List<Video> list = new List<Video>();

using ( MySqlConnection conn = GetConnection())
{
conn.Open();
MySQLCommand cmd = new MySQLCommand(
"select * from video WHERE video.operationID='" + id + "'", conn);
using ( MySQLDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
{
Video video = new Video();
video.fullPath = reader.GetString("fullPath");
video.size_bytes = reader.GetInt64("size_bytes");
video.fileName = reader.GetString("fileName");
video.duration = reader.GetInt64("duration_ms");
video.timeStamp = (DateTime)reader.GetMySqlDateTime("timeStamp");
video.type = reader.GetString("type");
list.Add(video);
}
}
}

```

```

return list;

}

//this method returns the specific patient's monitoring file
public PatientsMonitoringFile GetMonitoringFileForOperationID(long id)
{

PatientsMonitoringFile patientsFile = new PatientsMonitoringFile();
using (MySqlConnection conn = GetConnection())
{
conn.Open();
MySQLCommand cmd = new MySQLCommand(
"select * from monitor_system_file "+
"WHERE monitor_system_file.operationID='" + id + "'", conn);
using (MySQLDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
{

patientsFile.fullPath = reader.GetString(" fullPath");
patientsFile.size_bytes = reader.GetInt64(" size_bytes");
patientsFile.fileName = reader.GetString(" fileName");
patientsFile.timeStamp = (DateTime)reader.GetMySqlDateTime(" timeStamp");
patientsFile.type = reader.GetString(" type");
}
}
}
return patientsFile;

}

//this method returns all the audio files
//that are linked to the specific operation

```

```

public List<Audio> GetAudiosForOperationID(long id)
{
    List<Audio> list = new List<Audio>();

    using (MySqlConnection conn = GetConnection())
    {
        conn.Open();
        MySqlCommand cmd = new MySqlCommand(
            "select * from audio WHERE audio.operationID='" + id + "'", conn);
        using (MySqlDataReader reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                Audio audio = new Audio();
                audio.fullPath = reader.GetString("fullPath");
                audio.size_bytes = reader.GetInt64("size_bytes");
                audio.fileName = reader.GetString("fileName");
                audio.duration = reader.GetInt64("duration_ms");
                audio.timeStamp = (DateTime)reader.GetMySqlDateTime("timeStamp");
                audio.type = reader.GetString("type");
                list.Add(audio);
            }
        }
    }

    return list;
}

//this method returns all the staff stored in the database
public IEnumerable<Staff> GetAllStaff()
{

```

```

List<Staff> list = new List<Staff>();

using (MySqlConnection conn = GetConnection())
{
    conn.Open();
    MySqlCommand cmd = new MySqlCommand("SELECT * FROM staff", conn);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
        while (reader.Read())
        {
            Staff staff = new Staff();
            staff.staffID = reader.GetInt32("staffID");
            staff.firstName = reader.GetString("firstName");
            staff.lastName = reader.GetString("lastName");
            staff.address = reader.GetString("address");
            staff.hiringDate = reader.GetDateTime("hiringDate");
            staff.phoneNo = reader.GetString("phoneNo");
            if (reader.IsDBNull(5))
            {
                staff.speciality = reader.GetString("speciality");
            }

            list.Add(staff);

        }
    }

    return list;
}

```

```

//this method returns all the patients stored in the database
public IEnumerable<Patient> GetAllPatients()
{

    List<Patient> list = new List<Patient>();

    using (MySqlConnection conn = GetConnection())
    {
        conn.Open();
        MySqlCommand cmd = new MySqlCommand("SELECT * FROM patient", conn);
        using (MySqlDataReader reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                list.Add(new Patient()
                {
                    patientID = reader.GetInt64("patientID"),
                    firstName = reader.GetString("firstName"),
                    lastName = reader.GetString("lastName"),
                    address = reader.GetString("address"),
                    postCode = reader.GetString("postCode"),
                    phoneNo = reader.GetString("phoneNO")
                });
            }
        }

        return list;
    }
}

```

```

//this method returns a list of all the operating rooms
//given a hospital ID
public List<OperatingRoom> GetAllRoomsForHospitalID(int hospitalID)
{

