40329070

Software Architecture

SET10101 - Coursework

Contents

[Table of Figures 1](#_Toc531891520)

[Brief 3](#_Toc531891521)

[Architecture Recommendations 4](#_Toc531891522)

[Aspect Oriented 4](#_Toc531891523)

[What is it? 4](#_Toc531891524)

[Why use it? 5](#_Toc531891525)

[Advantages 5](#_Toc531891526)

[Disadvantages 5](#_Toc531891527)

[Repository 6](#_Toc531891528)

[What is it 6](#_Toc531891529)

[Why use it? 6](#_Toc531891530)

[Advantages 7](#_Toc531891531)

[Disadvantages 7](#_Toc531891532)

[Final Recommendation 8](#_Toc531891533)

[Repository 8](#_Toc531891534)

[Object oriented programming 8](#_Toc531891535)

[Design 9](#_Toc531891536)

[Class Diagram 9](#_Toc531891537)

[Activity Diagram 10](#_Toc531891538)

[Evaluation 13](#_Toc531891539)

[References 14](#_Toc531891540)

[Appendix A: Data Dictionary 15](#_Toc531891541)

[Appendix B: Class Diagrams 17](#_Toc531891542)

# 

# Table of Figures

[Figure 1 Aspect Oriented Architecture (Xiaodong, Liu 2018) 4](#_Toc531891479)

[Figure 2 - Basic Structure (Xiaodong Liu, 2018) 6](#_Toc531891480)

[Figure 3 View Class Diagram 9](#_Toc531891481)

[Figure 4 Activity Diagram: Operator 10](#_Toc531891482)

[Figure 5 Activity Diagram: PDA 11](#_Toc531891483)

[Figure 6 Activity Diagram: Hospital 12](#_Toc531891484)

[Figure 7 Patient Records 15](#_Toc531891485)

[Figure 8 Incident Records 15](#_Toc531891486)

[Figure 9 Ambulance Records 15](#_Toc531891487)

[Figure 10 Request Records 15](#_Toc531891488)

[Figure 11 Response Records 16](#_Toc531891489)

[Figure 12 Control Classes 17](file:///C:\Users\Richard\Documents\GitHub\SET10101\Documentation\Report.docx#_Toc531891490)

[Figure 13 Model Classes 18](file:///C:\Users\Richard\Documents\GitHub\SET10101\Documentation\Report.docx#_Toc531891491)

[Figure 14 Model Classes 18](file:///C:\Users\Richard\Documents\GitHub\SET10101\Documentation\Report.docx#_Toc531891492)

[Figure 16 View Classes 19](file:///C:\Users\Richard\Documents\GitHub\SET10101\Documentation\Report.docx#_Toc531891493)

[Figure 15 View Classes 19](file:///C:\Users\Richard\Documents\GitHub\SET10101\Documentation\Report.docx#_Toc531891494)

# Brief

An ambulance service would like to update its command and control system and link it to an existing database of medical records. The proposed name of the System is KwikMedical. The client would like a system that would allow a Telephone operator based at the NHS headquarters to enter information into the system, which will then check the information against existing patients within its database.

The system will then generate a request and will allow the operator to choose an ambulance response that will be best suited for the situation. The System will also have a smart phone application that will be used by the ambulance crew, which will extract the patient’s medical records and the incident notes. The ambulance crew can then submit their response notes, which can then be sent to a list of local hospitals as a Response.

The hospitals will also be running their own version of KwikMedical, which will allow them to generate a list of pending responses, as well as view the patient’s medical details, and passed visits to the hospital.

The client hopes that the additional information and features of the system will help them make more efficient rescues and provide data to support strategic decisions about their regional hospitals.

# Architecture Recommendations

Patient details already exist on databases that are accessed by MySQL, so it is critical that the software design can easily incorporate these databases. The two Architecture Systems that have been chosen are **Aspect-Oriented** and **Repository**.

## Aspect Oriented

When looking at larger systems, the relationship between requirements and components of a system are complex. A requirement may be implemented by numerous components and vice versa. This means that any changes to the requirements that need to be made may result in changing multiple components. Some other features, such as System Maintenance and component reuse are difficult to implement within a modular system, as they do not implement a single system abstraction, and code can end up dangled across the system.

