Sensor fusion for training doctors

Author:
Dimitrios Selalmazidis

Supervisor:
Ghita Kouadri-Mostefaoui

MSc Computer Science

September, 2018

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

List of Figures

List of Tables

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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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, consectetuer 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
	Uploading a new Operation	
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 (Neurosurgery, 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
	Searching for an Operation	
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
	Details of an Operation	
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.

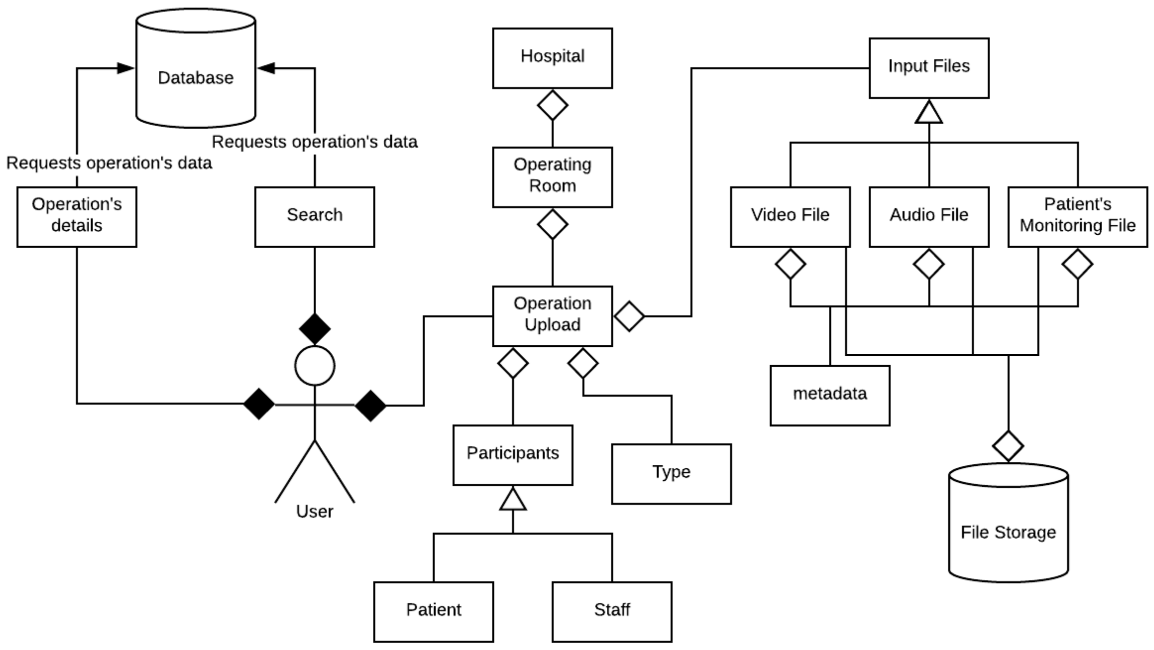


Figure 3.1: Conceptual model of the system's problem domain

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	**Upload a new operation**
UC02	**Search for an operation**
UC03	**View operation detals**

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	1. Home page is displayed 2. The User selects to register a new operation 3. The User enters the hospital name 4. The User enters the operating room number 5. The User enters all the staff that participated in the operation 6. The User enters the patient that underwent the surgery 7. The User enters the type of the operation 8. The User selects all the relevant video files 9. The User selects all the relevant audio files 10. The User selects the patient's monitoring file
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	1. The User does not enter any of the 1-7 fields and at least one input file 2. An error message is displayed
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	1. Home page is displayed 2. The 20 most recent uploaded operations are displayed 3. The User uses the Filters to input hospital name,operating room number, from/to date and participated staff 4. The filtered results are displayed, ordered by date (from most recent to oldest)
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	1. At least one operation must have been registered 2. The User must have searched for an Operation
Main Flow	1. Home page is displayed 2. The User searches for an Operation 3. The User views the filtered results 4. The User selects the desired Operation 5.The User views the Operation details
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

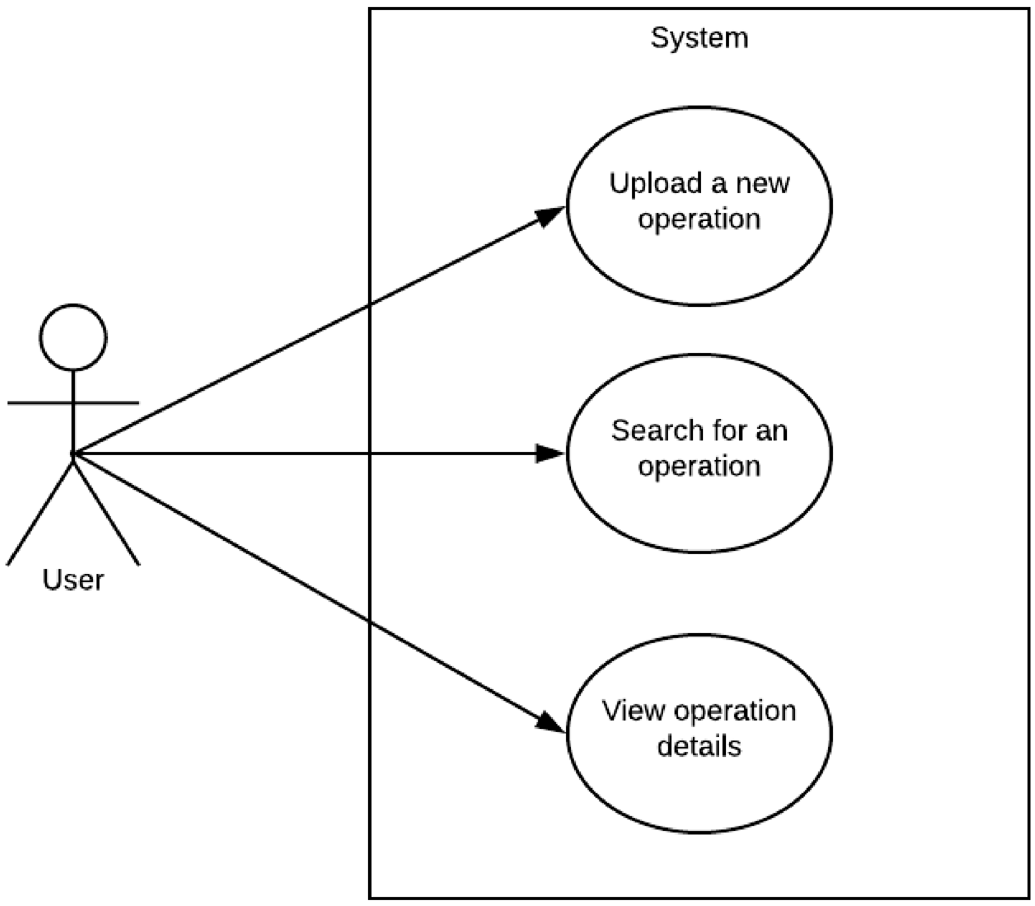


Figure 3.2: Use case diagram

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 fro 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.