List<OperatingRoom> list = new List<OperatingRoom>();

using (MySqlConnection conn = GetConnection())
{
conn.Open();
MySQLCommand cmd = new MySQLCommand("SELECT * FROM hospital_operating_room"
" WHERE hospitalID='" + hospitalID + "'", conn);
using (MySqlDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
{
list.Add(new OperatingRoom()
{
hospitalID = reader.GetInt32("hospitalID"),
roomNO = reader.GetString("roomNO"),
size = reader.GetInt32("size_m2")

});
}
}
}
return list;

}

//this method takes as input a NewOperationFormViewModel
//gets all the information and inserts it in the database

```

```

public void InsertOperation(NewOperationFormViewModel model)
{
    model.UploadedDate = new DateTime();
    model.UploadedDate = DateTime.Now;
    using (SqlConnection conn = GetConnection())
    {
        conn.Open();
        //insert the operation to the database
        MySqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "INSERT INTO operation "+
        "(patientID ,hospitalID ,roomNO,dateStamp ,"+
        "duration_ms ,operationTypeID ,uploadedDate) "+
        "VALUES (?patientID ,?hospitalID ,?roomNo ,?date ,"+
        "?maxDuration ,?operationTypeID ,?uploadedDate)";

        cmd.Parameters.AddWithValue("?patientID", model.patientID);
        cmd.Parameters.AddWithValue("?hospitalID", model.hospitalID);
        cmd.Parameters.AddWithValue("?roomNO", model.roomNo);
        var par1=model.date.ToString("yyyy-MM-dd HH:mm:ss");
        cmd.Parameters.AddWithValue("?date", par1);
        cmd.Parameters.AddWithValue("?maxDuration", model.maxDuration);
        cmd.Parameters.AddWithValue("?operationTypeID", model.operationTypeID);
        var par2=model.UploadedDate.ToString("yyyy-MM-dd HH:mm:ss");
        cmd.Parameters.AddWithValue("?uploadedDate", par2);
        cmd.ExecuteNonQuery();
        cmd.Parameters.Clear();

        long operationID=0;
        cmd.CommandText = "SELECT 'AUTO_INCREMENT' from ";

```



```

"INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'sensor_fusionv1' AND"+
"    TABLE_NAME    = 'operation'";
using (MySqlDataReader reader = cmd.ExecuteReader())
{
    while (reader.Read())
    {
        operationID = reader.GetInt64("AUTO_INCREMENT") - 1;
    }
}

if (model.videos != null)
{
    foreach (var video in model.videos)
    {
        cmd.CommandText = "INSERT INTO video "+
            "(operationID , size_bytes , timeStamp , type , duration_ms , fileName , fullPath) VALUES (" +
            "?operationID ,? size_bytes ,? timeStamp ,? type ,? duration ,? fileName ,? fullPath)";
        cmd.Parameters.AddWithValue("?operationID", operationID);
        cmd.Parameters.AddWithValue("?size_bytes", video.size_bytes);
        cmd.Parameters.AddWithValue("?timeStamp", video.timeStamp);
        cmd.Parameters.AddWithValue("?type", video.type);
        cmd.Parameters.AddWithValue("?duration", video.duration);
        cmd.Parameters.AddWithValue("?fileName", video.fileName);
        cmd.Parameters.AddWithValue("?fullPath", video.fullPath);