### What is it?

**Aspect-Oriented Software Architecture (AOSA)** is an up and coming architecture that is intended to address these problems and make software easier to maintain and use. If software has several interconnected parts, it can be difficult to make changes to the code without it breaking somewhere else. AOSA addresses this issue by using a type of abstraction called an *aspect.* Aspects are used alongside other abstractions, such as *objects* and *methods*.

Fig

Figure 1 Aspect Oriented Architecture (Xiaodong, Liu 2018)

### Why use it?

One of the main benefits of using AOSA systems is that concerns are separated into independent elements rather than including concerns within the same logical abstraction. They are centred around a central data source, which other components can then extend off the central core. This makes the system modular and means that features can be added in future updates as an extension or plugin.

### Advantages

1. **Promotes Reusability**

* AOSA does not interfere with other parts of the code as it is accessed in an abstract way. This saves time and improves efficiency as the code can simply just be reused.

1. **Maintainable**

As each aspect is an extended from the central core, it is easy to edit an aspect without causing an effect on other aspects of the system. This means that it is easier to identify issues and in turn fix them

1. **Auxiliary Architecture**

* AOSA is an auxiliary architecture type, which means that it can be used in conjunction with other architectural styles to come up with the best solution.

### Disadvantages

1. **Difficulties in testing**
   * At the moment, there is no standards set in relation to testing AOSA. There are also a few issues with testing AOSA. Any tests conducted must be derived from the way the aspects are specified, and they must be able to be tested independently.
2. **Requires previous knowledge**
   * The use of aspects makes the program resemble a web, more than a linear style. Therefore, knowledge of how aspects work is important in understanding how to read the code.

## Repository

There are two main categories of Data-Centred Architecture; Blackboard and Repository. Both categories make use of a central data store, however, the blackboard style uses an active data store, while the repository uses a passive data store.

### What is it

The Repository style is a data centred style. It uses a passive repository that users can query and add data to. There are two components involved with the architecture; The data store, and the client’s system. The clients act as an agent, which connect to the database using a connection protocol within the language of their choice. For example, the *Java Database Connectivity* (JDBC) is an API for the Java language. Using the JDBC, SQL Queries are passed from the Client to the database and results are then passed back to the client.

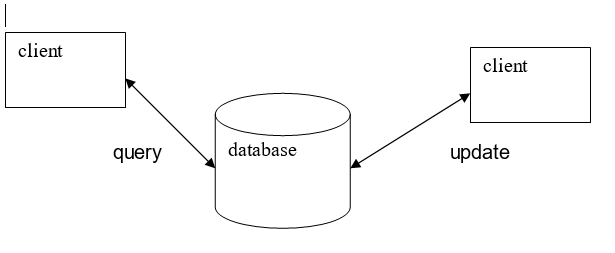


Figure 2 - Basic Structure (Xiaodong Liu, 2018)

### Why use it?

The existing databases for patient details are stored as passive data, so it would be a better idea to adopt this style. This style would allow the client to Query, Update, Insert and view data from the database in their own system via a GUI.

### Advantages

1. **Data is held securely and is easy to backup.** 
   * There are many database hosting solutions already available, such as WAMP, which can be used to create snapshots of data so if issues arise, the database can be rolled back to the last stable snapshot.
2. **The performance of the database can be scaled**
   * either by increasing the computing power of the server or machine that it is hosted on. This can be an expensive way to increase performance as the cost of newer hardware increases but is a straightforward process.
3. **Easily readable.**
   * The architecture style is based on a simple design. There is a central datastore, so it is easy to trace queries for troubleshooting.

### Disadvantages

1. **Data bottlenecks**
   * As there is only one central datastore that is connected to by all the clients, bottlenecks could arise if too many queries are sent at once, as an increased load could be put onto the database, causing performance problems.
2. **Points of failure.**
   * If the central data source fails, then the entire system fails.
3. **Data integrity.** 
   * Accurate data is important for the system to work correctly, so it is important that the datastore will not accept any invalid data.

# Final Recommendation

The chosen software architecture style will be a Repository, using Object-Oriented programming techniques, as many of the disadvantages of the repository style can be avoided through proper planning. The repository style will be used as an overall structure of the system, with a central datastore and a client system. The client system will be written in Java, using Object Oriented programming techniques. The source code will also be stored on GitHub, a version control repository.

## Repository

To optimise the existing databases, it is important to make use of the passive datastore holding the patient’s information. This style has been chosen so that the data is only being accessed when a client makes a request, rather than the data always being active, which may increase performance.