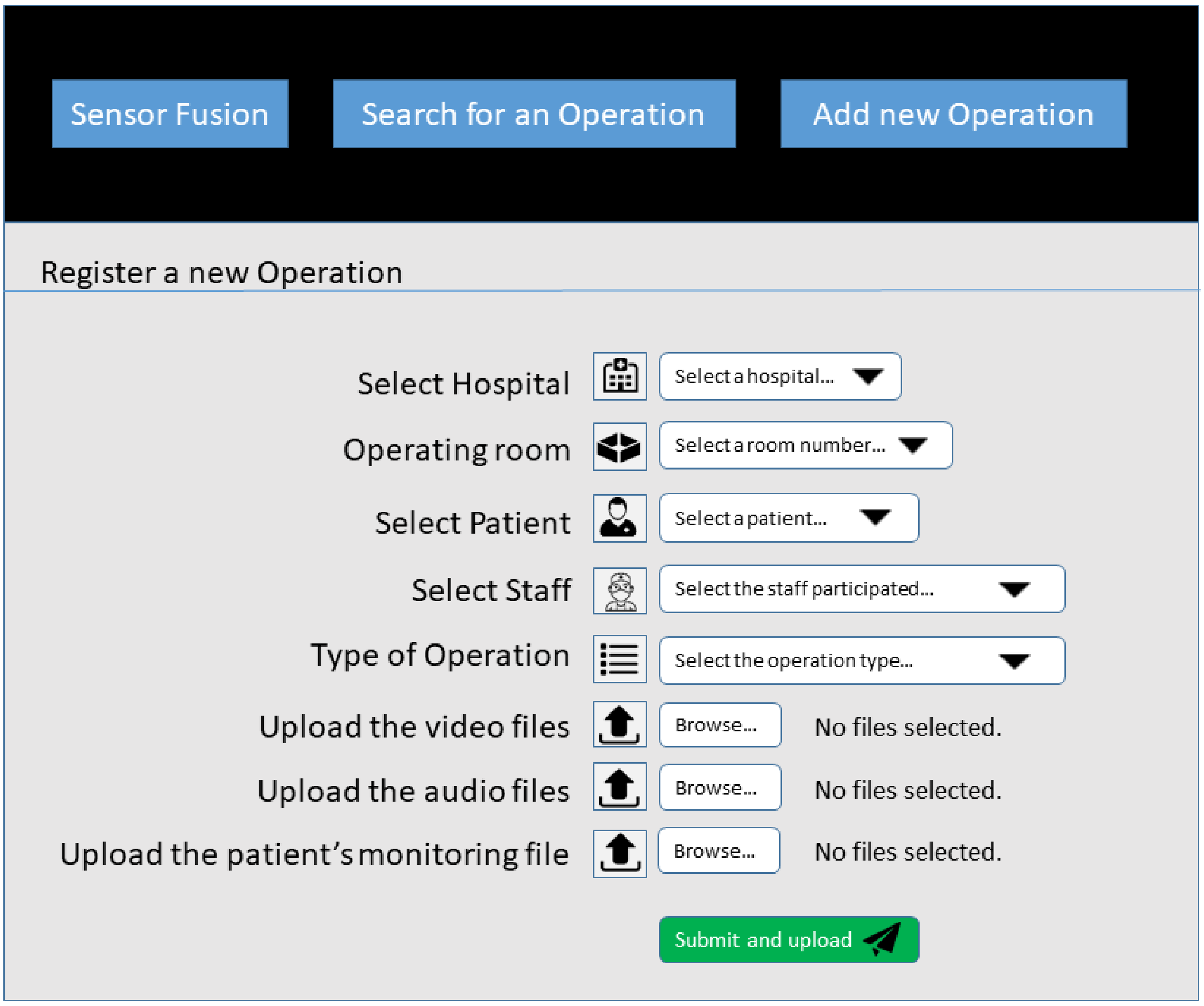
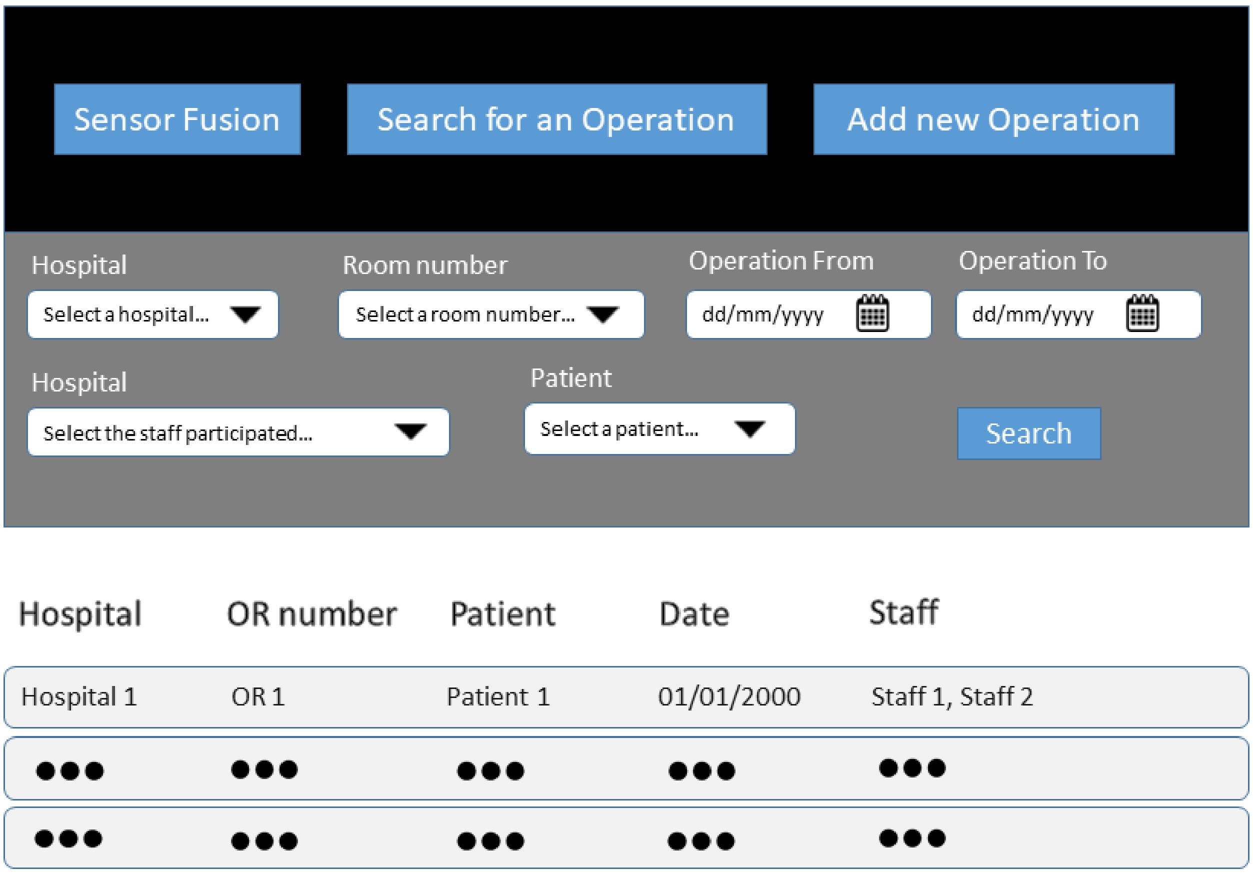
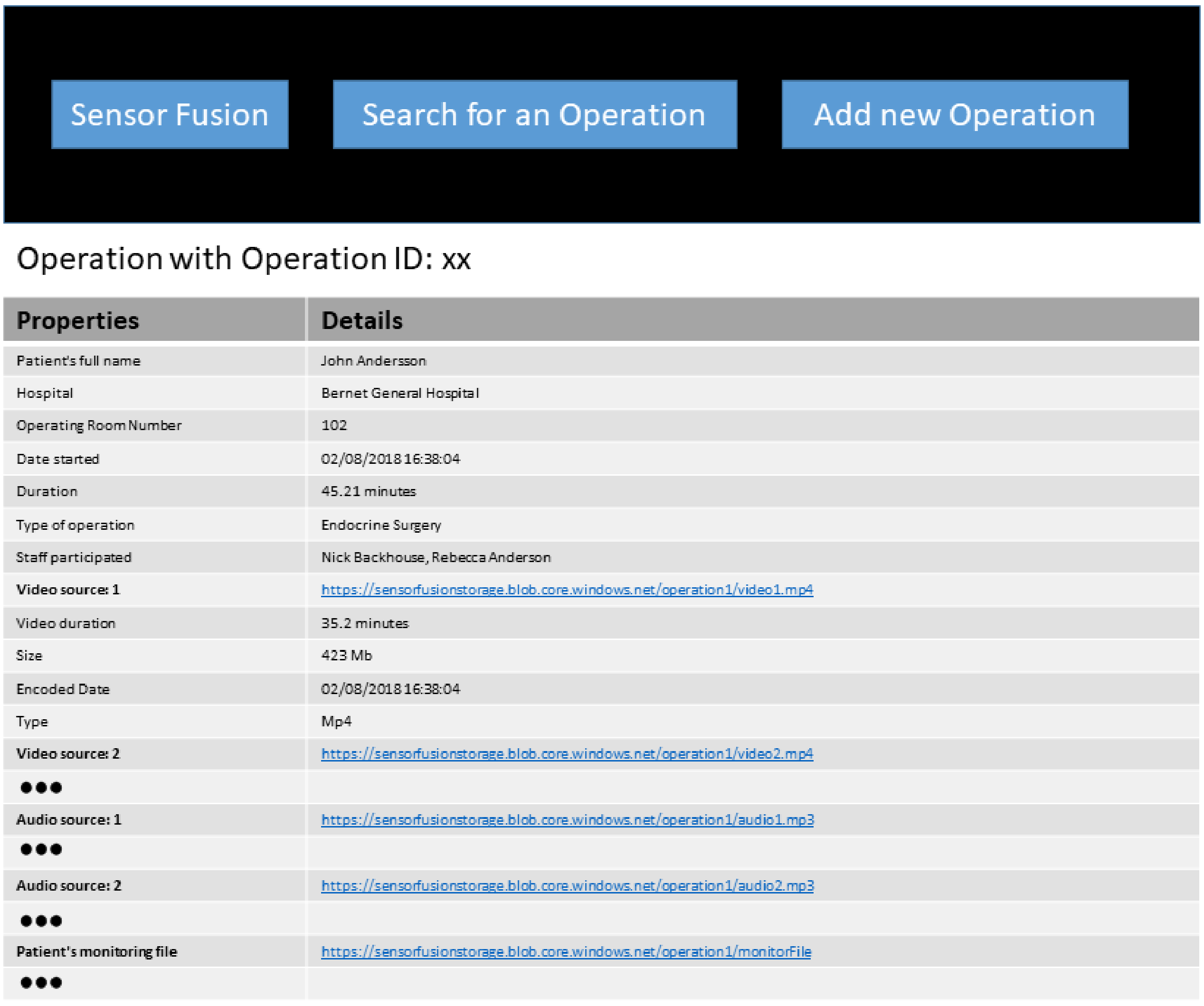


Figure 3.3: "Add new Operation" View



Hospital	OR number	Patient	Date	Staff
Hospital 1	OR 1	Patient 1	01/01/2000	Staff 1, Staff 2
●●●	●●●	●●●	●●●	●●●
●●●	●●●	●●●	●●●	●●●

Figure 3.4: “Search for an Operation” View



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	https://sensorfusionstorage.blob.core.windows.net/operation1/video1.mp4
Video duration	35.2 minutes
Size	423 Mb
Encoded Date	02/08/2018 16:38:04
Type	Mp4
Video source: 2	https://sensorfusionstorage.blob.core.windows.net/operation1/video2.mp4
●●●	
Audio source: 1	https://sensorfusionstorage.blob.core.windows.net/operation1/audio1.mp3
●●●	
Audio source: 2	https://sensorfusionstorage.blob.core.windows.net/operation1/audio2.mp3
●●●	
Patient's monitoring file	https://sensorfusionstorage.blob.core.windows.net/operation1/monitorFile
●●●	

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.

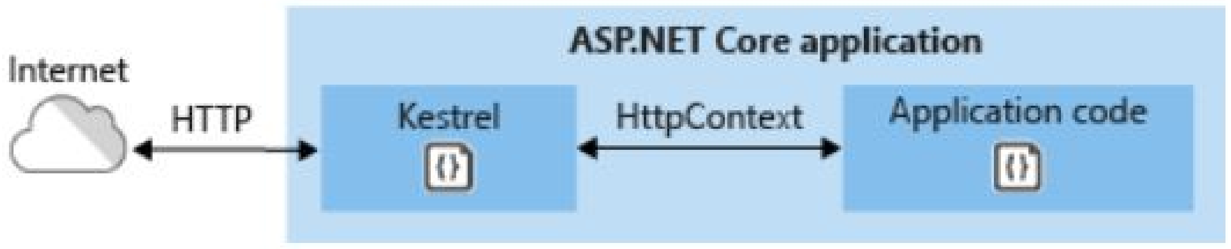


Figure 4.1: ASP.NET Structure

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.