        cmd.ExecuteNonQuery();
        cmd.Parameters.Clear();
    }
}

if (model.audios != null)
{

```

```

foreach (var audio in model.audios)
{
    Console.WriteLine(audio.fileName);
    cmd.CommandText = "INSERT INTO audio "+
        "(operationID ,size_bytes ,timeStamp ,type ,duration_ms ,fileName ,fullPath) VALU
        +" (?operationID ,? size_bytes ,? timeStamp ,? type ,? duration ,? fileName ,? fullPath
    cmd.Parameters.AddWithValue("?operationID", operationID);
    cmd.Parameters.AddWithValue("?size_bytes", audio.size_bytes);
    cmd.Parameters.AddWithValue("?timeStamp", audio.timeStamp);
    cmd.Parameters.AddWithValue("?type", audio.type);
    cmd.Parameters.AddWithValue("?duration", audio.duration);
    cmd.Parameters.AddWithValue("?fileName", audio.fileName);
    cmd.Parameters.AddWithValue("?fullPath", audio.fullPath);

    cmd.ExecuteNonQuery();
    cmd.Parameters.Clear();

}
}
if (model.patientsMonitoringFile != null)
{
    cmd.CommandText = "INSERT INTO monitor_system_file "+
        "(operationID ,size_bytes ,timeStamp ,type ,fileName ,fullPath) VALUES "+
        "(?operationID ,? size_bytes ,? timeStamp ,? type ,? fileName ,? fullPath)";
    cmd.Parameters.AddWithValue("?operationID", operationID);
    var par1=model.patientsMonitoringFile.size_bytes;
    var par2=model.patientsMonitoringFile.timeStamp;
    var par3=model.patientsMonitoringFile.type;
    var par4=model.patientsMonitoringFile.fileName;

```

```

var par5=model.patientsMonitoringFile.fullPath;
cmd.Parameters.AddWithValue("?size_bytes", par1 );
cmd.Parameters.AddWithValue("?timeStamp",par2 );
cmd.Parameters.AddWithValue("?type",par3 );
cmd.Parameters.AddWithValue("?fileName", par4);
cmd.Parameters.AddWithValue("?fullPath", par5);
cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
}

foreach (var id in model.staffIDs)
{
cmd.CommandText = "INSERT INTO operations_staff "+
"(operationID ,staffID) VALUES (?operationID ,? staffID)";
cmd.Parameters.AddWithValue("?staffID", id);
cmd.Parameters.AddWithValue("?operationID", operationID);
cmd.ExecuteNonQuery();
cmd.Parameters.Clear();
}
}
}
}

```

## A.4 File Storage Controller

```

using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;

```

```

using System.IO;
using Microsoft.Extensions.Configuration;
using Microsoft.AspNetCore.Http;
using System;
using Microsoft.AspNetCore.Hosting;

namespace SensorFusion.Controllers
{
    public class BlobsController : Controller
    {
        CloudBlobClient blobClient;
        private IHostingEnvironment _env;

        //this is the constructor of the blobs class
        //It gets as input the IHostingEnvironment object
        public BlobsController(IHostingEnvironment env)
        {
            _env = env;
            var builder = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json");
            IConfigurationRoot Configuration = builder.Build();
            var firstPart="ConnectionStrings:sensorfusionstorage";
            var connection=firstPart+"_AzureStorageConnectionString";
            CloudStorageAccount storageAccount = CloudStorageAccount
                .Parse(Configuration[connection]);
            blobClient = storageAccount.CreateCloudBlobClient();
        }

        //this method return a CloudBlobContainer
        private CloudBlobContainer GetCloudBlobContainer(string blobName)
        {

```

```

        CloudBlobContainer container = blobClient
            .GetContainerReference(blobName);

        return container;
    }

//THIS METHOD IS CALLING THE GetCloudBlobContainer.
//So this method is the entry point to create the container.
public async Task<ActionResult> CreateBlobContainer(string containerName)
{
    //Here we're calling the GetCloudBlobContainer method
    // (which takes the container name and return the equivalent container)
    CloudBlobContainer container = GetCloudBlobContainer(containerName);

    //check if the container exists and create a new if it doesn't
    ViewBag.Success = container.CreateIfNotExistsAsync().Result;

