

Development of an Electronic Medical Record System

Utveckling av ett elektroniskt journalsystem

Jemal Ramadan
Niclas Södermark



**KTH Information and
Communication Technology**

Examensarbete inom information- och
programvarusystem,
grundnivå
Högskoleingenjör

Degree Project in Information and Software Systems
First Level

Stockholm, Sweden 2014
Kurs II121X, 15hp

TRITA-ICT-EX-2014:20

Sammanfattning

Sofiaängen (<http://sofiaangen.com/>) är en psykoterapeutisk dagverksamhet och skola som ligger på Södermalm i Stockholm. Sofiaängen riktar sig till ungdomar i åldern 14 till 20 med psykiska och sociala problem.

Det nuvarande systemet som används av Sofiaängen för att hantera patientjournaler är i form av textfiler placerade i en delad mapp på en Mac-server, vilket varken är strukturerat, effektivt eller flexibelt. Syftet med detta examensarbete har därför varit att hitta det bästa sättet att implementera ett journalsystem för patientjournaler på, samt att utforma, utveckla och driftsätta den valda lösningen.

Arbetet delades in i fyra olika faser. Under den första fasen genomfördes en förstudie där möjliga implementeringar (fristående eller webbaserad) studerades och resultatet överlämnades till beställaren som fick välja den mest lämpliga lösningen. De tre övriga faserna var att utforma, genomföra (och testa) och distribuera den valda lösningen, samt att skriva en användarmanual.

Den valda lösningen, att skapa systemet som ett fristående program, implementerades med JavaFX:s plattform och MySQL:s databas-server. Vi använde en iterativ arbetsprocess för att stegvis utveckla de funktionella kraven. Detta för att kunna utvärdera vårt arbete tidigt, utifrån feedback på utvecklingen från beställaren, innan nästa uppsättning krav genomfördes. Examensarbetet har resulterat i ett journalsystem som uppfyller alla krav på funktionalitet och design av användargränssnittet som krävdes av Sofiaängen.

Nyckelord: EMRs, JavaFX, journalsystem

Abstract

Sofiaängen (<http://sofiaangen.com/>) is a psychotherapeutic day activities center and school located in Södermalm in Stockholm. Sofiaängen targets young people between the age of 14 and 20 and who have psychological or social problems.

The current system used by Sofiaängen for handling patient medical records is a Microsoft word file-system on a Mac server, which is neither efficient nor flexible. The objective of this bachelor thesis has therefore been to find the best way of implementing an EMRS as well as designing, developing and deploying the selected solution.

The work was divided into four different phases. The first phase was conducting feasibility study where possible implementations (standalone and web based) were studied. The result was submitted to Sofiaängen so they could make a selection. The second, third and fourth phases were respectively designing, implementing (and testing) and deploying the selected solution.

The selected solution was to implement the system as a standalone application. The system was implemented using the JavaFX platform and a MySQL database server.

We used an iterative software development model in order to implement the functional requirements incrementally and in doing so we were able to evaluate our work early, before the next set of requirements were implemented after getting feedback on the development from Sofiaängen.

The Degree project has resulted in an EMRS which fulfills all of the functional and look-and-feel requirements required by Sofiaängen.

Keywords: EMRS, JavaFX, journalsystem

Acknowledgement

We would like to thank all the personnel at Sofiaängen for their hospitality and cooperation specially psychotherapist Ingegerd Rindström who was our contact at Sofiaängen and who provided us with everything we asked for with great patience.

Finally we would like to thank our examiner Anders Sjögren for being there always for us when we needed him.

Table of Contents

Electronic Medical Record System (EMRS)	I
Sammanfattning	ii
Abstract	iii
Acknowledgement	iv
Terminology	viii
Abbreviations	viii
1 Introduction	1
1.1 Background and problem motivation	1
1.2 Goals	1
1.3 Limitations	1
1.4 Objectives (Concrete Goals)	1
1.5 Chapters	2
2 Background	3
2.1 Definition of terms	4
3 Method	6
3.1 Preliminary study	6
3.2 Development method	6
3.2 Project report	8
3.3 Testing	8
3.3.1 Unit Testing	8
3.3.2 Integration Testing	9
3.3.3 System Testing	9
3.4 Documentation	10
4 Solution options	11
4.1 Solution Options	11
4.1.1 Stand-Alone vs. Web-based EMRS	11
4.1.2 JavaFX vs. Swing	11
4.1.3 Database vs. File system	12
5 Implementation	13
5.1 Functional requirements	13
5.2 Functionality	13
5.3 Architecture and Design	13
5.3.1 Software Architecture	14
5.3.1 Design of the Graphic User Interfaces (Views)	17
5.3.2 Database Design	18

5.4 Implementation.....	19
5.5 Deployment.....	20
5.6 Testing	21
6 Results.....	22
6.1 Functionality	22
6.1.1 Logging in	22
6.1.2 Administrator	23
6.1.3 User	27
6.2 Logical division.....	32
6.3 Database.....	34
6.4 Documentation	34
7 Conclusions	35
7.1 Validity	35
7.2 Reliability.....	35
8 Future Development.....	36
8.1 Hash or encrypt Passwords.....	36
8.2 Logging for debugging	36
8.3 Support for the handicapped (Low vision)	36
REFERENCES	37
Appendix A: System Requirement Specification (SRS).....	39
Appendix B: Architecture Document	39
B1. Introduction.....	42
B1.1 Purpose	42
B1.2 Scope.....	42
B1.3 Definitions, Acronyms and Abbreviations.....	42
B1.4 References.....	42
B2. Architectural Representation	43
B3. Architectural Goals and Constraints.....	43
B3.1 Technical Platform.....	43
B3.2 Transaction.....	43
B3.3 Security.....	43
B3.4 Persistence	43
B4. Use-Case View	44
B4.1 List of use cases	44
B4.2 Use case diagram	45
B4.3 Use case description	46

B5. Logical view.....	47
B5.1 Architecture Overview – Package Layering.....	47
B5.2 Architecturally Significant Design Packages (Layers).....	47
B5.2.1 View (View-Layer).....	47
B5.2.2 Controller (Controller-Layer)	48
B5.2.3 Model (Model-Layer)	50
B5.3 Deployment View	52
Appendix C: Sample Source Code.....	53
Appendix D: Sample generated MR PDF.....	57

Terminology

Abbreviations

EMR(S)	Electronic Medical Record (System): A patient's medical record being stored in digital format rather than in traditional paper files.
MVC	Model-View-Controller design pattern that separates view from logic
GUI	Graphic user Interface
CRUD	Create, Read, Update and Delete (CRUD) are the four basic functions of persistent storage [1]
SRS	System Requirements Specification

1 Introduction

1.1 Background and problem motivation

An *Electronic Medical Record* (EMR) system is a rapid and efficient method to preserve and manipulate critical medical information. It enables users to keep track of the medical records over time and saves them a lot of time in their work hence improving the service they provide. Today most health care facilities in Sweden use an EMRS.

Sofiaängen is one of those health care facilities that do not use an EMRS. Sofiaängen is a psychotherapeutic day time activities center and school on Södermalm in Stockholm. This psychotherapeutic center and school is for young people between the age of 14 and 20 years and the school is an independent school with secondary and high-school levels.

Sofiaängen's school caters to young people who have psychological and/or social problems and who are in need of special support.

Prior to this project assignment Sofiaängen did not have an electronic medical record system. Their medical records and patients' information were stored in Microsoft Word files located in a shared folder on a Mac server. This is not a rapid and efficient method to preserve critical health information. The personnel had to enter and add health information in a free and non-organized style which made the medical journals lack uniformity and the collection and transmission of information unsystematic. Therefore we were given an assignment to create a stand-alone EMRS that solves the above mentioned problems.

1.2 Goals

The purpose of this project was to research how to best implement an EMRS that is rapid, efficient, secure and easy-to-use and then develop an EMRS that has a backup storage and user friendly interface which enhances the uniformity of the entry and presentation of patient information. The software should allow access only to authorized personnel and help the personnel to efficiently store and share patient information in a secure way.

1.3 Limitations

- We decided from the beginning that we were going to use NetBeans IDE [2] to develop the system since we are very familiar with NetBeans and have been using it for a long time.
- We decided to create the system in the Java programming language because it is the programming language we are most familiar with besides the language being cross-platform.

1.4 Objectives (Concrete Goals)

This project's goal was to achieve the following two-fold objective:

1. Operational objective – An EMRS would be designed and developed that fulfills all the customer's requirements.

2. Documentation objective - The EMRS should include documentation in the form of a user manual, architecture document, installation manual and a software requirements specification document.

1.5 Chapters

Here is a short description of the chapters in this report:

Chapter 1: Introduction - presents the project assignment with goals, limitations and methods.

Chapter 2: Background Material - explains the basic theory (concepts, technologies and techniques) needed to understand the report.

Chapter 3: Method - describes the software development model we used and the research we made prior to the development process.

Chapter 4: Solution options - describes the different implementation methods and technologies evaluated and used by the application.

Chapter 5: Implementation - explains how the selected implementation method and technologies in chapter 4 were implemented in the application.