There will be one central database, called KwikMedical, with the patients, incidents, active ambulances, responses, and hospitals each being a table in their own right.

## Object oriented programming

The Model-View-Controller pattern is being used to develop an interactive system by splitting them into three components:

* Model: Contains the core functionality of the system and the Data
* View: Contains the GUI information
* Controller: Handles all the user input and logical processes.

This pattern uses encapsulation, as the components of the system only have access to what they need to, and the rest is hidden, which also allows code to be reused. Polymorphism is also being used to reuse code, as instead of creating separate classes for each patient, one patient class can be created, and then used as a basis to create an object from that class.

# Design

## Class Diagram

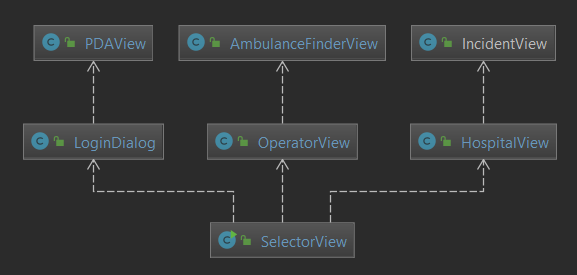


Figure 3 View Class Diagram

### Activity Diagram



Figure 4 Activity Diagram: Operator



Figure 5 Activity Diagram: PDA



Figure 6 Activity Diagram: Hospital

## Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Class | Description | Completed |
| 1 | Must Have | Receive input containing patient Details | Yes |
| 2 | Must Have | Verify that patient exists | Yes |
| 3 | Must Have | Assign patient to hospital and send records to that hospital. | Yes |
| 4 | Must Have | Operator assigns an ambulance to the patient | Yes |
| 5 | Must Have | Send request to allocated ambulance containing details | Yes |
| 6 | Must Have | Ambulance sends response details to selected Hospital | Yes |
| 7 | Must Have | Hospital receives and view response details | Yes |
|  |  |  |  |
| 8 | Should Have | Update a patient’s details with latest information | Operator can update Patient Details |
| 9 | Should Have | Add a patient to the database | Operator can add patient |
| 10 | Should Have | Close request once completed | No, Requests are saved to provide patient history |
| 11 | Should Have | Work out closest hospital | No |
| 12 | Should Have | Have a GUI to interact with the software | Yes |
| 13 | Should Have | Have input validation | Yes |
| 14 | Should Have | Prevent duplicate entries | Yes – Through Database set up of unique identifiers |
| 15 | Should Have | Code should be well-commented | Yes |
|  |  |  |  |
| 16 | Could Have | Ability to delete a patient | No |
|  |  |  |  |
| 17 | Won’t Have | “Real” components such as an actual PDA or real hospital | No |
| 18 | Won’t Have | Use GPS (Global Positioning System) to optimise vehicle assignment | No |
| 19 | Won’t Have | User Guides | No |

# References

Xiaodong Liu. (2018). Unit 10: Aspect-Oriented Architecture and Engineering.