    //Update the view bag with the name of the blob container
    ViewBag.BlobContainerName = container.Name;
    BlobContainerPermissions permissions = await container
        .GetPermissionsAsync();
    permissions.PublicAccess = BlobContainerPublicAccessType.Container;
    await container.SetPermissionsAsync(permissions);

    return View();
}

//this method gets as input a specific container name and a specific
//blob name and a specific file name and upload it to azure storage

```

```

public async Task UploadBlob(string containerN, string blobN, IFormFile file)
{

    //try to get the container with the specified name
    CloudBlobContainer container = GetCloudBlobContainer(containerN);
    //if there is no container, create a new one
    if (!await container.ExistsAsync())
    {
        await CreateBlobContainer(containerN);
    }
    //try again to read the container
    container = GetCloudBlobContainer(containerN);

    //we just create the reference to the object
    CloudBlockBlob blob = container.GetBlockBlobReference(blobN);

    blob.UploadFromStreamAsync(file.OpenReadStream()).Wait();

}

//This method lists all the blobs in a specific container
public ActionResult ListBlobs(string containerName)
{

    CloudBlobContainer container = GetCloudBlobContainer(containerName)

    List<string> blobs = new List<string>();
    BlobResultSegment resultSegment = container
        .ListBlobsSegmentedAsync(null).Result;

    foreach (IListBlobItem item in resultSegment.Results)

```

```

    {
        if (item.GetType() == typeof(CloudBlockBlob))
        {
            CloudBlockBlob blob = (CloudBlockBlob)item;
            blobs.Add(blob.Name);
        }
        else if (item.GetType() == typeof(CloudPageBlob))
        {
            CloudPageBlob blob = (CloudPageBlob)item;
            blobs.Add(blob.Name);
        }
        else if (item.GetType() == typeof(CloudBlobDirectory))
        {
            CloudBlobDirectory dir = (CloudBlobDirectory)item;
            blobs.Add(dir.Uri.ToString());
        }
    }
    return View(blobs);
}

//This method gets the full path of a specific container
public string GetBlobFullPath(string containerName, string blobName)
{
    CloudBlobContainer container = GetCloudBlobContainer(containerName)
    CloudBlockBlob blob = container.GetBlockBlobReference(blobName);
    return blob.StorageUri.PrimaryUri.ToString();
}

//this method allows the calling method to download the whole container
public string DownloadBlob(string containerN, string blobN, string Path)

```

```
{
```

```
CloudBlobContainer container = GetCloudBlobContainer(containerN);
```

```
CloudBlockBlob blob = container.GetBlockBlobReference(blobN);
```

```
using (var fileStream = System.IO.File.OpenWrite(Path))
```

```
{
```

```
    blob.DownloadToStreamAsync(fileStream).Wait();
```

```
}
```

```
return "success!";
```

```
}
```

```
//this methos deletes a blob from a given container
```

```
public string DeleteBlob(string containerName, string blobName)
```

```
{
```

```
CloudBlobContainer container = GetCloudBlobContainer(containerName)
```

```
CloudBlockBlob blob = container.GetBlockBlobReference(blobName);
```

```
blob.DeleteAsync().Wait();
```

```
return "success!";
```

```
}
```

```
public string DeleteContainer(string containerName)
```

```
{
```

```
CloudBlobContainer container = GetCloudBlobContainer(containerName)
```

```
container.DeleteIfExistsAsync();
```

```
return "success!";
```

```
}
```

```
}
```



```
}
```

## A.5 Media Utilities Controller

```
using MediaInfoLib;
using Microsoft.AspNetCore.Hosting;
using System;
using System.IO;
using System.Linq;

namespace SensorFusion.Controllers
{

    // this class is responsible for getting the metadata from
    //the input files
    public class MediaUtilities
    {
        private IHostingEnvironment _env;
        string path;
        MediaInfo mi = new MediaInfo();

        public MediaUtilities(IHostingEnvironment env, string fileName)
        {
            _env = env;
            path = _env.WebRootPath + "\\TempFiles\\" + fileName;
        }

        //this method returns the duration of a video file
        public TimeSpan GetVideoDuration()
        {
            mi.Open(path);
            var videoInfo = new VideoInfo(mi);
            var result= videoInfo.Duration;
        }
    }
}
```