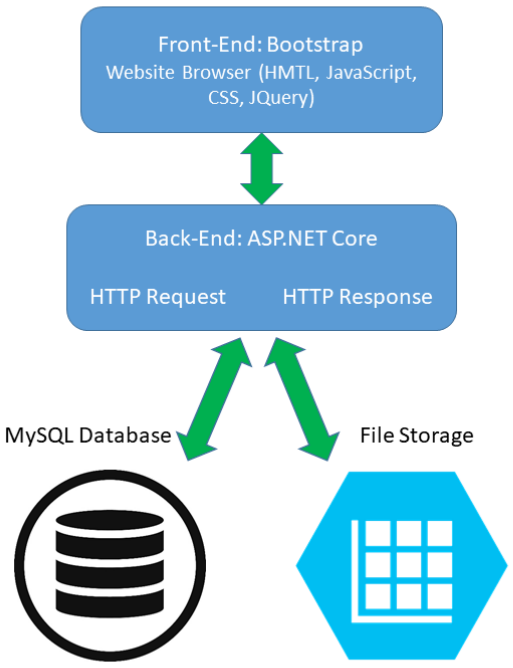


Figure 4.2: Application General Structure

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 the 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 a 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].

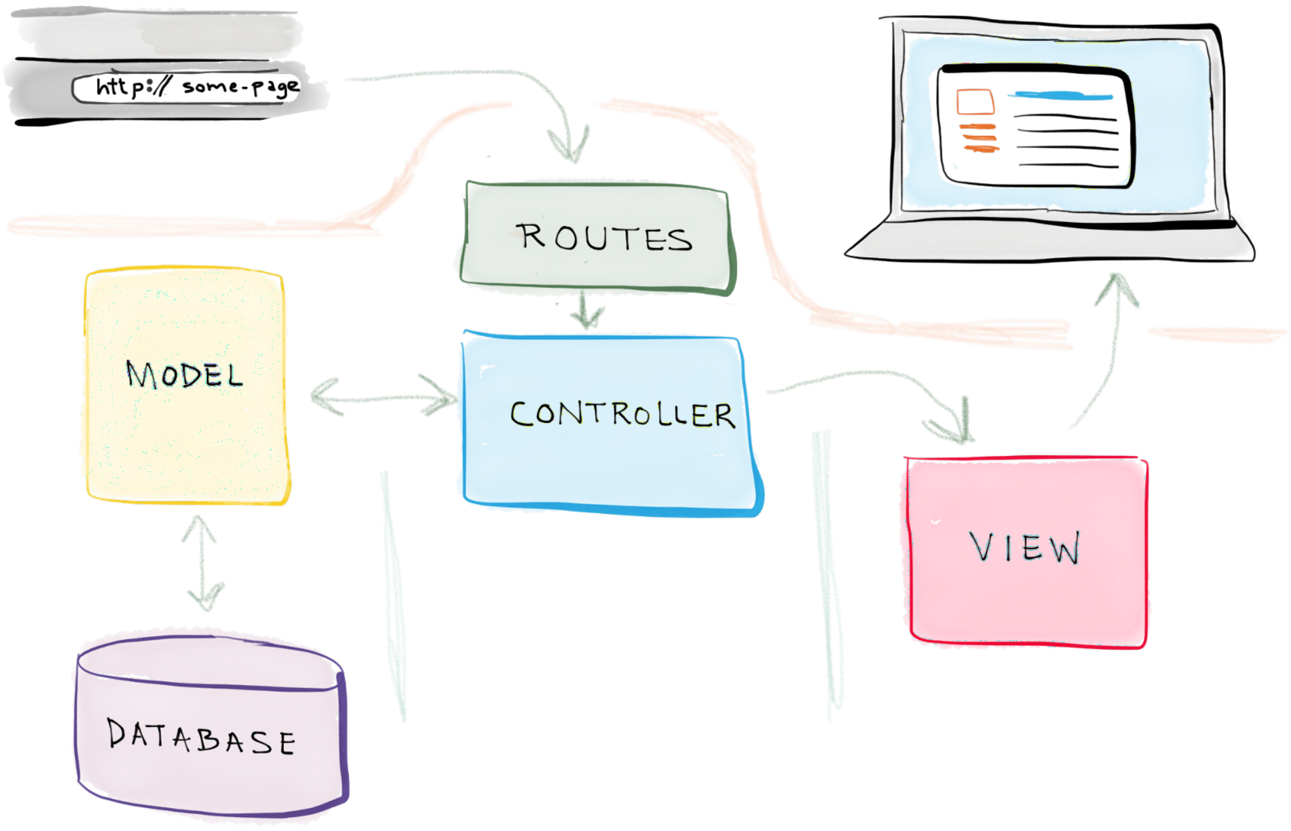


Figure 4.3: The Model-View-Controller Pattern of ASP.NET Core

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. On 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].