Chapter 6: Result - describes the results achieved at the end of this project.

Chapter 7: Conclusion - conclusions of this assignment.

Chapter 8: Future Development - authors thoughts about possible future improvements to the application.

2 Background

There are many commercial EMR systems available today. According to the *Healthcare Information and Management Systems Society* (HIMSS) (<http://www.himss.org/>), there are more than 400 vendors for Electronic Medical Records systems in the USA alone [3]. Even though we couldn't find exactly how many EMRS vendors there are here in Sweden, we looked into some of the EMRS that are available today or are currently being used.

Today in hospitals, it is six medical record systems that dominate the Swedish market: [TakeCare](#), [Cosmic](#), [Melior](#), [Infomedix](#), VAS (Vårdadministrativt System) and SYSteam Cross. To understand how widely these systems are used in Sweden look at figure E1 in Appendix E as reported by the newspaper *Sjukhusläkaren* [4]. TakeCare is owned by a software company called *CompuGroup Medical* (CGM)[5], Cosmic by Cambio Healthcare Systems[6], Melior by Siemens and Infomedix by INFOMEDIX [7].

VAS is owned, developed and maintained by Norrbotten county council in cooperation with Jämtlands and Halland county councils [8] while SYSteam Cross's development projects are controlled by Sörmland county council. These systems are structured similarly. All are built in modules, but the modules are different in name and in content. Most of the features (functionalities) are similar in all the systems. All systems have functions for care documentation (medical recording), drug prescription, drug list and referral and response. They also have various planning functions such as calendar, booking and similar. In all of these systems, except VAS and SYSteam Cross, it is possible to install these modules independently which makes them adaptable to different health facilities that do not need all the functionalities of the system [9]. These systems can be used or adapted to all three parts of the patient care facilities; hospital, psychiatry and primary care [10].

PsykbBase [11] is another EMRS which is specifically developed for psychologists, psychotherapists and psychiatrists and is suited for small practices. It is available as a web based and as a standalone application. Some of the features offered by this system, besides keeping records especially suited for psychiatrists/psychologists, are accounting, calendar, report generation etc. One downside of this system is that the standalone version doesn't work on Mac computers. Another downside is, it is designed for single user not for a group of users.

There are even many EMRS that are licensed under an open source license or in the public domain for use in the healthcare industry. OpenEMR [12] is one example. It is a Free and Open Source Practice management and Electronic Medical Records software application and it features fully integrated electronic medical records, practice management, scheduling, electronic billing, internationalization, and free support. It can run on Microsoft systems, Unix-like systems (Linux, UNIX, and BSD systems), Mac OS X and other platforms [12]

Here are some of the free EMRS without going into details. For detailed information one can follow their respective references. VistA [13], GNU Health [14], Open Source Clinical Applications & Resources (OSCAR)[15].

These electronic medical records systems, which are commercially available, are big and have a lot of functions which are not necessary for a small private facility like Sofiaängen besides being expensive.

2.1 Definition of terms

In this section, we give a general description of the background to the technologies, platforms and libraries that have been relevant in the development of this medical record system.

EMR (Electronic Medical Record) - a patient's medical and health related history in digital format. It stores electronically all medical information of a patient's past and present health status and health care.[16]

Standalone application - a standalone application refers to an application running on a desktop environment such as Windows or Mac platforms that is the code runs on the client as well as the processing of data. The software may reside on the same machine or on a remote server (can be accessed without the Internet). [17]

Web-based application - a web-based application refers to any program that is accessed over a network connection using the HTTP protocol. Web-based applications often run inside a web browser. However, web-based applications may also be client-based, where a small part of the program is downloaded to a user's desktop, but processing is done over the Internet on an external server. [17]

JavaFX - a set of graphics and media packages that enable application developers to easily design create and deploy Rich Internet Applications (RIA) and Desktop applications that behave consistently across multiple platforms. Built on Java technology, the JavaFX platform provides a rich set of graphics and media API with high performance hardware-accelerated graphics and media engines that simplify the development of data-driven enterprise client applications. [18]

JavaFX Scene Builder - a key development tool for the JavaFX platform. It provides a visual layout environment that lets you quickly design user interfaces (UI) for JavaFX applications without needing to write any code. As you build the layout of your UI, the FXML code for the layout is automatically generated. JavaFX Scene Builder provides a simple yet intuitive interface that can help even nonprogrammers to quickly prototype interactive applications that connect UI components to the application logic. [18]

FXML - an XML-based language that provides the structure for building a user interface separate from the application logic code. [18]

Swing - a set of packages included in the Java Standard Edition and is used for creating graphical user interfaces (GUI: s). [19]

iText - a free and open source library for creating and manipulating PDF files in Java. [20]

Backup - the procedure for making extra copies of data that can be called on in the event that the original is either lost or damaged.

PreparedStatement - a Java object that represents a precompiled SQL statement.

3 Method

3.1 Preliminary Study

Before starting our project, a preliminary study was made to look into currently existing EMRS and see if we can find one that could fulfill Sofiaängen requirements. We also made a study to determine whether to implement the system as a stand-alone or web-based application in the case that we had to develop the system ourselves. We also researched what the law says regarding cloud services. We found out that there are a lot of rules and regulations to be followed as well as agreements to be made between the cloud service provider and the data controller demanding the cloud provider meets the requirements of the Swedish Personal Data Act [21] in order to use a cloud service for storing patient data. The results of the inquiries were then presented to Sofiaängen in order for them to make an informed decision. Based on that decision we were able to compare and choose different technologies (techniques) for the different parts of the system. The techniques which we compared and examined to create the system were JavaFX vs. Swing and different file systems vs. databases. See chapter 4 for the different techniques considered.

3.2 Development Method

There are several software development models that play a big role in guiding the software development process. Each one has its pros and cons, and we had to choose one that we thought was appropriate for our project.

At the beginning of this project, Sofiaängen didn't exactly know what they wanted and they had to discuss among themselves to figure that out. It was also difficult for us to understand all their needs exactly as they required and as a result we started with an incomplete requirement specification.

3.2.1 Candidate Development Methods

Here are the process models we considered and the reason we rejected them.

Waterfall model - This takes the fundamental process activities of specification, development, validation and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing and so on. The downside we found with this model is that it should only be used when the requirements are well understood and unlikely to change radically during system development, which was not the case here as we have explained earlier.[22]

Spiral model - In this approach the software process is represented as a spiral, rather than a sequence of activities with some backtracking from one activity to another. Each loop in the spiral represents a phase of the software process. Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design, and so on.[22]

This model focuses on identifying and eliminating high-risk problems (for example, if the intention is to use a new programming language, a risk is that the available compilers are unreliable or do not produce sufficiently efficient object code [23]) by careful process design.[24]. But we didn't expect high-risk problems in our software development as it is a small application.

Prototyping model [24] - This model requires developing a prototype to illustrate to the customer the functionalities of the system and gain a better understanding of the customer's needs. But our system is small and we didn't need to create a prototype.

3.1.2 The Chosen Development Method

We decided to use an iterative and incremental development process [22]. The reason we chose this model is to enable us to show the customer, iteratively, a system that can demonstrate some new functionality in action early in the development process without having to develop the whole system. As a result we get feedback from the customer early and the customer gets the opportunity to evaluate if their requirements are being met early.

The general iteration process in the iterative and incremental development process looks as follows (figure 3.1).