```

        mi.Close();
        return result;
    }
    //this method returns the size of a video file in bytes
    public long GetVideoSize()
    {
        mi.Open(path);
        var videoInfo = new VideoInfo(mi);
        long size = videoInfo.FileSize;
        mi.Close();
        return size;
    }
    //this method returns the video tagged date
    //which is the date that the video started to record
    public DateTime GetVideoTaggedDate()
    {
        mi.Open(path);
        var videoInfo = new VideoInfo(mi);
        string[] taggedDate = videoInfo.TaggedDate.Split(' ');
        string[] date = taggedDate[1].Split('-');
        string[] time = taggedDate[2].Split(':');
        var year=Int32.Parse(date[0]);
        var month=Int32.Parse(date[1]);
        var day=Int32.Parse(date[2]);
        var hour=Int32.Parse(time[0]);
        var minute=Int32.Parse(time[1]);
        var second=Int32.Parse(time[2]);

        DateTime fullDateTime;
        fullDateTime=new DateTime(year,month,day,hour,minute,second);
        mi.Close();
        return fullDateTime;
    }

```

```

}

//this method returns the video encoded date
public DateTime GetVideoEncodedDate()
{
    mi.Open(path);
    var videoInfo = new VideoInfo(mi);
    string[] encodedDate = videoInfo.EncodedDate.Split(' ');
    string[] date = encodedDate[1].Split('-');
    string[] time = encodedDate[2].Split(':');
    var year=Int32.Parse(date[0]);
    var month=Int32.Parse(date[1]);
    var day=Int32.Parse(date[2]);
    var hour=Int32.Parse(time[0]);
    var minute=Int32.Parse(time[1]);
    var second=Int32.Parse(time[2]);
    DateTime fullDateTime;
    fullDateTime=new DateTime(year,month,day,hour,minute,second);
    mi.Close();
    return fullDateTime;
}

//this method prints all the metadata that can be extracted
// from the video files
public void PrintVideoAvailableProperties()
{
    mi.Open(path);
    var videoInfo = new VideoInfo(mi);
    mi.Option("Language", "raw");
    Console.WriteLine(mi.Inform());
}

//this method gets and returns the last modified date

```

```

//from an audio file
public DateTime GetAudioLastModifiedTime()
{
    FileInfo file = new FileInfo(path);
    DateTime timeStamp = file.LastWriteTime;
    return timeStamp;
}
//this method returns the creation time
//from an audio file
public DateTime GetAudioCreationTime()
{
    FileInfo file = new FileInfo(path);
    DateTime timeStamp = file.CreationTime;
    return timeStamp;
}

public DateTime GetAudioLastAccessTime()
{
    FileInfo file = new FileInfo(path);
    DateTime timeStamp = file.LastAccessTime;
    return timeStamp;
}

public DateTime GetAudioLastWriteTime()
{
    DateTime timeStamp = File.GetLastWriteTimeUtc(path);
    return timeStamp;
}

//this method gets and returns the duration of a
//audio file

```

```

public TimeSpan GetAudioDuration()
{
    mi.Open(path);
    var audioInfo = new AudioInfo(mi);
    var result = audioInfo.Duration;
    mi.Close();
    return result;
}
//this method gets and returns the size
// of an audio file
public string GetAudioSize()
{
    mi.Open(path);
    var audioInfo = new AudioInfo(mi);
    var result = audioInfo.FileSize;
    mi.Close();
    return result;
}
//this method gets and returns the size
// of an audio file
public string GetAudioStreamSize()
{
    mi.Open(path);
    var audioInfo = new AudioInfo(mi);
    var result = audioInfo.StreamSize;
    mi.Close();
    return result;
}

//this method examines all the dates that an audio files has
// and returns the earliest one

```