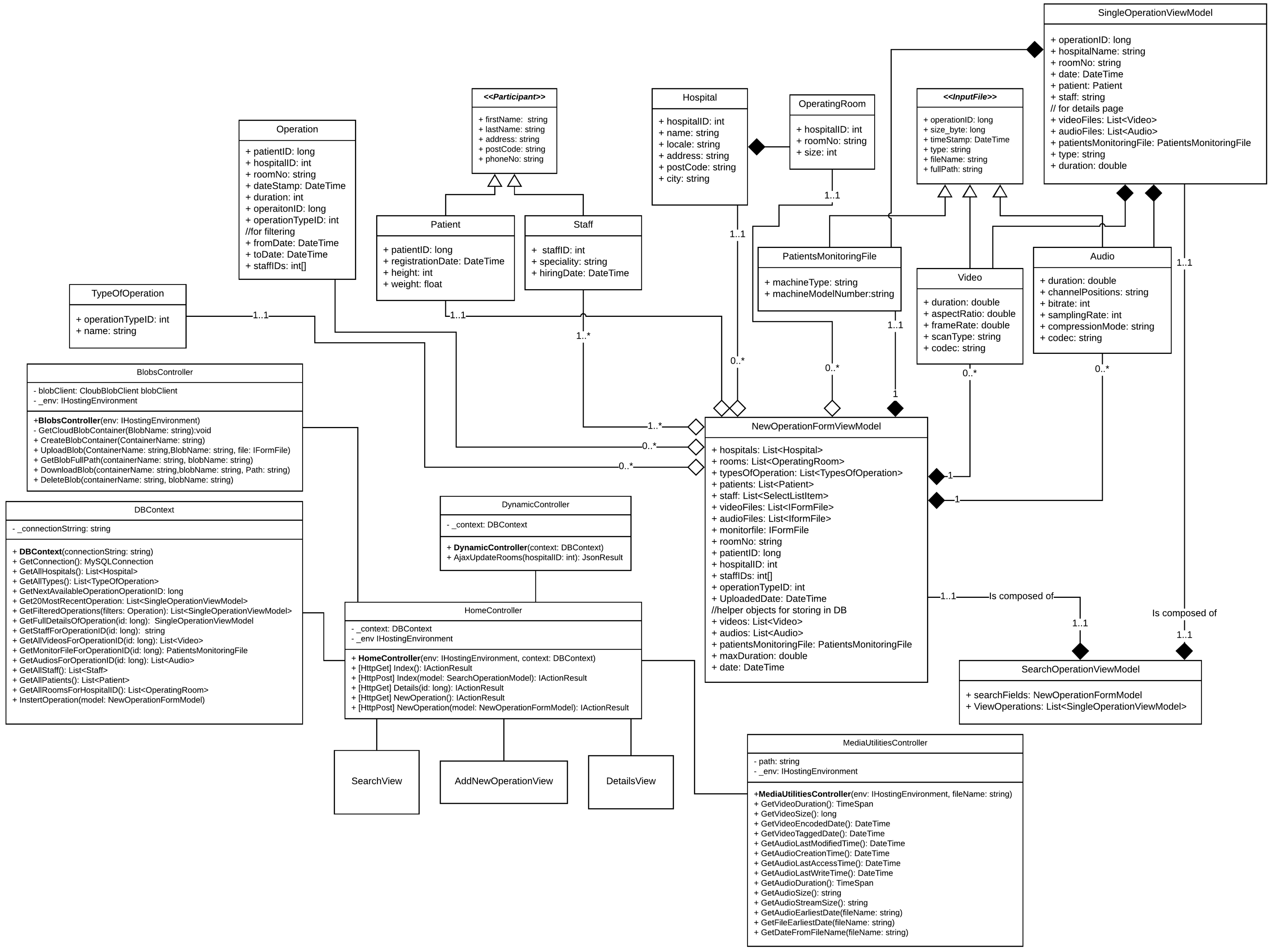

SingleOperationViewModel
+ operationID: long
+ hospitalName: string
+ roomNo: string
+ date: DateTime
+ patient: Patient
+ staff: string
// for details page
+ videoFiles: List<Video>
+ audioFiles: List<Audio>
+ patientsMonitoringFile: PatientsMonitoringFile
+ type: string
+ duration: double
<<Participant>>
+ firstName: string
+ lastName: string
+ address: string
+ postCode: string
+ phoneNo: string
Hospital
+ hospitalID: int
+ name: string
+ locale: string
+ address: string
+ postCode: string
+ city: string
OperatingRoom
+ hospitalID: int
+ roomNo: string
+ size: int
<<InputFile>>
+ operationID: long
+ size_byte: long
+ timeStamp: DateTime
+ type: string
+ fileName: string
+ fullPath: string
Operation
+ patientID: long
+ hospitalID: int
+ roomNo: string
+ dateStamp: DateTime
+ duration: int
+ operaitonID: long
+ operationTypeID: int
//for filtering
+ fromDate: DateTime
+ toDate: DateTime
+ staffIDs: int[]
Patient
+ patientID: long
+ registrationDate: DateTime
+ height: int
+ weight: float
Staff
+ staffID: int
+ speciality: string
+ hiringDate: DateTime
PatientsMonitoringFile
+ machineType: string
+ machineModelNumber:string
Video
+ duration: double
+ aspectRatio: double
+ frameRate: double
+ scanType: string
+ codec: string
Audio
+ duration: double
+ channelPositions: string
+ bitrate: int
+ samplingRate: int
+ compressionMode: string
+ codec: string
TypeOfOperation
+ operationTypeID: int
+ name: string
1..1
1..1
1..*
1..1
1..1
1..1
1..1
0..*
0..*
0..*
0..*
1
BlobsController
- blobClient: CloubBlobClient blobClient
- _env: IHostingEnvironment
+**BlobsController**(env: IHostingEnvironment)
- GetCloudBlobContainer(BlobName: string):void
+ CreateBlobContainer(ContainerName: string)
+ UploadBlob(ContainerName: string,BlobName: string, file: IFormFile)
+ GetBlobFullPath(containerName: string, blobName: string)
+ DownloadBlob(containerName: string,blobName: string, Path: string)
+ DeleteBlob(containerName: string, blobName: string)
1..*
0..*
0..*
NewOperationFormViewModel
+ hospitals: List<Hospital>
+ rooms: List<OperatingRoom>
+ typesOfOperation: List<TypesOfOperation>
+ patients: List<Patient>
+ staff: List<SelectListItem>
+ videoFiles: List<IFormFile>
+ audioFiles: List<IformFile>
+ monitorfile: IFormFile
+ roomNo: string
+ patientID: long
+ hospitalID: int
+ staffIDs: int[]
+ operationTypeID: int
+ UploadedDate: DateTime
//helper objects for storing in DB
+ videos: List<Video>
+ audios: List<Audio>
+ patientsMonitoringFile: PatientsMonitoringFile
+ maxDuration: double
+ date: DateTime
1
1
DBContext
- _connectionStrring: string
+ **DBContext**(connectionString: string)
+ GetConnection(): MySQLConnection
+ GetAllHospitals(): List<Hospital>
+ GetAllTypes(): List<TypeOfOperation>
+ GetNextAvailableOperationOperationID: long
+ Get20MostRecentOperation: List<SingleOperationViewModel>
+ GetFilteredOperations(filters: Operation): List<SingleOperationViewModel>
+ GetFullDetailsOfOperation(id: long): SingleOperationViewModel
+ GetStaffForOperationID(id: long): string
+ GetAllVideosForOperationID(id: long): List<Video>
+ GetMonitorFileForOperationID(id: long): PatientsMonitoringFile
+ GetAudiosForOperationID(id: long): List<Audio>
+ GetAllStaff(): List<Staff>
+ GetAllPatients(): List<Patient>
+ GetAllRoomsForHospitalID(): List<OperatingRoom>
+ InstertOperation(model: NewOperationFormModel)
DynamicController
- _context: DBContext
+ **DynamicController**(context: DBContext)
+ AjaxUpdateRooms(hospitalID: int): JsonResult
HomeController
- _context: DBContext
- _env IHostingEnvironment
+ **HomeController**(env: IHostingEnvironment, context: DBContext)
+ [HttpGet] Index(): IActionResult
+ [HttpPost] Index(model: SearchOperationModel): IActionResult
+ [HttpGet] Details(id: long): IActionResult
+ [HttpGet] NewOperation(): IActionResult
+ [HttpPost] NewOperation(model: NewOperationFormModel): IActionResult
1..1
Is composed of
1..1
Is composed of
1..1
SearchOperationViewModel
+ searchFields: NewOperationFormModel
+ ViewOperations: List<SingleOperationViewModel>
SearchView
AddNewOperationView
DetailsView
MediaUtilitiesController
- path: string
- _env: IHostingEnvironment
+**MediaUtilitiesController**(env: IHostingEnvironment, fileName: string)
+ GetVideoDuration(): TimeSpan
+ GetVideoSize(): long
+ GetVideoEncodedDate(): DateTime
+ GetVideoTaggedDate(): DateTime
+ GetAudioLastModifiedTime(): DateTime
+ GetAudioCreationTime(): DateTime
+ GetAudioLastAccessTime(): DateTime
+ GetAudioLastWriteTime(): DateTime
+ GetAudioDuration(): TimeSpan
+ GetAudioSize(): string
+ GetAudioStreamSize(): string
+ GetAudioEarliestDate(fileName: string)
+ GetFileEarliestDate(fileName: string)
+ GetDateFromFileName(fileName: string)

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.

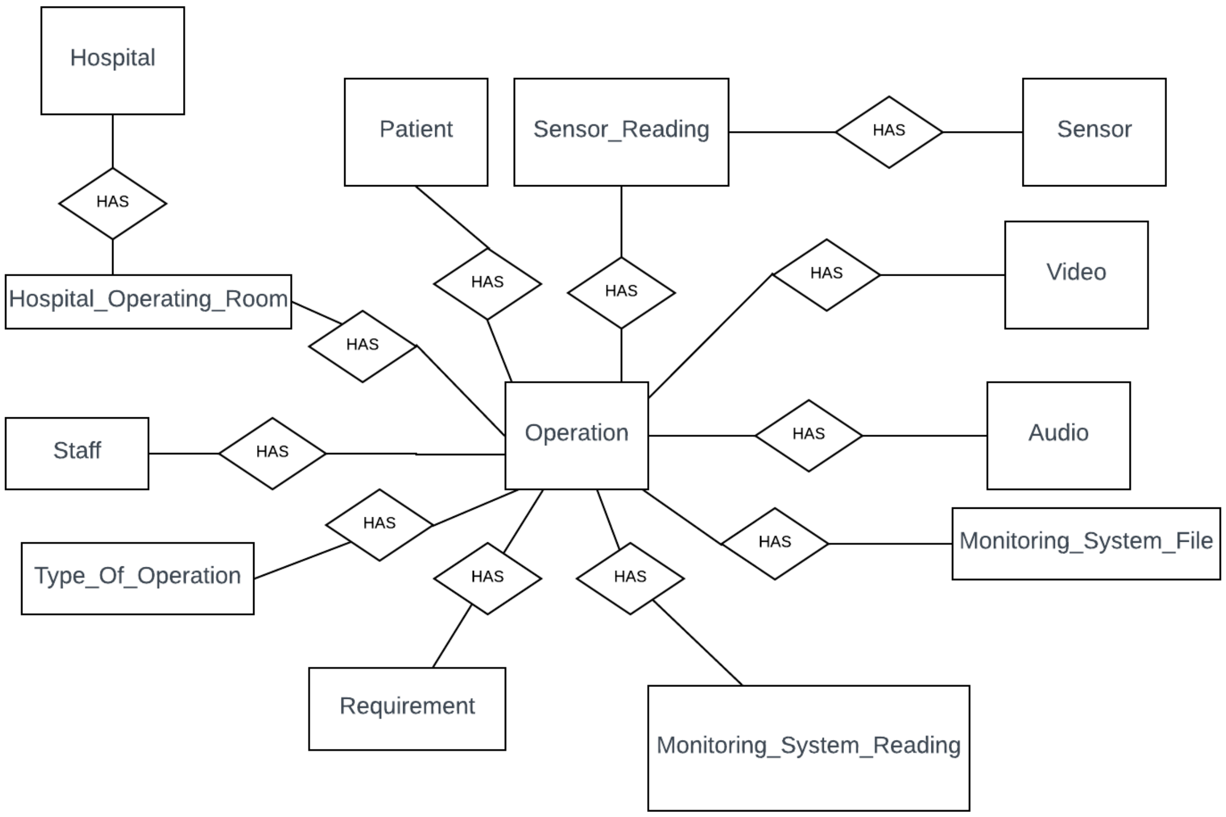


Figure 4.5: Conceptual schema

4.4.4 Physical schema

In this final phase of the database design methodology, the logical database design (entities, attributes, relationships, and contrains) 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 constrains 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