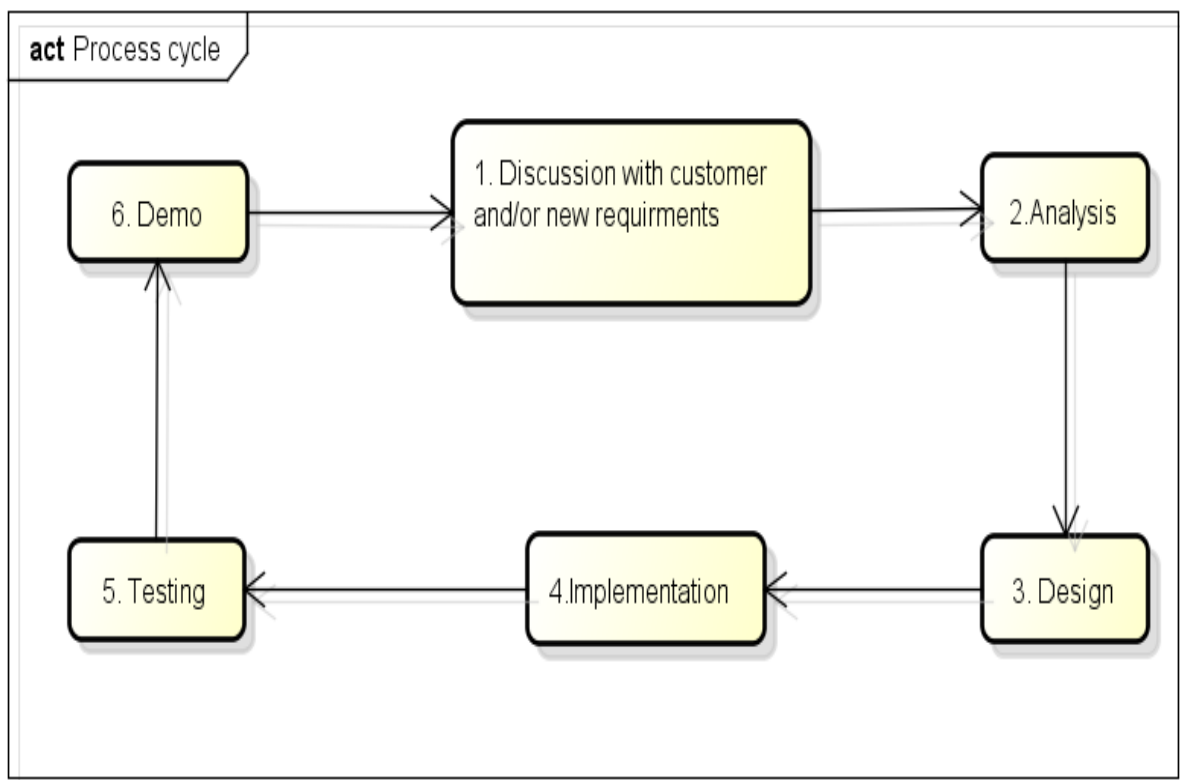


Figure 3.1 The Iterative process used in this project

1. Discussion with the customer to gather new or additional requirement specifications.
2. Study the specification for realism, consistency and completeness. Requirements that do not meet these criteria are further discussed with the customer.
3. Design how to best implement the specifications. Here data structures, interface representations and algorithmic details are designed.
4. Implement the design. This includes writing (or improving) source code, creating the database (data and data structures) and testing.
5. Function testing is performed to ensure that the system(new added functionalities) is working correctly and efficiently.
6. The final step is to demonstrate the result to the customer for feedback.

The work was divided into smaller iteration cycles and reported as various demos during the last step of every iteration phase.

The project was intended to have the following iteration cycles:

1. An initial graphical user interface (GUI) was designed using the JavaFX Scene Builder and presented to the customer.
2. A more refined GUI and login functionality that goes all the way to a database were developed.
3. The rest of the functionality (except converting medical records to pdf) were implemented e.g. creating and deleting (inactivating) a patient or user, adding records to a patient, giving different rights to a user etc.
4. At last the functionality of converting patient records in the database to PDF format was implemented.

3.2 Project report

We decided to follow the template for technical-report-writing given to us by our supervisor.

We started by writing the introduction, background theory, method, solution options and implementation chapters first since they didn't need the completion of the system. The rest of the report: result, discussion and future development were written after we completed the system and had deployed it.

3.3 Testing

To make sure or demonstrate that our system's functions were working according to the requirement specifications of Sofiaängen we had to perform different tests and debugging as often as necessary. To accomplish testing we utilized the three levels of testing *unit testing*, *integration testing* and *system testing* [22] but before any kind of testing could be done we had to install the MySQL server [25] as well as create a database instance in the MySQL server, since that is the system's storage (see section 4.1.3).

3.3.1 Unit Testing

In our software the different units consist of:

- Java entity (object) classes that represents the business logic, and their respective methods.

- The Java controller classes
- The database handler Java class. This class handles all the interaction and queries to the database.

Unit testing starts with the methods in the database handler class because it is the basis for subsequent tests depending on the database handler. Hence a test-class, for the object classes and the database handler class, connects to the database using a method in the database handler class and creates the database-tables (if they are not already created) and tests all the features (attributes and operations) of the object classes. Here the results are shown as data entered into or retrieved from the database.

The views, defined in FXML files created by the JavaFX Scene Builder, were easy to develop and could easily be connected to their corresponding scene controller classes (see section 5.3.1 for further information). In order to test the methods in the controller classes independently, it was not necessary to create specific test-classes because we could use the GUIs to test these methods by calling them with appropriate input parameters. Here unit testing ends in class testing, that is, when the interaction between the attributes and the methods are tested. The results are then shown in the standard output.

The GUI components (TextFields, Buttons etc.) were tested by entering different kinds of input. For example, entering too short or too long password and verifying that the system responded with the appropriate error message.

3.3.2 Integration Testing

The GUIs and their corresponding controller classes are already integrated in conducting unit tests on the methods of the controller classes. There are six scene controller classes each handling one of the six views/scenes and one main controller class that handles communication between the scene controllers and the backend classes (database and PDF-writer). Hence the integration test includes parameter passing and method invocation between the main controller and the other six controllers, between the main controller and the object classes and between the main controller and the database handler class. Here Thread integration testing [26] is used, that is, integrating the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually and then integrated with the already tested and integrated threads. The integrated threads must be tested in case the new added thread causes side effects. For example, the authentication and authorization functionality in our software requires the interaction of the following files LoginScene.fxml (the view), LoginController.java (the scene controller) and DatabaseHandler.java (the backend database class). See section 5.3.1 for further information on these files.

3.3.3 System Testing

Since we developed every part of the system ourselves, we don't see much difference between integration and system testing when we are testing it

ourselves, because here, similar to the integration tests, we would make sure that all the components interact correctly and transfer the right data at the right time across their interfaces. However, in the system testing, in addition to detecting defects, we tested that all requirements were being satisfied. That is, demonstrating, using simulated test data, that the system has properly implemented the requirements. System testing has also to be done on the customer site. Hence, we also performed deployment testing and then acceptance testing [26].

3.3.3.1 Deployment Testing

Sofiaängen have both Mac and windows computer, therefore, we had to install MySQL server first and then install the application and make sure it works on both operating systems.

3.3.3.2 Acceptance Testing

Here Sofiaängen would be the ones to test the system for its functionality, that is, to make sure the system worked and that it met their requirements.

3.4 Documentation

As part of the documentation of the system the following documents were created:

- Requirements specification document: specifying the user requirements.
- Architecture document.
- Installation document: describes how the system can be installed and configured.
- User manual: helps the user to understand how to use the system.

4 Solution options

This section aims at presenting and explaining the different implementation methods and technologies we considered for the EMRS.

4.1 Solution Options

This section aims at presenting the two implementation options we considered for the EMR system. The first option is stand-alone application and the second option is web-based application.

4.1.1 Stand-Alone vs. Web-based EMRS

4.1.1.1 Stand-Alone EMRS

This is a medical records program in which the software may reside on the same machine or on a remote server.

Advantages:

- Better performance (speed).
- No Internet connection needed.
- Full control over the application and data. Hence, better security.

Disadvantages:

- Manual updating may be required.
- Most of the time they are only usable within a certain location.

4.1.1.2 Web-based EMRS

A web-based EMRS runs on the web server. For this type, all one needs is an Internet connection.

Advantage:

- Convenient for users to access the application from any location using the Internet.

Disadvantages:

- No full control over own data and application (controlled by third party). Hence vulnerable to different threats or risks
- No access to own data and application when there is no Internet connection.

4.1.1.3 Selected implementation

Having considered the pros and cons of both implementations Sofiaängen selected to implement the EMRS as a stand-alone application based on security, connectivity (availability) and cost factor. See REQ 10, 11 and 12 in the SRS document in appendix B. In other words, a standalone application fulfills the aforementioned nonfunctional requirements.

4.1.2 JavaFX vs. Swing

Even though Swing is the only GUI toolkit we have used when developing different applications in the past we chose JavaFX to build our GUI not based

on comparison with Swing but for the simple reason that Oracle recommends using JavaFX when building new applications.

Some of the benefits of using JavaFX are:

Familiar Java development tools: Developers can use their favorite Java development tools, such as the NetBeans and Eclipse IDEs, for development of JavaFX applications. All the standard IDE features, including syntax highlighting, auto-completion, interactive step-through debugging, and profiling are available for JavaFX developers. In addition, JavaFX Scene Builder offers an intuitive visual layout environment. [27]

Deploy on the desktop or in the browser: JavaFX applications can be deployed either as desktop applications or in browser applets with the help of the Java browser plugin. The improved Java browser plugin provides JavaFX applications a secure way to run inside a browser. When deployed on the desktop, JavaFX applications get the benefits of better performance and native operative system integration. [27]

4.1.3 Database vs. File system

Both databases and file systems are used for storing data with integrity and they both provide concurrency but databases have some important features that make them better data managers. These features that database have and most file-systems lack are:

Transaction is the property that an operation, that changes the contents of the data, has to succeed or fail in its entirety. That means if that operation fails after partially changing the data then the data would return to the state it was before the operation started and if it succeeds then the changes are stored.

Fast Indexing is being able to query the database based on different attributes. Attributes can be, for example, patient record types like psychological records, treatment records etc. Hence we can retrieve all patient records or only psychological records for example.

Snapshot is a copy of the database taken at a particular point in time without interrupting the services provided by the database. This copy can be used as a backup in case of an error.

Relational View of Data Databases allow us to define relationships between different data and then store these relationships themselves with the data in the database. Hence this property enables us to retrieve related data easily.

So as a result of the above mentioned features that databases have, we decided to use MySQL server as data storage for our system and convert the medical records to PDF as backup.