```

public DateTime GetAudioEarliestDate(string fileName)
{
    DateTime earliest = new DateTime(9000,1,1);
    //earliest is later than GetAudioCreationTime
    var audioCreation=earliest.CompareTo(GetAudioCreationTime()) > 0;
    var audioLastAcc=earliest.CompareTo(GetAudioLastAccessTime())>0;
    var audioLastMod=earliest.CompareTo(GetAudioLastModifiedTime())>0;
    var audioLastWrite=earliest.CompareTo(GetAudioLastWriteTime())>0;

    earliest = audioCreation ? GetAudioCreationTime() : earliest;
    earliest = audioLastAcc ? GetAudioLastAccessTime() : earliest;
    earliest = audioLastMod ? GetAudioLastModifiedTime() : earliest;
    earliest = audioLastWrite ? GetAudioLastWriteTime() : earliest;

    mi.Open(path);
    var audioInfo = new AudioInfo(mi);
    if (audioInfo.TaggedDate!=null && audioInfo.TaggedDate!="")
    {
        string[] taggedDate = audioInfo.TaggedDate.Split(' ');
        string[] date = taggedDate[1].Split('-');
        string[] time = taggedDate[2].Split(':');
        var year=Int32.Parse(date[0]);
        var month=Int32.Parse(date[1]);
        var day=Int32.Parse(date[2]);
        var hour=Int32.Parse(time[0]);
        var minute=Int32.Parse(time[1]);
        var second=Int32.Parse(time[2]);

        DateTime TaggedDate=new DateTime(year,month,day,hour,minute,second)
        earliest=(earliest.CompareTo(TaggedDate)>0)?TaggedDate:earliest;
    }
}

```

```

        if(audioInfo.EncodedDate != null && audioInfo.EncodedDate!= "")
        {
            string[] encodedDate = audioInfo.EncodedDate.Split(' ');
            string[] date = encodedDate[1].Split('-');
            string[] time = encodedDate[2].Split(':');
            var year=Int32.Parse(date[0]);
            var month=Int32.Parse(date[1]);
            var day=Int32.Parse(date[2]);
            var hour=Int32.Parse(time[0]);
            var minute=Int32.Parse(time[1]);
            var second=Int32.Parse(time[2]);
            DateTime EncodedDate;
            EncodedDate = new DateTime(year, month, day, hour, minute, second);

            earliest=(earliest.CompareTo(EncodedDate)>0)?EncodedDate:earliest;
        }
        mi.Close();
        return earliest;
    }
}

```

```

//this methods prints all the available metatada that
//can be extracted from the audio files
public void PrintAudioAvailableProperties()
{
    mi.Open(path);
    var videoInfo = new AudioInfo(mi);
    mi.Option("Language", "raw");
    Console.WriteLine(mi.Inform());
}

```

```

//this method examines all the available dates that exist
// in a file and returns the earliest one
public DateTime GetFileEarliestDate(string fileName)
{
    DateTime earliest = new DateTime(9000, 1, 1);
    var fileName=(GetDateFromFileName(fileName)
    .CompareTo(new DateTime()) != 0);
    earliest = fileName ? GetDateFromFileName(fileName) : earliest;

    //earliest is later than GetAudioCreationTime
    var audioCreation=earliest.CompareTo(GetAudioCreationTime()) > 0;
    var audioLastAccessss=earliest.CompareTo(GetAudioLastAccessTime())>0
    var audioLastMod=earliest.CompareTo(GetAudioLastModifiedTime()) > 0
    var audioLastWrite=earliest.CompareTo(GetAudioLastWriteTime()) > 0;

    earliest = audioCreation ? GetAudioCreationTime() : earliest;
    earliest = audioLastAccessss ? GetAudioLastAccessTime() : earliest;
    earliest = audioLastMod ? GetAudioLastModifiedTime() : earliest;
    earliest = audioLastWrite ? GetAudioLastWriteTime() : earliest;
    return earliest;
}

//this method get the date from a file
public static DateTime GetDateFromFileName(string fileName)
{
    try
    {
        string[] datesArray = fileName.Split('_');
        datesArray[0]=datesArray[0].Substring(datesArray[0].Length-4, 4);
        datesArray[5] = datesArray[5].Substring(0, 2);
    }
}

```