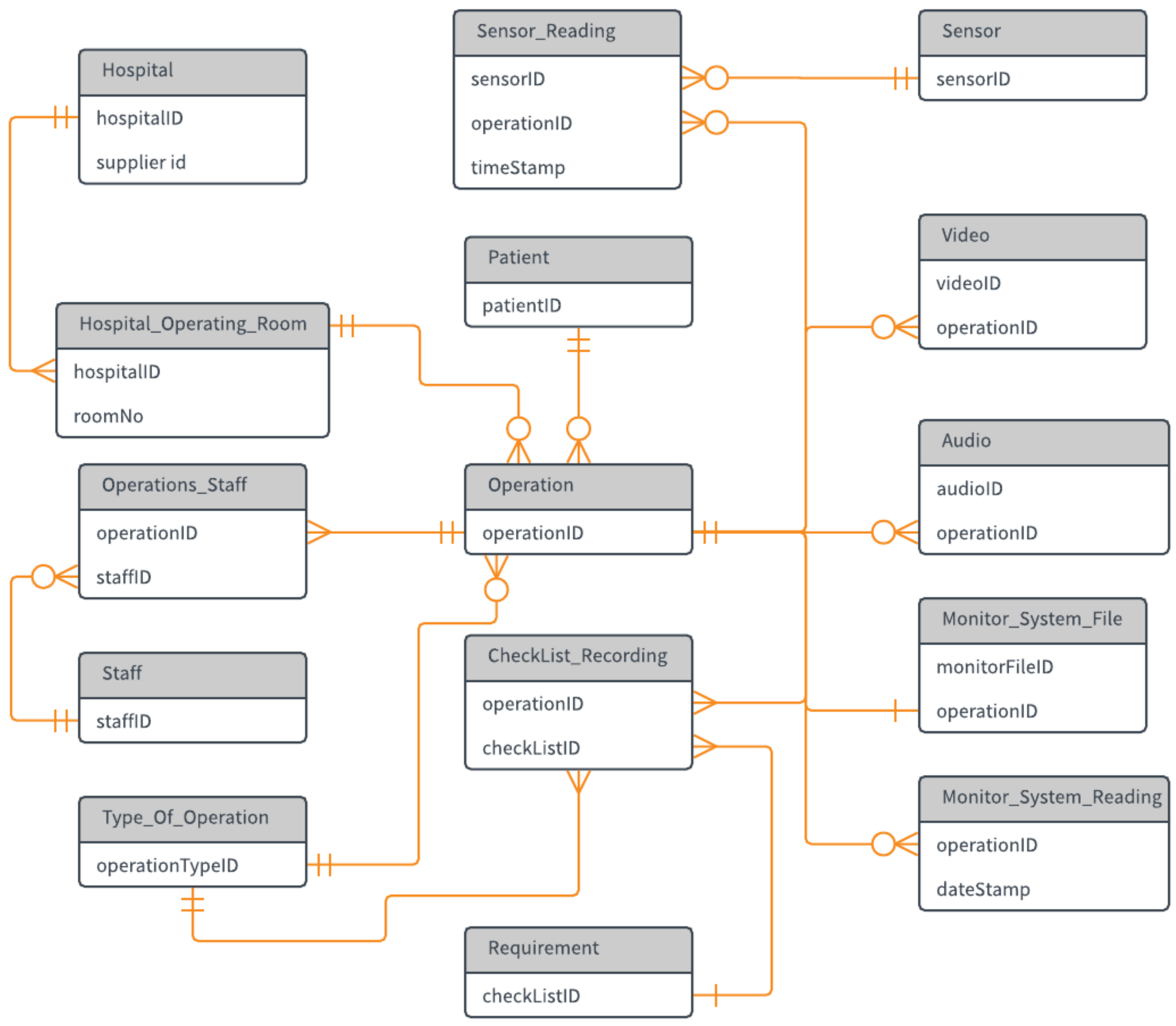


Figure 4.6: Entity-Relationship Diagram

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")

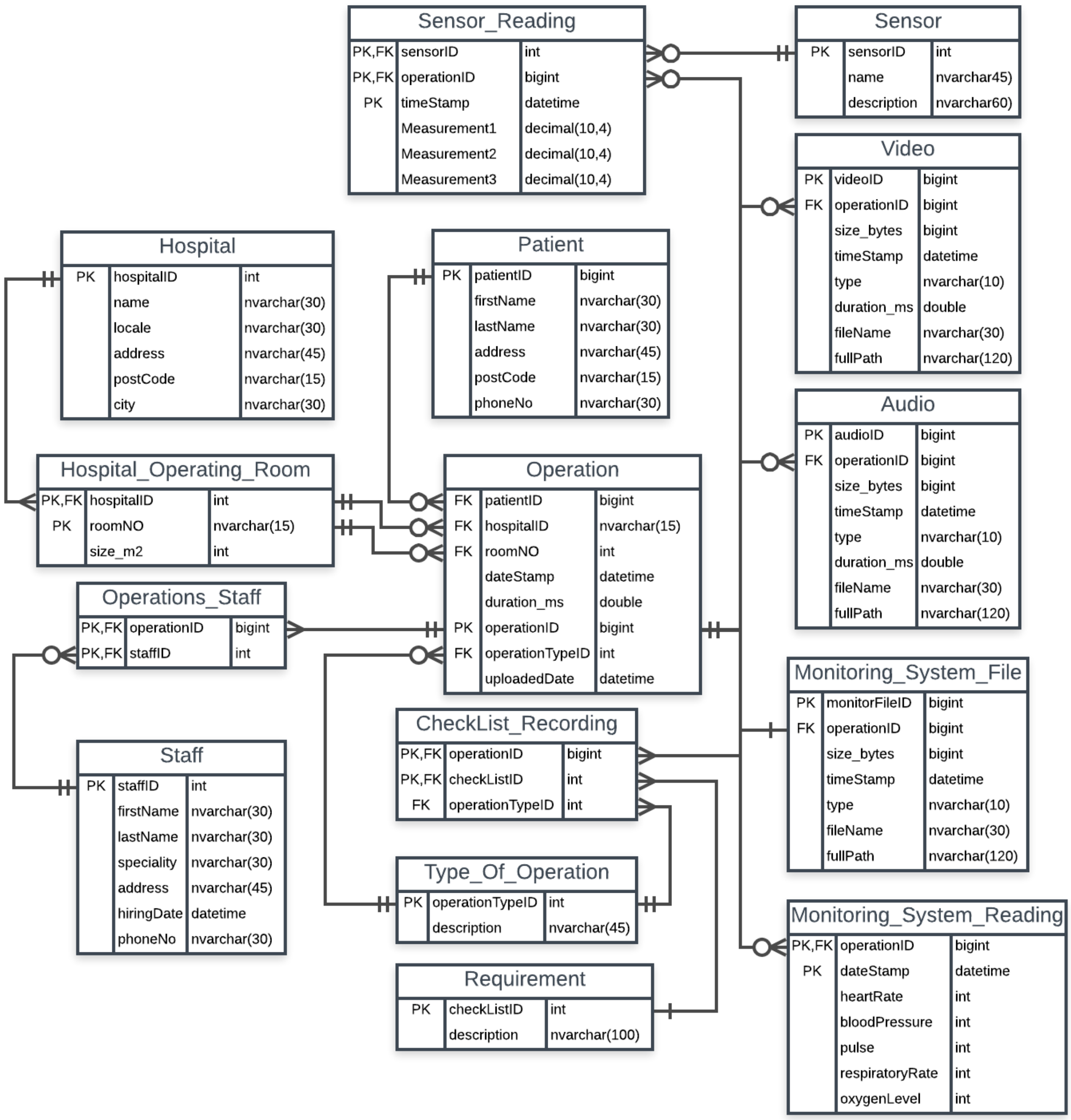
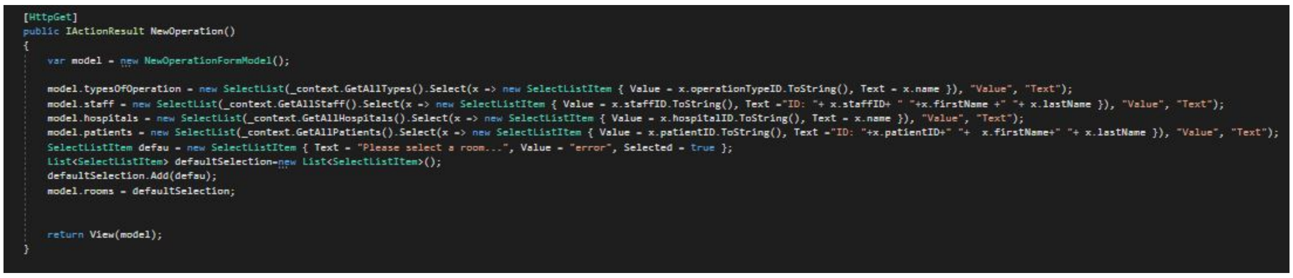


Figure 4.7: Physical database schema

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.

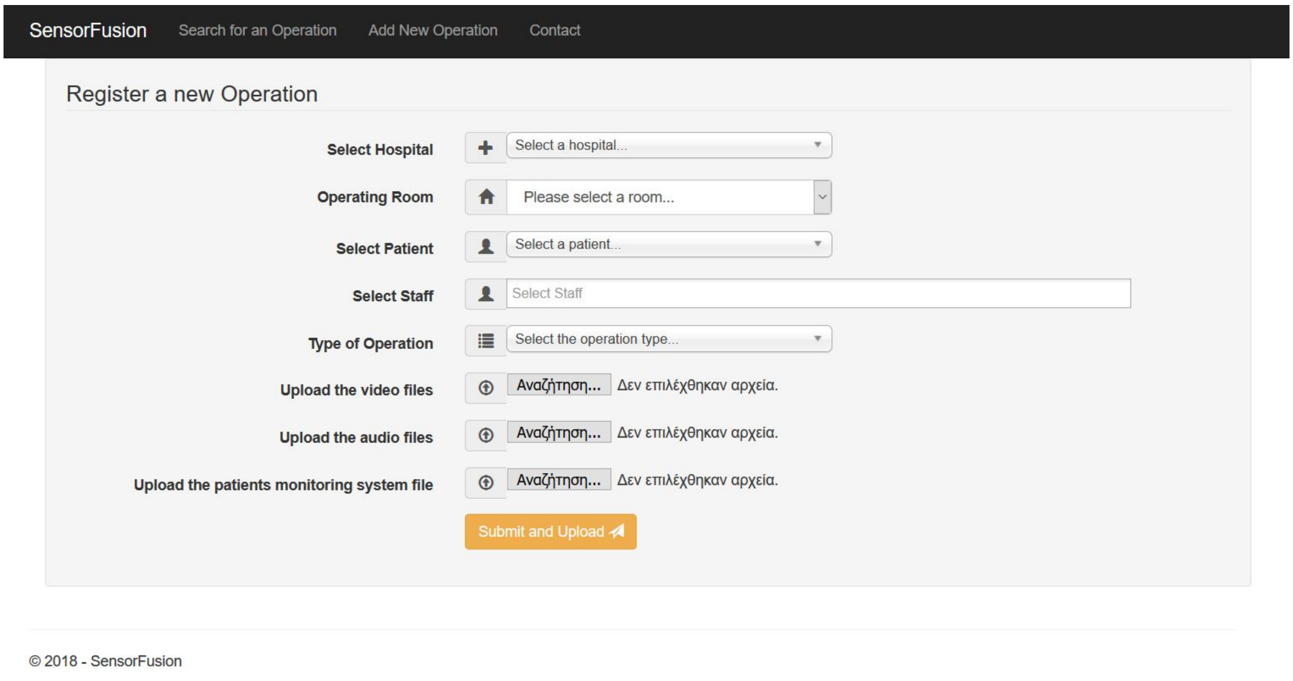


```
[HttpGet]
public IActionResult NewOperation()
{
    var model = new NewOperationFormModel();

    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 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);
}
```