5 Implementation

The method we used consisted of four phases. The first phase consisted of the preliminary study presented in Chapter 3 section 3.1. The second phase was the design of the program. During the third phase implementation and tests were performed. This phase was repeated several times as new bugs or requirements came up. The fourth phase consisted of writing the user manual for the system and deploying the finished EMR system in the live environment.

5.1 Functional requirements

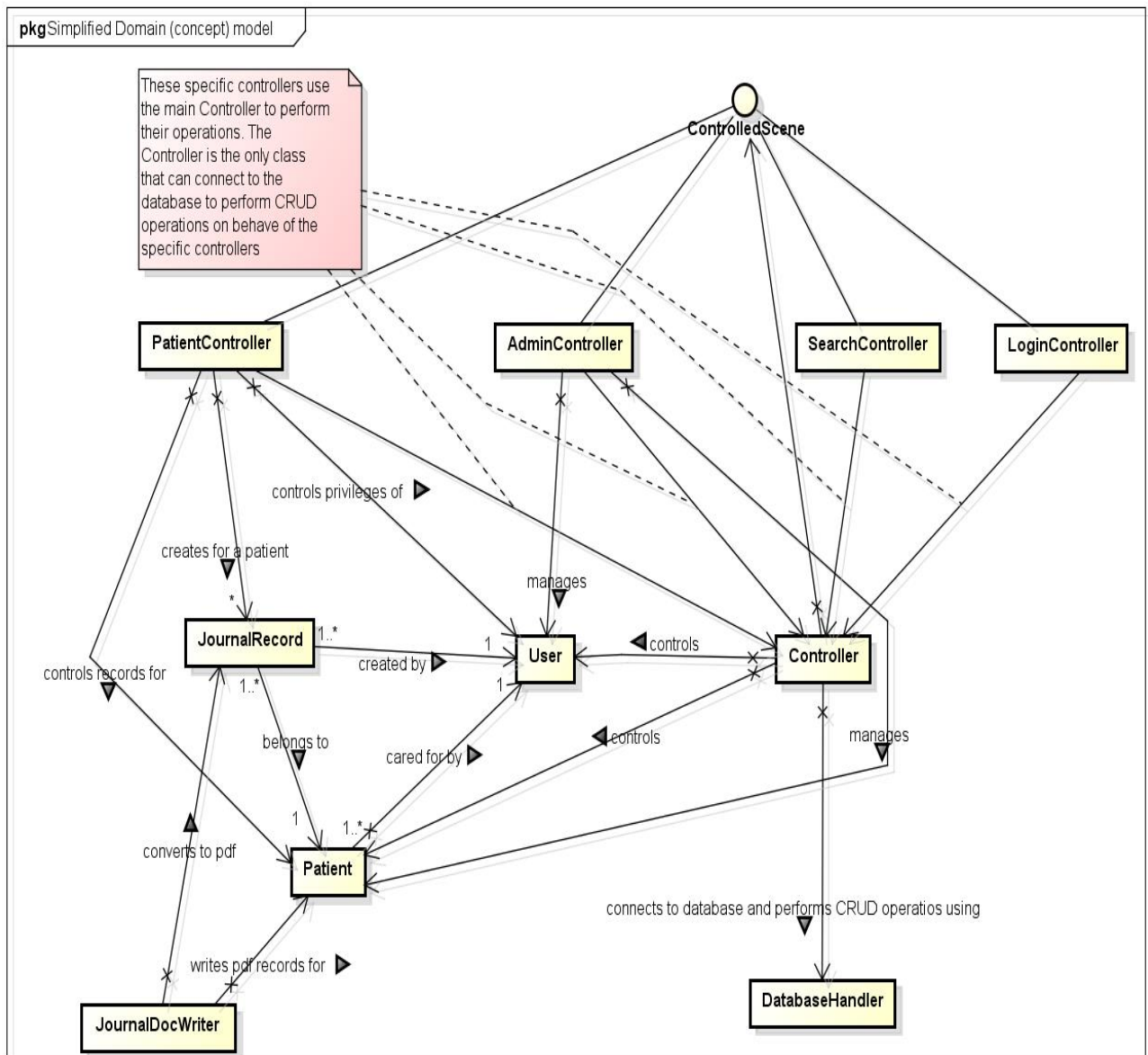
We met with Sofiaängen several times to collect the different functional requirements they had for the system. We categorized these functional requirements into two key functional areas that are critical to the definition of an EMR: (i) basic demographic and clinical health information, and; (ii) security and confidentiality. See the System Requirements Specification (SRS) in appendix A.

5.2 Functionality

To determine which functionalities were actually possible to implement discussion was held with the customer. We explained to Sofiaängen's representatives that all requirements were feasible to implement except requirement number 7, **REQ 7: Report generation**, i.e. converting stored data into Microsoft Word format (see the System Requirements Specification (SRS) in appendix A). The reason for this is because we could not find any Java tool that converts Java objects to Microsoft Word files. Therefore, it was decided to convert the medical records to PDF format instead.

5.3 Architecture and Design

The following figure depicts a simplified concept model of the system by showing the main entities (Java objects) and their relationships with each other and other controller and help objects.



powered by Astah

Figure 5.1 Conceptual model of the EMRS

5.3.1 Software Architecture

Even though we have developed an architecture document for the system (see appendix B: Architecture document) and explained the architectural overview of the system in it, we think it is necessary to mention some things about the architecture pattern we chose and the reasoning behind it.

We decided to use the MVC (Model View Controller) architecture-pattern [22]. The reason is we knew more and more requirements would come from the customer and using the MVC pattern would enable us to separate the three layers, view, controller, and model, from each other resulting in fewer dependencies. Hence adding new code for new requirements would be relatively easy. For example, adding a new view or changing an existing view could be done with-

out any changes to the underlying data in the model. And that is the reason we chose MVC pattern so the system can be maintained and improved while being developed and in the future. It also made it easier to write unit tests while maintaining and extending our application and it also enabled us to divide the work between us.

In a typical MVC architecture, a controller would trigger state changes to the model based on user events in the view as picture 2 shows. The model then notifies the view of the change. [22]

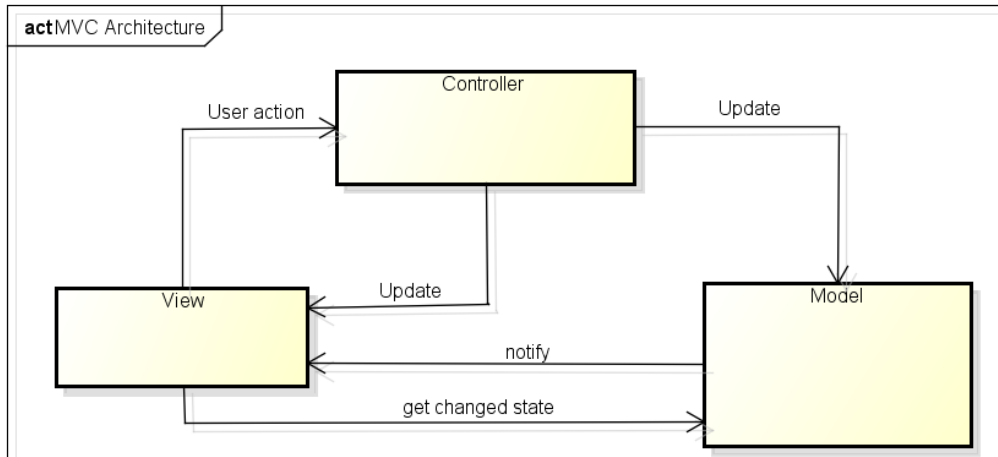
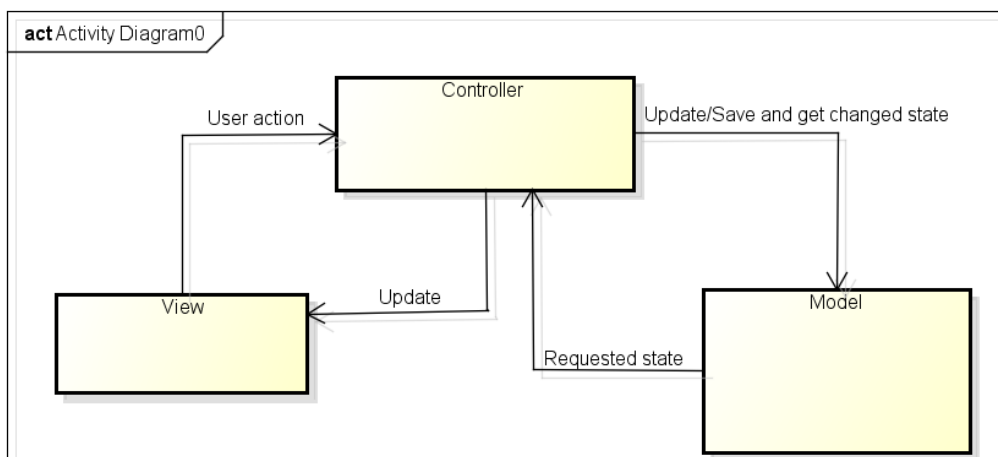


Figure 5.2 Traditional MVC.