```

var year=Int32.Parse(datesArray[0]);
var month=Int32.Parse(datesArray[1]);
var day=Int32.Parse(datesArray[2]);
var hour=Int32.Parse(datesArray[3]);
var minute=Int32.Parse(datesArray[4]);
var second=Int32.Parse(datesArray[5]);

        DateTime result;
        result=DateTime(year,month,day,hour,minute,second);
        return result;
    }
    catch (Exception)
    {

        return new DateTime();
    }

}

public static void CleanTempFolder(IHostingEnvironment hostingEnvironment)
{

    System.IO.DirectoryInfo di;
    di=new DirectoryInfo(hostingEnvironment.WebRootPath+"\\TempFiles");

    foreach (FileInfo file in di.GetFiles())
    {
        file.Delete();
    }

}

```

```
}
```

```
//this class instantiates the MediaInfo object and holds all the  
//metadata for audio and video files
```

```
public class VideoInfo
```

```
{
```

```
public string Codec { get; private set; }
```

```
public int Width { get; private set; }
```

```
public int Height { get; private set; }
```

```
public double FrameRate { get; private set; }
```

```
public string FrameRateMode { get; private set; }
```

```
public string ScanType { get; private set; }
```

```
public TimeSpan Duration { get; private set; }
```

```
public int Bitrate { get; private set; }
```

```
public string AspectRatioMode { get; private set; }
```

```
public double AspectRatio { get; private set; }
```

```
public string TaggedDate { get; private set; }
```

```
public string EncodedDate { get; private set; }
```

```
public long FileSize { get; private set; }
```

```
public VideoInfo(MediaInfo mi)
```

```
{
```

```
    Codec = mi.Get(StreamKind.Video, 0, "Format");
```

```
    Width = int.Parse(mi.Get(StreamKind.Video, 0, "Width"));
```

```
    Height = int.Parse(mi.Get(StreamKind.Video, 0, "Height"));
```

```
    var dur=int.Parse(mi.Get(StreamKind.Video, 0, "Duration"));
```

```
    Duration = TimeSpan.FromMilliseconds(dur);
```

```
    Bitrate = int.Parse(mi.Get(StreamKind.Video, 0, "BitRate"));
```

```
    AspectRatioMode =mi.Get(StreamKind.Video, 0, "AspectRatio/String");
```

```

        AspectRatio = double.Parse(mi.Get(StreamKind.Video, 0, "AspectRatio"));
        FrameRate = double.Parse(mi.Get(StreamKind.Video, 0, "FrameRate"));
        FrameRateMode = mi.Get(StreamKind.Video, 0, "FrameRate_Mode");
        ScanType = mi.Get(StreamKind.Video, 0, "ScanType");
        TaggedDate = mi.Get(StreamKind.Video, 0, "Tagged_Date");
        EncodedDate = mi.Get(StreamKind.General, 0, "Encoded_Date");
        FileSize = Int64.Parse(mi.Get(StreamKind.General, 0, "FileSize"));
    }
}

```

```

public class AudioInfo
{
    public string Codec { get; private set; }
    public string CompressionMode { get; private set; }
    public string ChannelPositions { get; private set; }
    public TimeSpan Duration { get; private set; }
    public int Bitrate { get; private set; }
    public string BitrateMode { get; private set; }
    public int SamplingRate { get; private set; }
    public string TaggedDate { get; private set; }
    public string EncodedDate { get; private set; }
    public string FileSize { get; set; }
    public string StreamSize { get; set; }

    public AudioInfo(MediaInfo mi)
    {
        Codec = mi.Get(StreamKind.Audio, 0, "Format");
        var dur=int.Parse(mi.Get(StreamKind.Audio, 0, "Duration"));
        Duration = TimeSpan.FromMilliseconds(dur);
        Bitrate = int.Parse(mi.Get(StreamKind.Audio, 0, "BitRate"));
        BitrateMode = mi.Get(StreamKind.Audio, 0, "BitRate_Mode");
        CompressionMode = mi.Get(StreamKind.Audio, 0, "Compression_Mode");
    }
}

```