# Appendix A: Data Dictionary

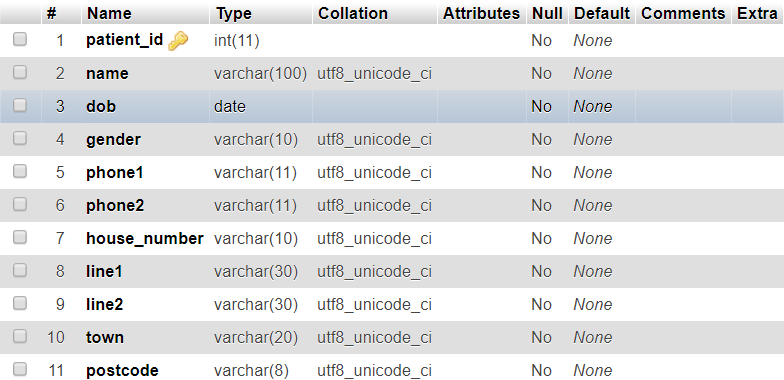


Figure 7 Patient Records

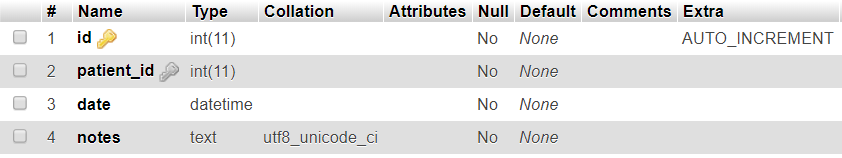


Figure 8 Incident Records

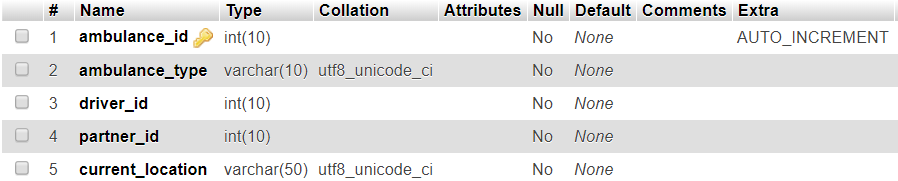


Figure 9 Ambulance Records

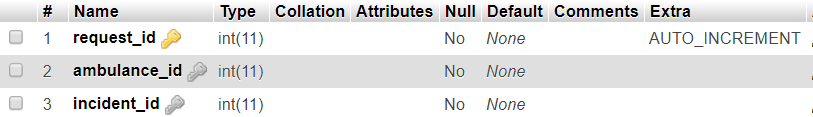


Figure 10 Request Records

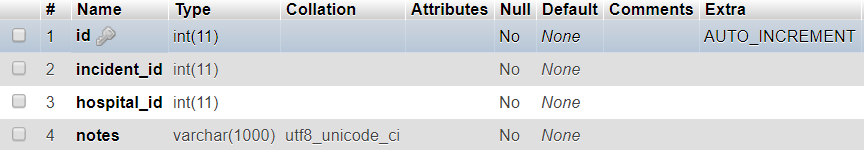