We decided to use another variant of MVC pattern that completely decouples the view and the model as figure 5.3 shows. The controller takes care of the bindings between view-model and model-view. We chose this pattern because of the growing concern that the observer pattern is prone to error [28] and the systems doesn't allow deleting anything important related to medical records from the database except storing new records hence the absence of an observer will not cause problems.



powered by Astah

Figure 5.3 Alternative MVC-variant.

View Layer - Consists of several JavaFX GUI:s a.k.a. scenes. It accepts input and/or commands from the user and passes it to the model layer through the controller layer and it also presents the data coming from the model layer through the controller layer.

This layer contains in total six FXML files (one for each separate scene). These files define the user interfaces of the system. The view layer also contains one Java class (*JournalSystem.java*) with the main method and it is responsible for starting the application. These six files are: *JournalSystem.java*, *LoginScene.fxml*, *AdminScene.fxml*, *SearchScene.fxml*, *PatientScene.fxml*, *ShowRecordScene.fxml*, and *DialogScene.fxml*.

For the UML class diagram see figure 11 in section **6.2 Logical division** or in Appendix A: Architecture Document, section **BB5.2.1 View (View Layer)**.

Controller layer - This layer sends commands to the model to update the model's state. It also sends commands to the active view to change the view to reflect the current state of the model.

This layer contains seven controllers and one interface. They are: *LoginController.java*, *AdminController.java*, *SearchController.java*, *PatientController.java*, *ShowRecordController.java*, and *DialogController.java* and *Controller.java*.

For the UML class diagram see figure 11 in section **6.2 Logical Division** or in Appendix B: Architecture Document, section **B5.2.2 Controller (Controller Layer)**.

Model layer - This layer implements the business logic. It is the layer that interacts with the database. It retrieves data from the database and stores data entered by the user in the database. For example saving user or patient data, saving patient and records associations, storing and retrieving patient records, converting records to PDF files.

See figure **D1 in Appendix D** for a sample of a patient's medical journal converted into a PDF file.

The classes in this layer are: *Patient.java*, *User.java*, *JournalRecord.java*, *DatabaseHandler.java*, *JournalDocWriter.java*, *PersonNumber.java* and *DateHandler.java*.

For the UML class diagram see Figure 11 in section **6.2 Logical Division** or in Appendix A: Architecture Document, section **5.2.3 Model (Model Layer)**.

5.3.1 Design of the Graphic User Interfaces (Views)

To organize the user interface elements in the manner that we had imagined and thought would be user-friendly we needed to use a flexible layout manager that would allow us to position elements in an intuitive way.

Instead of coding the interfaces ourselves using FXML we used JavaFX SceneBuilder, see the *figure 5.4* below, to design our interfaces. All we had to do was drag and drop the UI components where we wanted them and give the required properties to the components, like assigning unique names to the components. As a result the desired FXML file for each designed view was generated automatically. JavaFX SceneBuilder also allowed us to register a controller class (for each generated FXML file) that handles the user interaction. The look and feel of our interfaces were achieved using cascading style sheets (CSS).

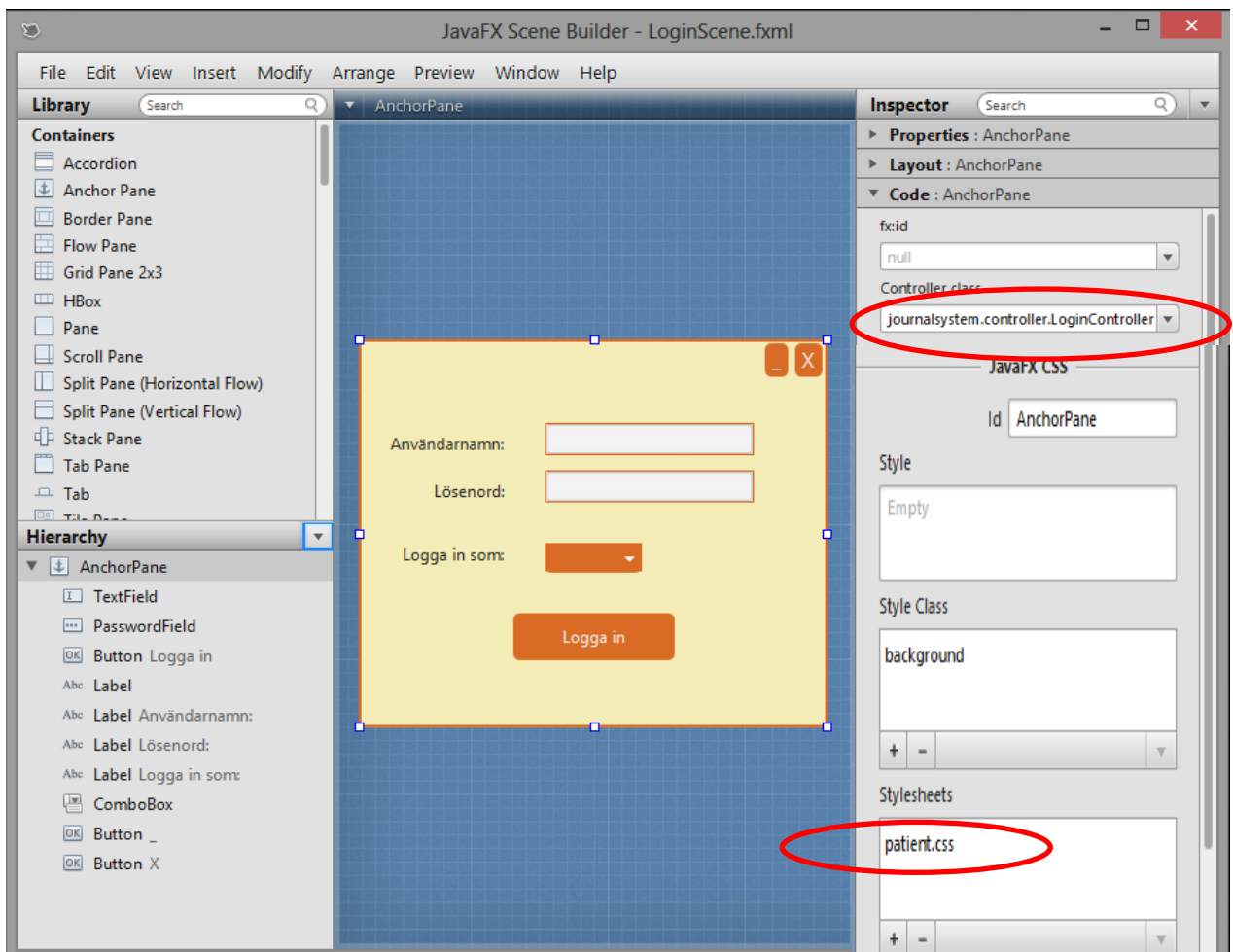


Figure 5.4 Design of the login view using JavaFX Scene Builder

Figure 5.4 above shows, as an example, the login view/scene for the Sofiaängen EMRS being designed using JavaFX Scene Builder. The file *patient.css* is the style sheet used for the login view and the controller class for this view is

journalssystem.controller.LoginController.java as shown in circled with red circles. When the "Logga in"-button is pressed, the registered controller, *LoginController.java*, gets the user name and password, authenticates the user (via the model) and decides the next course of action (show login error message or allow the user to log in to the system).

5.3.2 Database Design

The EMRS uses a MySQL server for data storage.

MySQL is one of the world's most widely used open source database and is available in a community version that runs under the GPL (GNU General Public License) and is free to use. It is also available in several versions that contain additional functionality and support services that can be obtained for certain expenses. MySQL can be used for most platforms and is used by many large companies and websites like Google, YouTube, Adobe and Facebook. [29]

We chose not to use the O/R (Object Relational) – mapping [30] in our application because the database was relatively small and not particularly complex. We used JDBC and PreparedStatements and mapped tables against our own entity classes which are just ordinary POJO:s (Plain Old Java Objects) [31]. These entity classes are:

1. *User.java*: Contains information about a user including name, password, and privileges.
2. *Patient.java*: Contains information about a patient including social security number, name, address and care giver.
3. *JournalRecord.java*: Contains information about a journal entry including journal type, category, care giver and patient.

We thought that these POJO:s were enough to represent all the information and relationships that exist between the care givers (medical doctor, psychiatrists, nurses and personal assistants) and the patients.

On the MySQL server, we created a normalized database called *journal_db* that contains five tables as can be viewed in the database structure (design) as generated by MySQL Admin tool in the figure on the following page.