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

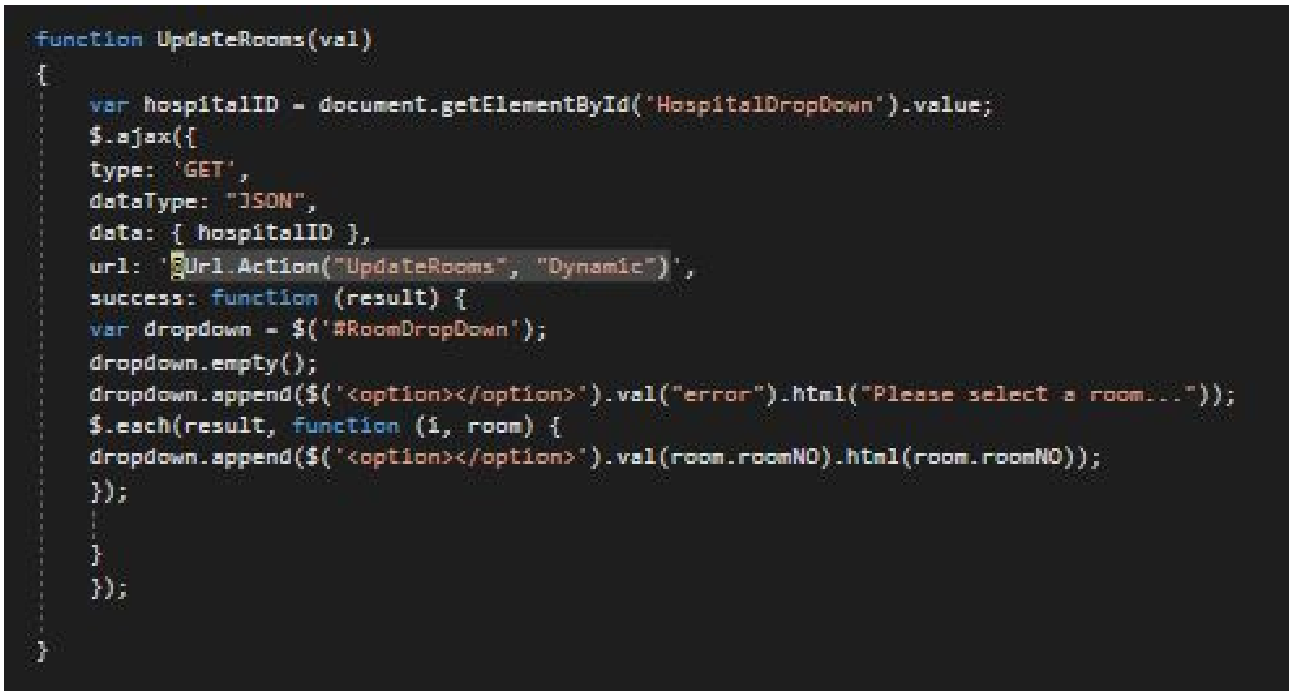

```
function UpdateRooms(val)
{
    var hospitalID = document.getElementById('HospitalDropDown').value;
    $.ajax({
    type: 'GET',
    dataType: "JSON",
    data: { hospitalID },
    url: '@Url.Action("UpdateRooms", "Dynamic")',
    success: function (result) {
    var dropdown = $('#RoomDropDown');
    dropdown.empty();
    dropdown.append($('<option></option>').val("error").html("Please select a room..."));
    $.each(result, function (i, room) {
    dropdown.append($('<option></option>').val(room.roomNO).html(room.roomNO));
    });

    }
    });

}
```

Figure 4.10: AJAX Call

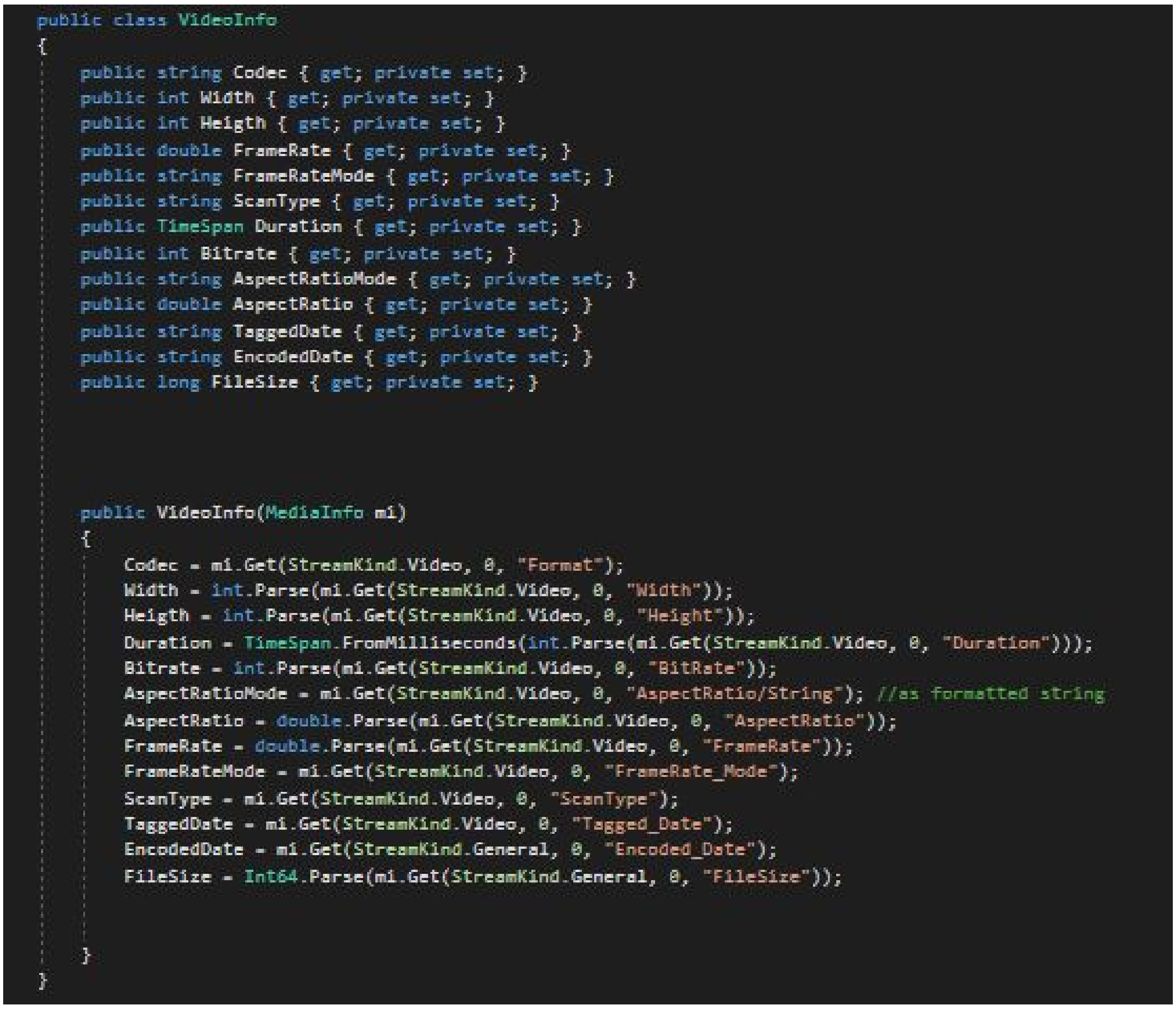
called (“NewOperation”) which is responsible for the HttpPost requests and takes as input the view model (NewOperationFormViewModel).

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 regisration 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