Figure 11 Response Records

# Appendix B: Class Diagrams

Figure 12 Control Classes

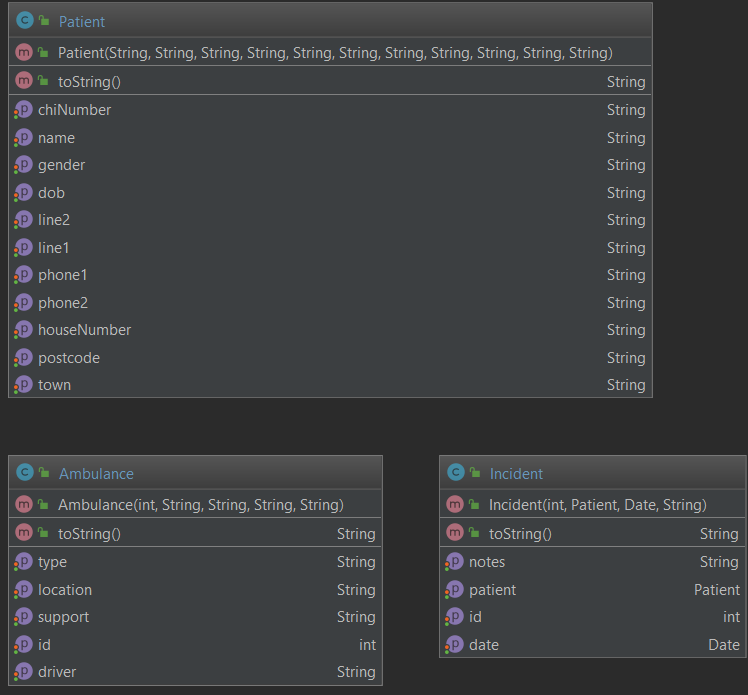


Figure 13 Model Classes

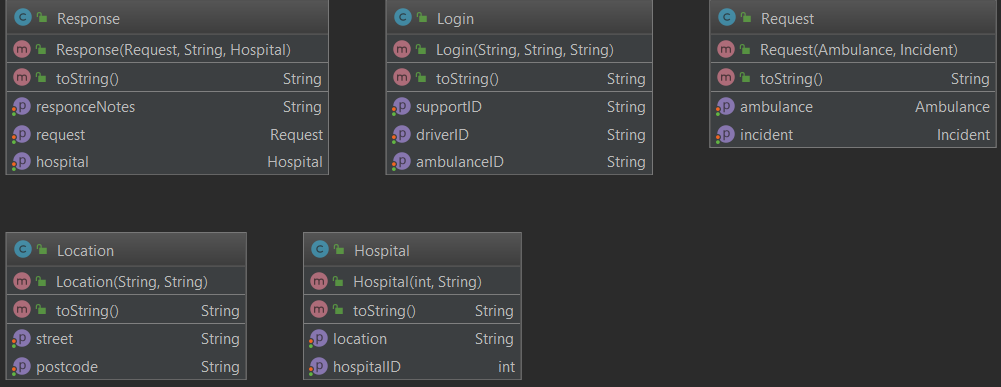


Figure 14 Model Classes

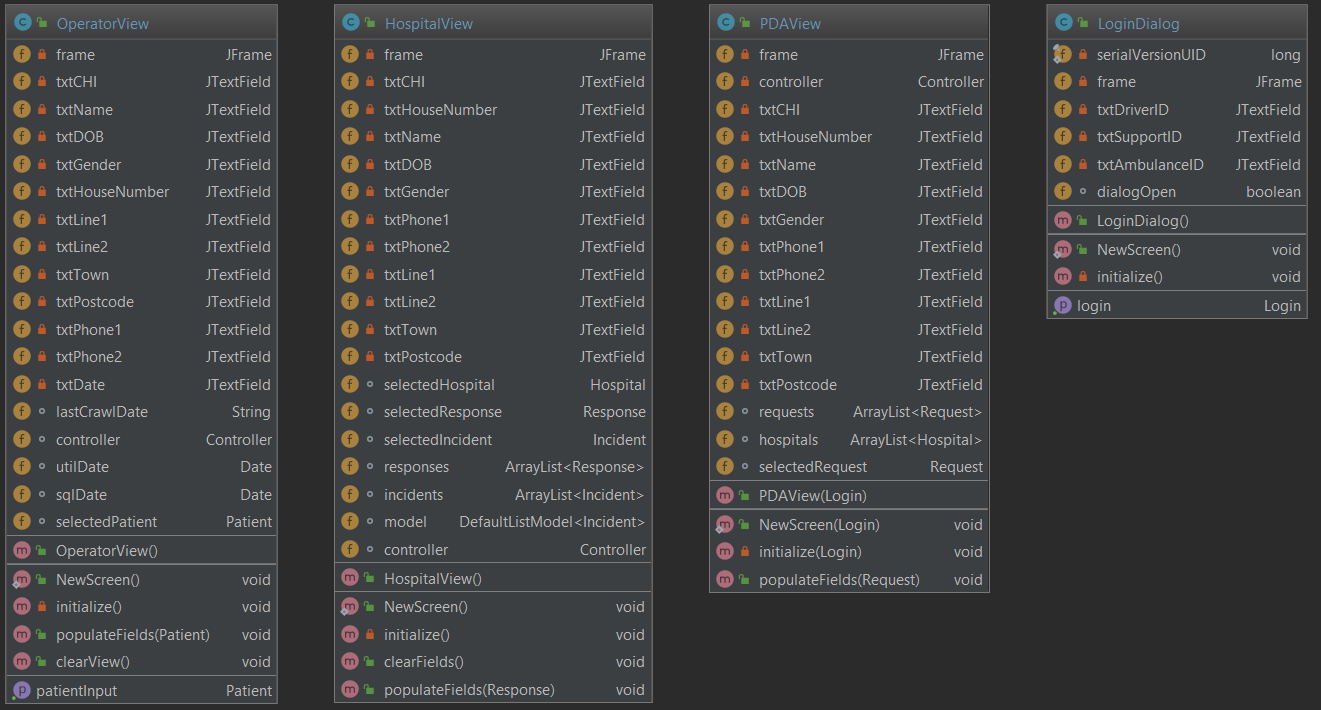


Figure 16 View Classes

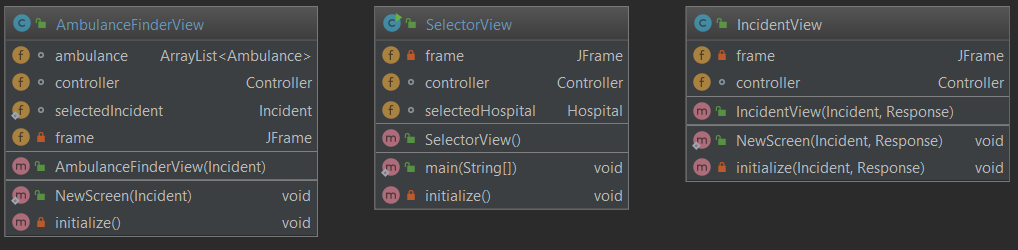


Figure 15 View Classes