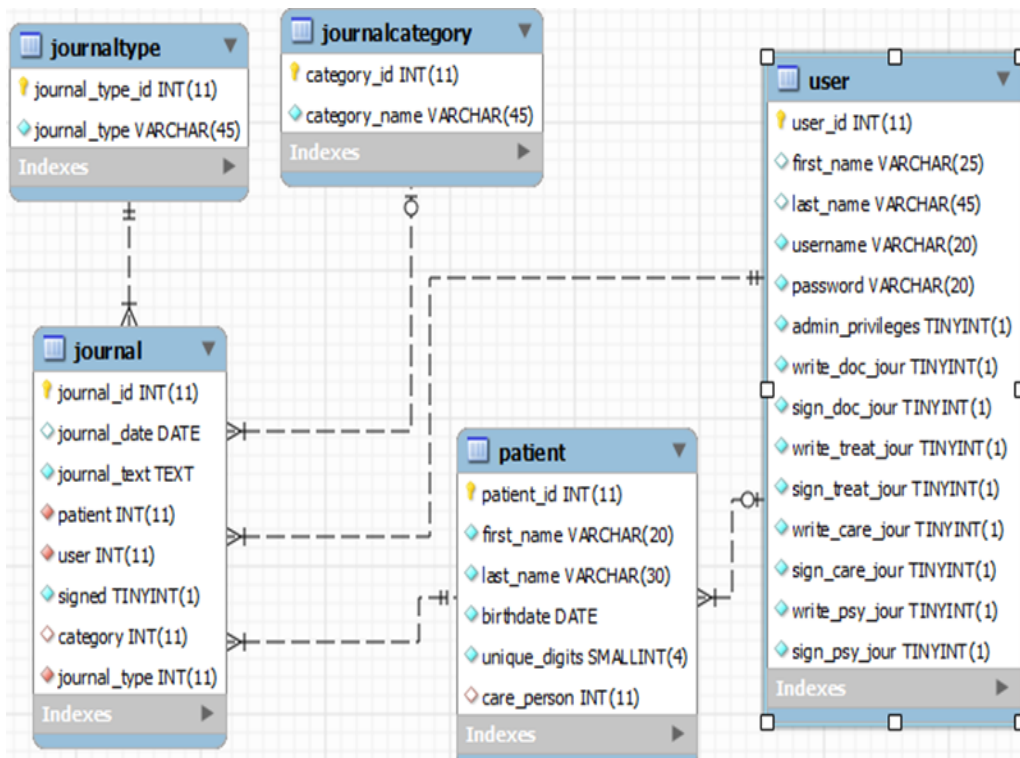


Figure 5.5 shows the relationship that exists between patient, journal and user entities.

According to figure 5.5, a user (care giver) can take responsibility for many patients and can create many journals for all the patients. A patient can have only one responsible care giver and many journals, while a journal belongs to only one patient. A journal must belong to one category and one type.

5.4 Implementation

To develop, test, and debug the Sofiaängen EMRS, we used local environments, i.e. our own computers, and consist of the following:

- A client computer running Windows 7 Enterprise.
- NetBeans IDE 7.3, for developing and debugging of the system and it also supports JavaFX 2
- MySQL Community Server for data storage.

For the system to show or enter the requested information, events caused by the user must be handled. Events are notifications that a user has pressed a key, clicked a button, or moved a mouse. Registered event handlers in the controllers then receive the event and provide a response.

As we have mentioned earlier, FXML source files are XML files (generated by the JavaFX Scene Builder) in which we defined our user interfaces.

With each FXML file, it is possible to associate a controller class defined for that FXML file. That controller class will make the link between the UI (the objects defined in the FXML) and the application logic. To each object defined in FXML, we can associate an *fx:id* and/or *onAction* attribute. The value of the id must be unique within the scope of the FXML, and is the name of an instance variable inside the controller class, in which the object will be injected and the

value of the *onAction* is an event handler method in the controller. [32]

As an example, figure C2 and C3 in Appendix C show parts of the *SearchScene.fxml* file and its controller (*SearchController.java*) respectively. In this example suppose we want to have access to the *TextFields* and to handle the event caused by pressing the button (see figure C1 in Appendix C) then we will need to add *fx:ids* attributes to the *TextFields* and *onAction* attribute to the button in *SearchScene.fxml* file, and declare variables and define an event handling method in the controller class with the same names as the values of these attributes. The variables and the method must be annotated with *@FXML*. See the red-underlined lines in figures C2 and C3 in Appendix C.

Since now the interfaces and the controllers are connected, the controllers now can get or store the required information by accessing the database with the *DatabaseHandler.java* file in the model. Figure C4 (Appendix C) shows part of the *DatabaseHandler.java* where a controller can use to get a patient by social security number. The database handler uses *PreparedStatement*s to query the database. See Appendix C for sample source codes.

5.5 Deployment

We started our deployment processes by installing MySQL server on Sofiaängen Mac server and since Sofiaängen is a small clinic with few computers, we decided to package the application in the form of an executable JAR file. The application package is available on a local drive and one can start the application by double-clicking the application JAR file. This JAR file contains a *.properties* file that contains the following properties:

- URL to the Sofiaängen Mac server and MySQL port number.
- Name of the database instance.
- Username and password to the database.
- URL to the base directory where the patients' printed PDF records would be stored.

See the “Deployment View” in the Architecture Document in Appendix B.

5.6 Testing

Unit, integration and system testing on our development environment went smoothly, just as they were described in section **3.3.1 Unit Testing**, **3.3.2 Integration Testing** and **3.3.3 System Testing**. However on site, at the customer's work place, during deployment testing we found some bugs that took a long time to figure out.

One of the bugs we found during deployment testing on site was related to file naming in Mac (for the application folder) and as we found out (by accident) after we had spent a lot of time to solve this bug is that the Mac operating system doesn't accept spaces in folder names, when executing JAR files. Hence it is better to avoid spaces if you can.

The second bug is related to formulating the URL for the base directory, where the patients' PDF files are to be stored, in the properties file to accommodate special characters (ä, å and ö). We thought that the character encoding of the file system of Sofiaängens server doesn't match the character encoding of the client computers, where the application is located. This leads to the PDF files not being stored in the location specified in the .properties file. This problem was solved when Sofiaängen helped us by removing the alphabets: ä, å and ö from the name of the folder that contains their patients' records.

Right now the system can successfully convert records to PDF (see Figure D1, Appendix D) and store them in the specified location.

6 Results

The result of this project is a fully functional EMRS that fulfills all the requirements of the customer. It has a user-friendly GUI and shows relevant error and informative messages to make it easy for the user to understand what is happening.

6.1 Functionality

There are only two actors in the system. One is the *administrator* and the other is the *user* which can be anyone who has the credentials to log in to the system. Users have titles assigned to them when they are created, like "behandlare" (care giver), "sjuksköterska" (nurse), "psykoterapeut" (psychotherapist) and "konsultläkare" (consulting doctor).

The use cases description and UML diagram for all the actors and system operations can be found in the Architecture Document, B4 Use Case View in Appendix B.

6.1.1 Logging in

Every user must login in to the system before he/she can have access to the system.

The following UML sequence diagram describes how the main components of the system interact to accomplish the log in use case and describes how the internal parts of the component interact to achieve the result required.

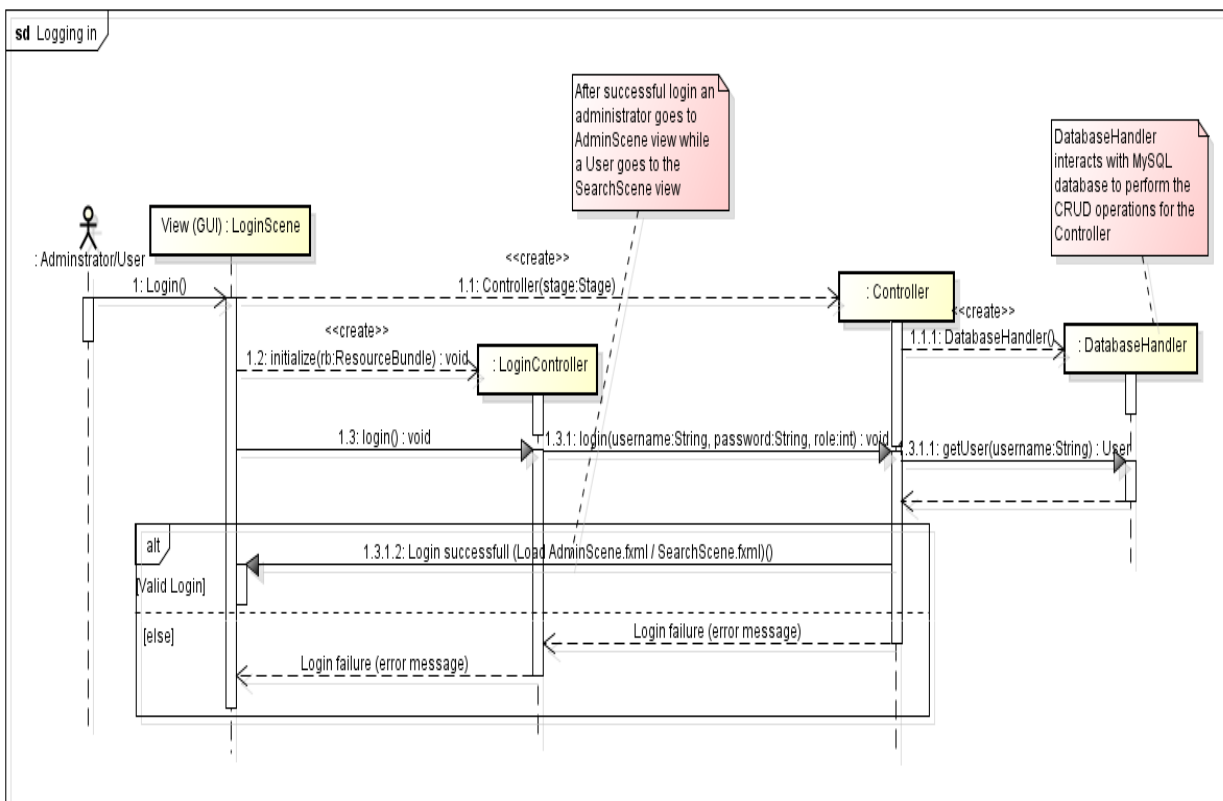


Figure 6.1 UML sequence diagram explaining a user logging in.