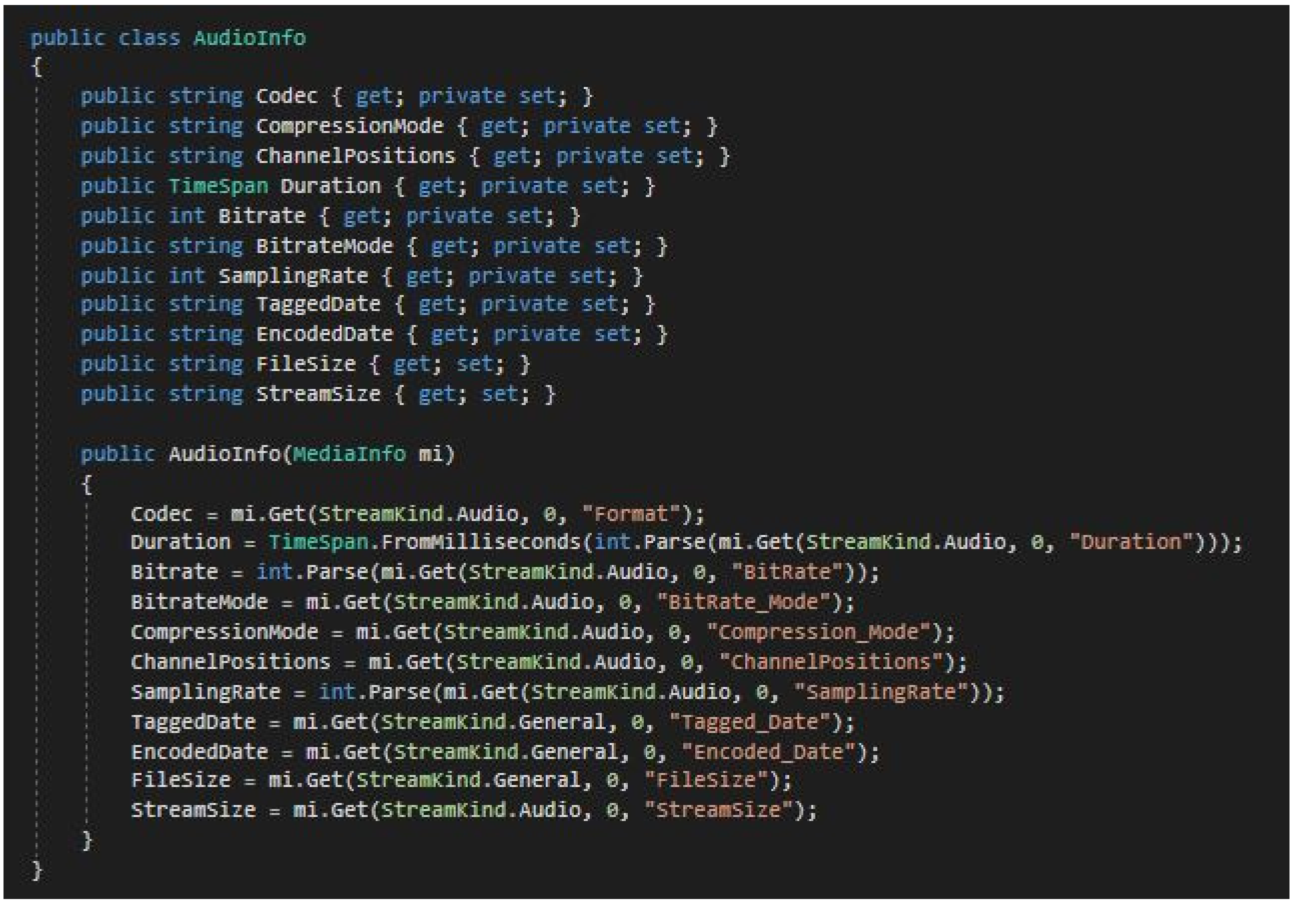
```
public class VideoInfo
{
    public string Codec { get; private set; }
    public int Width { get; private set; }
    public int Heigth { get; private set; }
    public double FrameRate { get; private set; }
    public string FrameRateMode { get; private set; }
    public string ScanType { get; private 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"));
        Heigth = 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

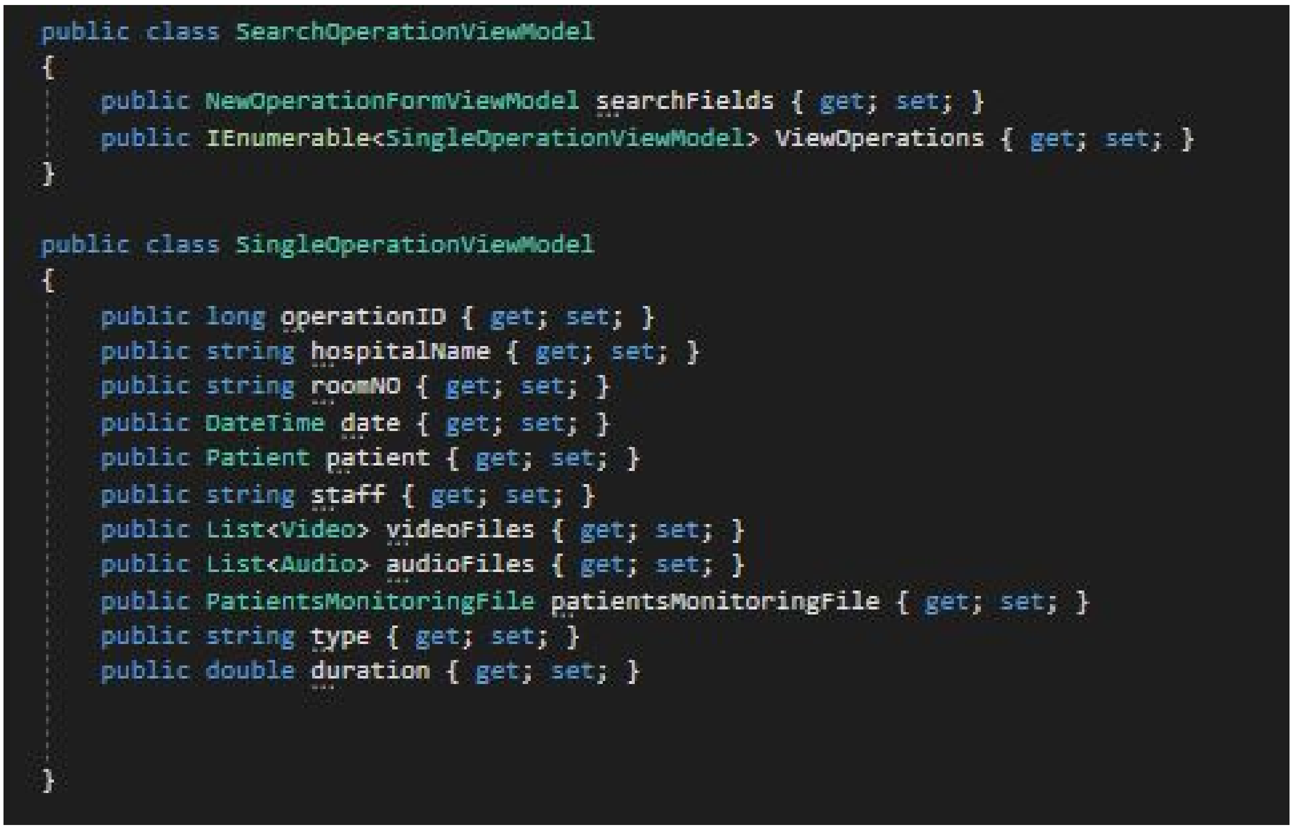
The last part of the application’s user interface is about displaying all the relevant information of an operation.From the home page, the user clicks on any listed operation and the details page will be loaded. It is important here to highlight that another controller method was deemed appropriate to 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");
    }
}
```

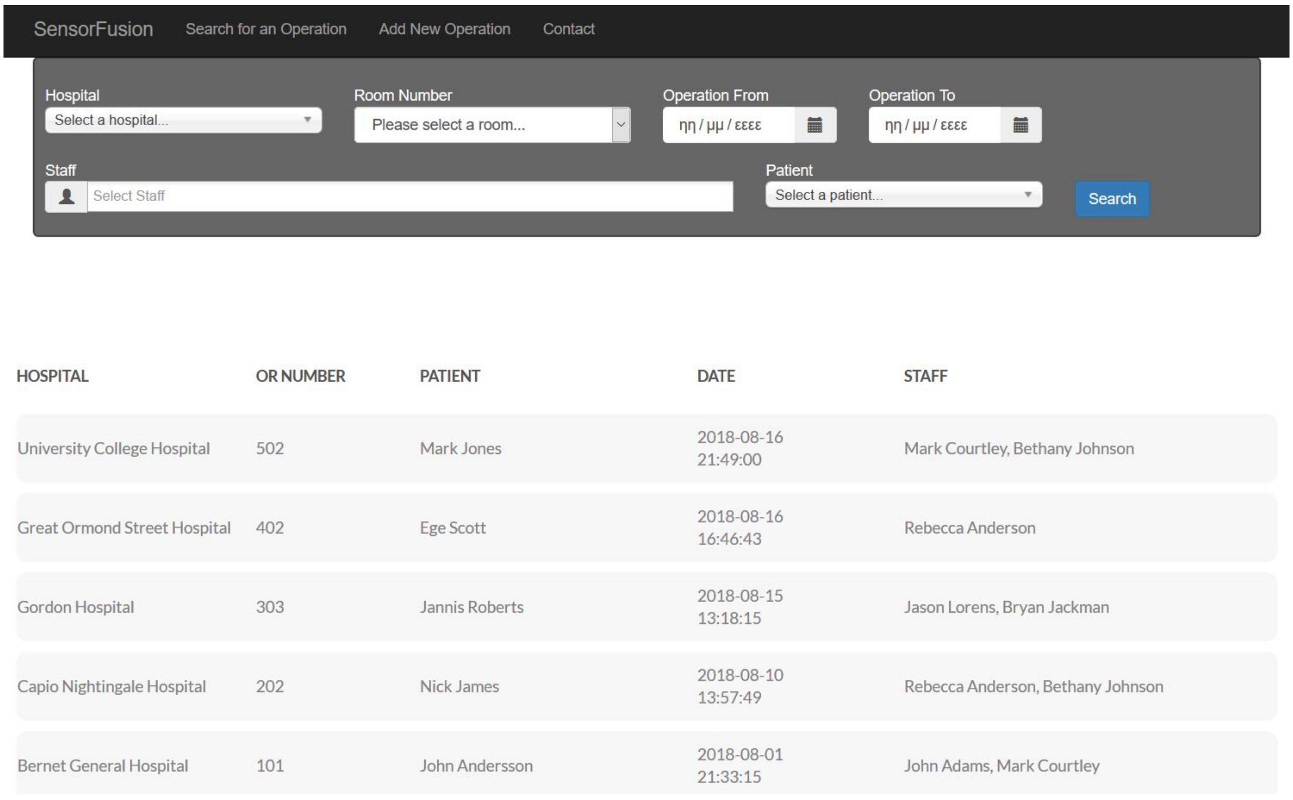
Figure 4.12: Extracting meta-data from audio files

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 methods 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



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 Critical 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, fro testing the correctness, completeness and quality of the application, the test process was divided into two main phases: unit testing [17] and user testing. Unit testing was performed throughout the development of the application while user testing was performed towards the completion of the development stage. In this chapter, the outcomes of both testing procedures will be discussed.

System testing is used to test the functionality of the entire system. Finally, acceptance testing is defined as that testing, which when completed successfully, will result in the customer accepting the software and giving us their money"[52]. Of these strategies the first three 39Chapter 5. Testing 40 were implemented as outlined in the remainder of this chapter. Acceptance testing wasn't applicable at this point as the application is more a proof of concept and only had to satisfy the author himself.

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 suit, which proves a professional test guide to help developers write test code for unit testing. In order to facilitate the unit testing of the application, three seperate 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.”