```
ChannelPositions = mi.Get(StreamKind.Audio, 0, "ChannelPositions");
SamplingRate=int.Parse(mi.Get(StreamKind.Audio,0,"SamplingRate"));
TaggedDate = mi.Get(StreamKind.General, 0, "Tagged_Date");
EncodedDate = mi.Get(StreamKind.General, 0, "Encoded_Date");
FileSize = mi.Get(StreamKind.General, 0, "FileSize");
StreamSize = mi.Get(StreamKind.Audio, 0, "StreamSize");
}
}
}
```

# **Appendix B**

## **System Manual**

# Bibliography

- [1] Anne Author. Example Journal Paper Title. *Journal of Classic Examples*, 1(1):e1001745+, January 1970.
- [2] Karl Wiegers and Joy Beatty. *Software Requirements, 3rd Edition*. Microsoft Press, 2013.
- [3] D. Clegg and R. Barker. *Case method fast-track: a RAD approach*. AddisonWesley, Longman Publishing Co., Inc., 1994.
- [4] Domain modeling. Available at:  
<https://www.scaledagileframework.com/domain-modeling>, September 2017. Accessed: [12 June 2018].
- [5] Ivar Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach: A Use CASE Approach (ACM Press)*. Addison-Wesley Professional, 1992.
- [6] Karl E. Wiegers. Listening to the customer's voice. Available at:  
<http://www.processimpact.com/articles/usecase.pdf>, 1997. Accessed: [11 July 2018].
- [7] Stephen Halter Tom Dykstra, Steve Smith and Chris Ross. Web server implementations in asp.net core. Available at: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/?view=aspnetcore-2.1&tabs=aspnetcore2x>, March 2018. Accessed: [12 July 2018].
- [8] Edsger W.Dijkstra. On the role of scientific thought. Available at: <https://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html>, August 1974. Accessed: [15 July 2018].
- [9] Steve Smith. What is the mvc pattern? Available at: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.1#routing>, August 2018. Accessed: [15 August 2018].

- [10] K. Scott Allen J. Galloway, B. Wilson and D. Matson. *Professional ASP.NET MVC 5*. Wrox, 2014.
- [11] Peter P. Chen. *Entity Relationship Approach to Information Modeling and Analysis*. ER institute (1981), 1981.
- [12] James R. Rumbaugh Michael R. Blaha. *Object-Oriented Modeling and Design with UML: International Edition, 2/E*. Pearson, 2005.
- [13] Hugh Darwen C.J. Date. *A Guide to SQL Standard (4th Edition)*. Addison-Wesley Professional, November 1996.
- [14] Carolyn Begg Thomas Conolly. *Database Systems, Sixth Edition*. Pearson, 2005.
- [15] Rachel Appel. Model binding in asp.net core. Available at:  
<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-2.1>, August 2018. Accessed: [20 August 2018].
- [16] Ryan Nowak and Rick Anderson. Routing to controller actions in asp.net core. Available at:  
<https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-2.1>, March 2017. Accessed: [15 July 2018].
- [17] Roger S Pressman. *Software engineering: a practitioner's approach*. McGraw-Hill Higher Education, April 2009.
- [18] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, November 2002.
- [19] David Astels. *Test-Driven Development: A Practical Guide: A Practical Guide*. Prentice Hall, July 2003.
- [20] Robert Martin. *The Clean Coder: A Code of Conduct for Professional Programmers (Robert C. Martin)*. May 2011.
- [21] System testing fundamentals. Available at:  
<http://softwaretestingfundamentals.com/system-testing/>, January 2011. Accessed: [20 August 2018].

This document was set in the Times Roman typeface using L<sup>A</sup>T<sub>E</sub>X and BibT<sub>E</sub>X, composed with a text editor.