Figure 6.2 below shows the login view (GUI) where users can select a role from the dropdown list and enters their password to login. If the log in was successful, the user will have access to either the “Aminstratör” UI or the “Användar” UI based the role of the user.



Figure 6.2 The login UI showing the two possible roles, *Användare* and *Administratör*.

6.1.2 Administrator

The UML Sequence Diagram, figure 6.3, on the next page describes how the main components of the system interact to perform all Administrator system operations and describes how the internal parts of the component interact to achieve the result required.

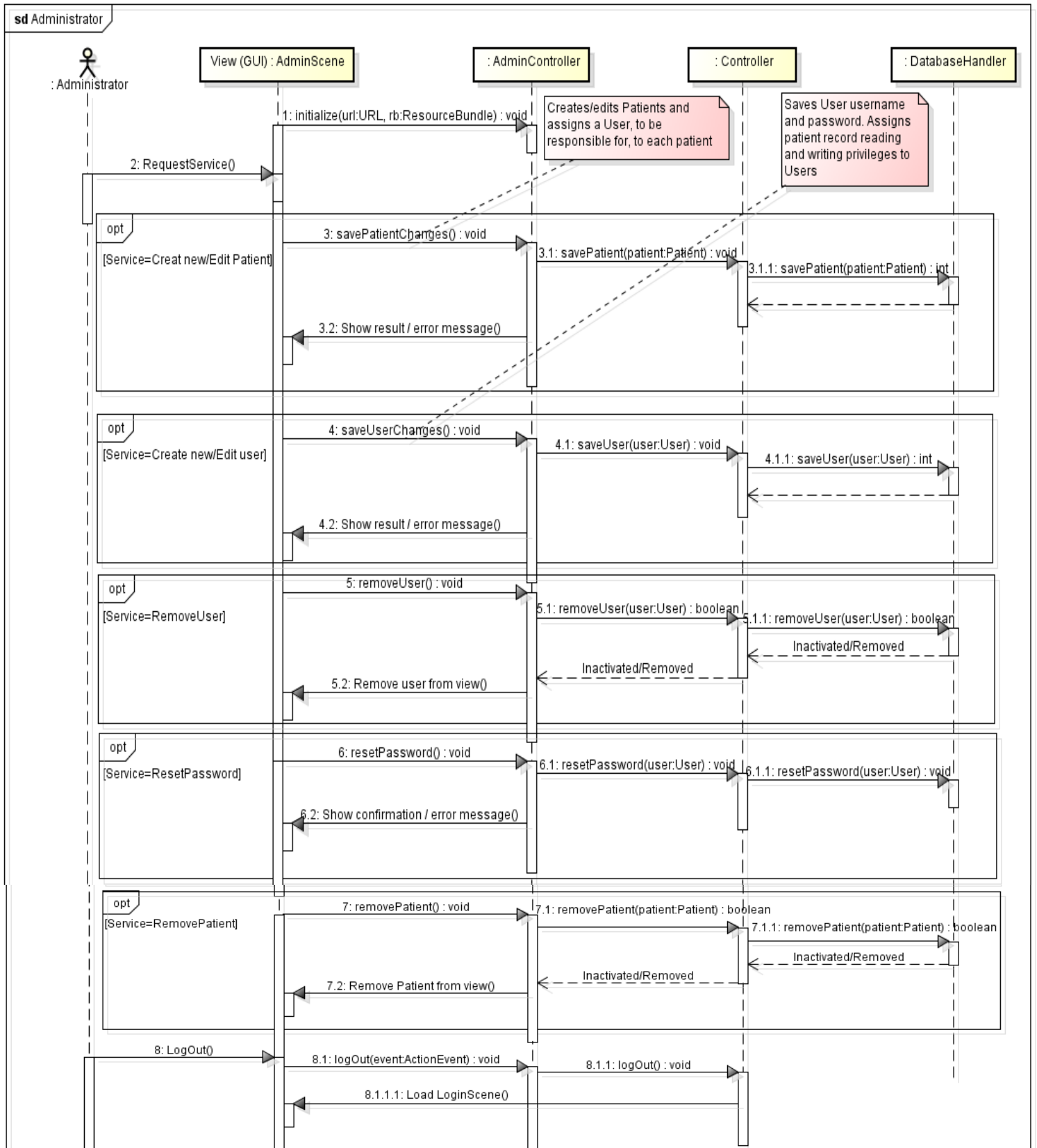


Figure 6.3 UML sequence diagram for all the actions an administrator can perform in the AdminScene.

Figure 6 below shows the administrator UI before any event (pressing a button or choosing from a dropdown menu) is caused by the administrator. The administrator UI allows the administrator to create and/or edit patients and users (see figure 6.5).



Figure 6.4 The administrator UI

The administrator can also give the users different rights of administration and change their password if need be (see figure 6.5).

Figure 6.5 Administrator UI showing user rights check boxes and input text fields for new or an already existing user.

The blue rectangle shows the parts of the UI that appear when the *Ny användare/New User* button (red circle) is pressed and also when a user is chosen from the black-circled drop down menu (*Välj användare/Choose user*) but this time the fields are filled with a hypothetical new user being created. The green circled drop-down menu indicates the title of the new user while the orange circled part is when a user forgets his/her password, it enables the admin to reset the password. The *aktiv användare/active user* checkbox must be checked in order for a user to be able to log in to the system. This action can be used to inactivate a user that for example has stopped working at the company and hence should not have access to the system any more.

The screenshot shows a web application interface for creating a new patient and assigning a user. The interface is in Swedish and features the following elements:

- Top Bar:** "Inloggad som: jemal" and a "Logga ut" button.
- Användar-administrering:** A section for user management with a "Välj användare:" dropdown menu, "Radera användare" button, and "Ny användare" button.
- Patient-administrering:** A section for patient management with a "Välj patient:" dropdown menu, "Radera patient" button, and "Ny patient" button.
- Patient Form:**
 - Personnummer: 19950404-4646
 - Förnamn: Aron
 - Efternamn: Smith
 - Ansvarig person: Sarah (Sarah Johannes) (with a dropdown menu open showing options: jemal (jemal Saleh), sodermar (Niclas Södermark), Sarah (Sarah Johannes))
 - aktiv patient
 - Spara button

Figure 6.6 Shows a new patient being created and a user, which we have created earlier (in Figure 6.5), which is can now be selected in the list of users, is assigned to be the responsible person for this new patient. Setting this patient to active (using the "aktiv patient"-checkbox) enables the patient to be seen by users of the system.

6.1.3 User

The following UML Sequence Diagram describes how the main components of the system interact to perform all User system operations and describes how the internal parts of the component interact to achieve the result required.

Due to lack of space on the next page, we excluded the log out part for the User actions in the sequence diagram (figure 6.7) on the next page, but it works the same as the Administrator's log out (see figure 6.3 above).