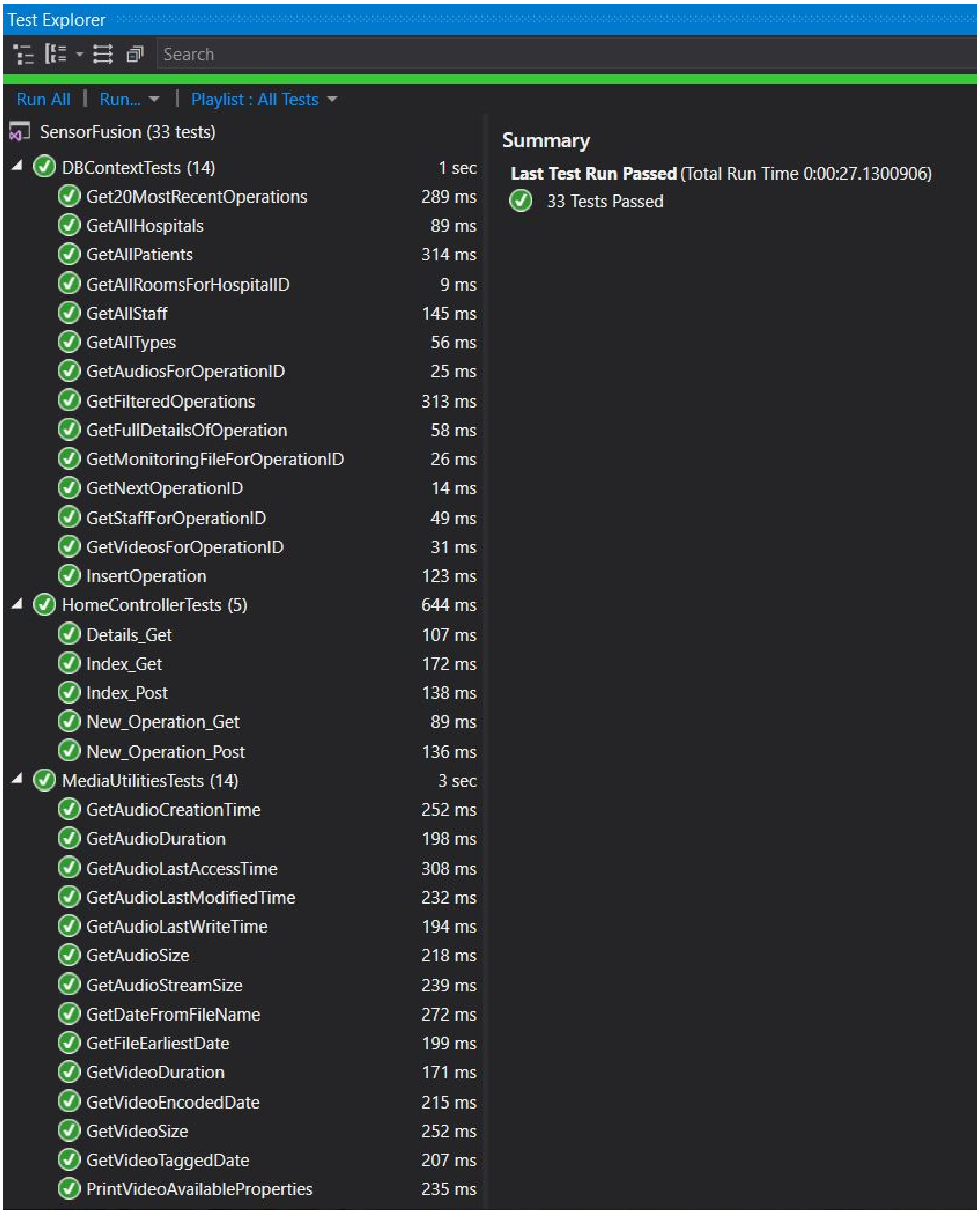
Being in line with the aforementioned approach,every time a single test was developed, the testing collection was run to ensure that all the previously created tests were successful and that the newly created test was failing. So, essentially, the tests for each method was created first and then the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

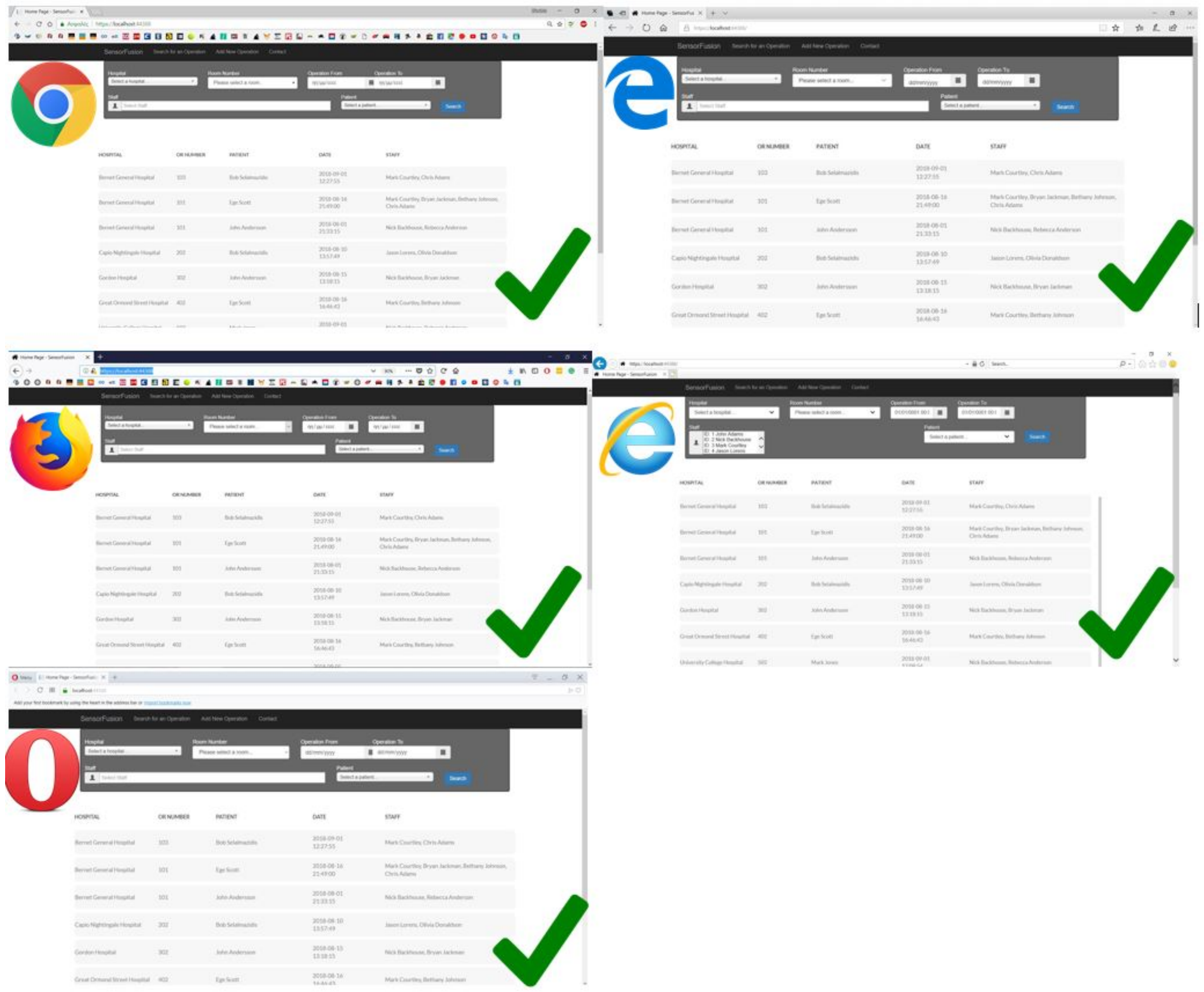
5.3 Acceptance Testing

ensures overall product is acceptable for delivery from the client perspective.



```
SensorFusion (33 tests)
◢ ✓ DBContextTests (14)                          1 sec
    ✓ Get20MostRecentOperations                  289 ms
    ✓ GetAllHospitals                             89 ms
    ✓ GetAllPatients                             314 ms
    ✓ GetAllRoomsForHospitalID                     9 ms
    ✓ GetAllStaff                                145 ms
    ✓ GetAllTypes                                 56 ms
    ✓ GetAudiosForOperationID                     25 ms
    ✓ GetFilteredOperations                      313 ms
    ✓ GetFullDetailsOfOperation                   58 ms
    ✓ GetMonitoringFileForOperationID             26 ms
    ✓ GetNextOperationID                          14 ms
    ✓ GetStaffForOperationID                      49 ms
    ✓ GetVideosForOperationID                     31 ms
    ✓ InsertOperation                            123 ms
◢ ✓ HomeControllerTests (5)                      644 ms
    ✓ Details_Get                                107 ms
    ✓ Index_Get                                  172 ms
    ✓ Index_Post                                 138 ms
    ✓ New_Operation_Get                           89 ms
    ✓ New_Operation_Post                         136 ms
◢ ✓ MediaUtilitiesTests (14)                     3 sec
    ✓ GetAudioCreationTime                       252 ms
    ✓ GetAudioDuration                           198 ms
    ✓ GetAudioLastAccessTime                     308 ms
    ✓ GetAudioLastModifiedTime                   232 ms
    ✓ GetAudioLastWriteTime                      194 ms
    ✓ GetAudioSize                               218 ms
    ✓ GetAudioStreamSize                         239 ms
    ✓ GetDateFromFileName                        272 ms
    ✓ GetFileEarliestDate                        199 ms
    ✓ GetVideoDuration                           171 ms
    ✓ GetVideoEncodedDate                        215 ms
    ✓ GetVideoSize                               252 ms
    ✓ GetVideoTaggedDate                         207 ms
    ✓ PrintVideoAvailableProperties              235 ms

Summary
Last Test Run Passed (Total Run Time 0:00:27.1300906)
✓ 33 Tests Passed
```

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



HOSPITAL	OR NUMBER	PATIENT	DATE	STAFF
Bernet General Hospital	103	Bob Selalmazidis	2018-09-01 12:27:55	Mark Courtley, Chris Adams
Bernet General Hospital	101	Ege Scott	2018-08-16 21:49:00	Mark Courtley, Bryan Jackman, Bethany Johnson, Chris Adams
Bernet General Hospital	101	John Anderson	2018-08-01 21:33:15	Nick Backhouse, Rebecca Anderson
Capio Nightingale Hospital	202	Bob Selalmazidis	2018-08-10 13:57:49	Jason Lorens, Olivia Donaldson
Gordon Hospital	302	John Anderson	2018-08-15 13:18:15	Nick Backhouse, Bryan Jackman
Great Ormond Street Hospital	402	Ege Scott	2018-08-16 16:46:43	Mark Courtley, Bethany Johnson

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

Chapter 6

Conclusion and Project evaluation

Lorem ipsum dolor sit amet, consectetuer 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, consectetuer 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.

Appendix A

Source Code

katherine hinke

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.

This document was set in the Times Roman typeface using LaTeX and BibTeX, composed with a text editor.