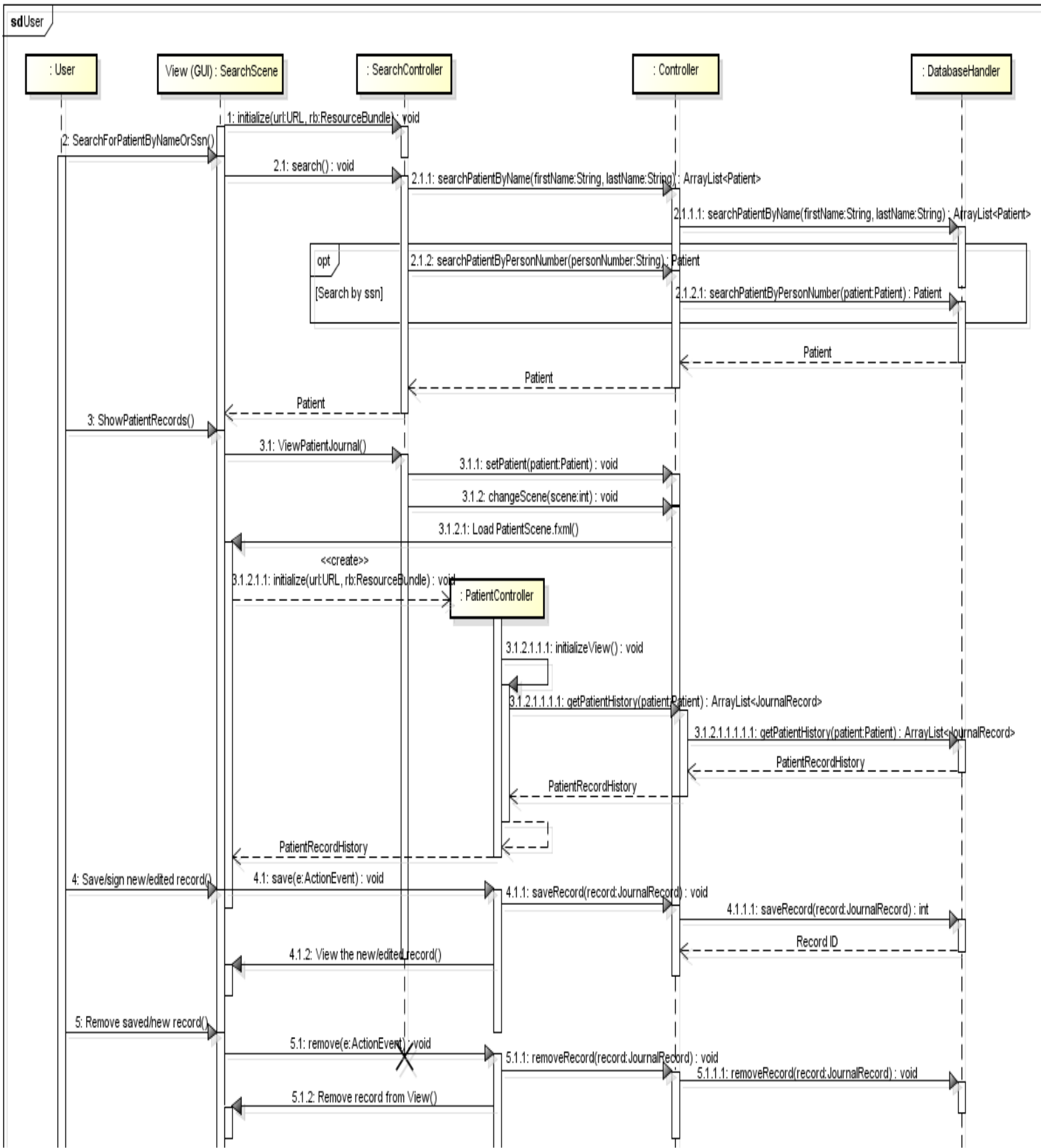


Figure 6.7 UML sequence diagram of all the actions that can be performed as a user.

As a user, one can search for a patient by first name, last name or social security number. Figure 6.8 below shows the view for searching and listing patients and figure 6.9 shows a patient being searched by first name. Choosing a patient in the “Resultat” section in figure 6.9 leads to the view shown in figure 6.10, where the patient’s records are shown and where a new record can be created for the patient. A created (saved) patient record can be signed which means it is stored permanently and cannot be deleted or edited. If it is not signed then it can still be edited and deleted.

Inloggad som: jemal Logga ut

Sök patient:
(Fyll i de fält som du vill söka efter)

Personnummer: (ÅÅÅÅMMDD-XXXX) Förnamn: Efternamn:

Sök

Resultat:

Personnummer	Förnamn	Efternamn
--------------	---------	-----------

Välj

Figure 6.8 The search patient view before any event.

– X

Inloggad som: Sarah

[Logga ut](#)

Sök patient:
 (Fyll i de fält som du vill söka efter)

Personnummer: Förnamn: Efternamn:

(ÅÅÅÅMMDD-XXXX)

[Sök](#)

Resultat:

Personnummer	Förnamn	Efternamn
19950404-4646	Aron	Smith

[Välj](#)

Figure 6.9 The same patient created earlier being searched by a user, also created earlier, using the patient's first name.

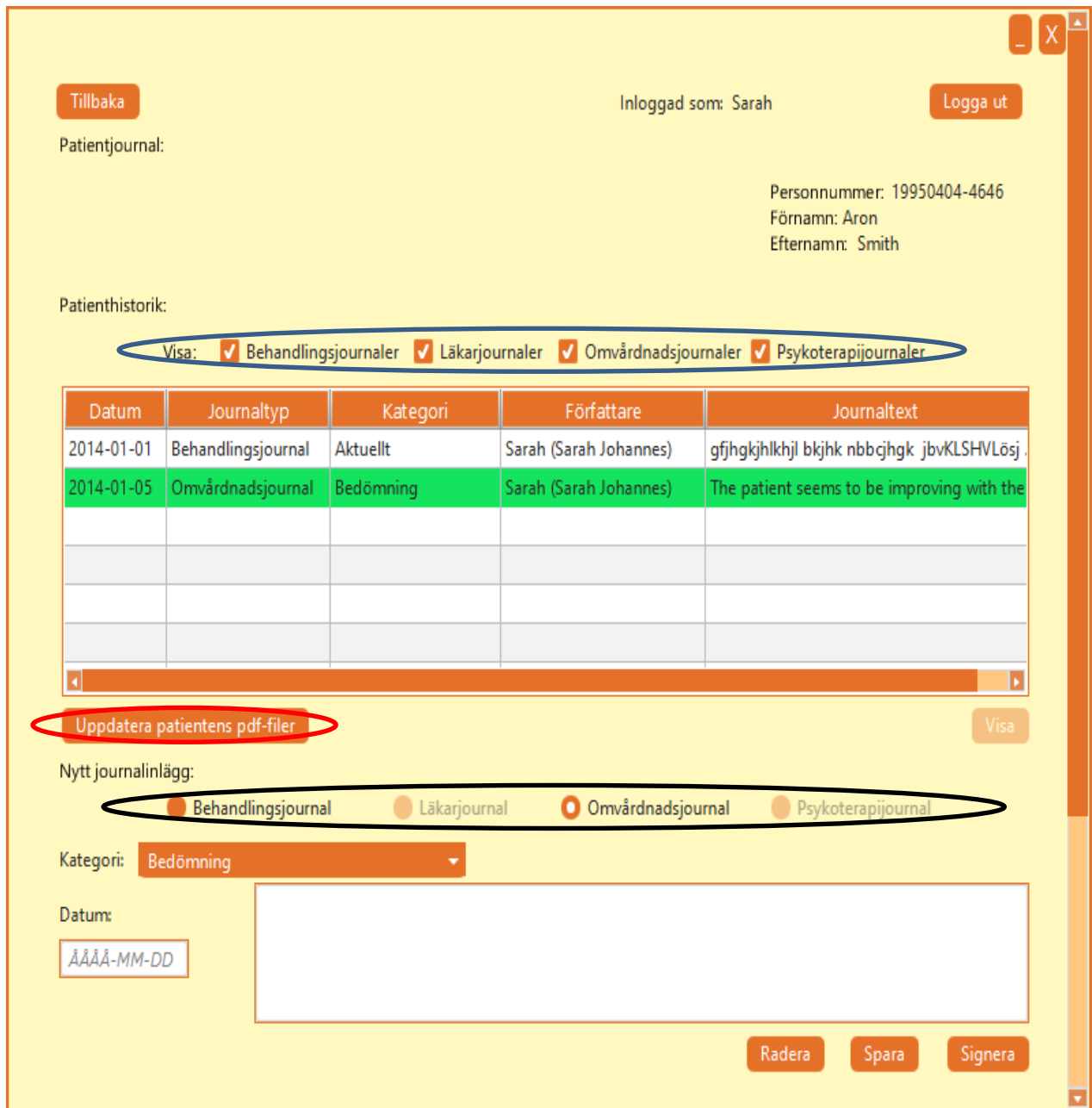


Figure 6.10 The view showing a chosen patient's medical records.

The blue-circled part is used to choose the type of patient record to be shown in the list (this enables a user to sort by type of record) and the black-circled part (RadioButtons) are used to choose the type of patient record to create for this particular patient. These RadioButtons show also what privileges, i.e. the types of records the user can create. Here, for example, the user can create only *Behandlingsjournal/Treatment records* and *Omvårdnadsjournal/Care records*. The green-colored patient records in the table are signed and the gray ones are unsigned. Selecting a green-colored record leads to a new view showing the details of the record while selecting a gray one enables the user to edit, delete or sign the record. The red circled button causes all the records for this particular

patient in the database to be written as PDF in the server's directory structure, see figure D1 in Appendix D for sample patient medical record in PDF format. The text area is where a new patient record is written.

6.2 Logical division

As we have described earlier in chapter 5, the application is based on the MVC pattern and as a result we created the following JavaFX package/file structure (see figure 6.11 on the next page).

The view-package holds all the FXML files representing the different views of the system. Each of these FXML files has a corresponding controller file in the controller-package. These controller files use the Java classes in the model-package to perform system operations. The model-package contains the Java objects that represent patient, record and user. It also contains the *DatabaseHandler.java* class (connects to database and performs CRUD operations) and some helper classes, *DateHandler.java* (checks for correct date format) and *PersonNumber.java* (checks social security number format). The *JournalDocWriter.java* in the model-package is a class used by the *PatientController.java* class in the controller-package to convert the medical records in the database to a file system in the server as PDF files.

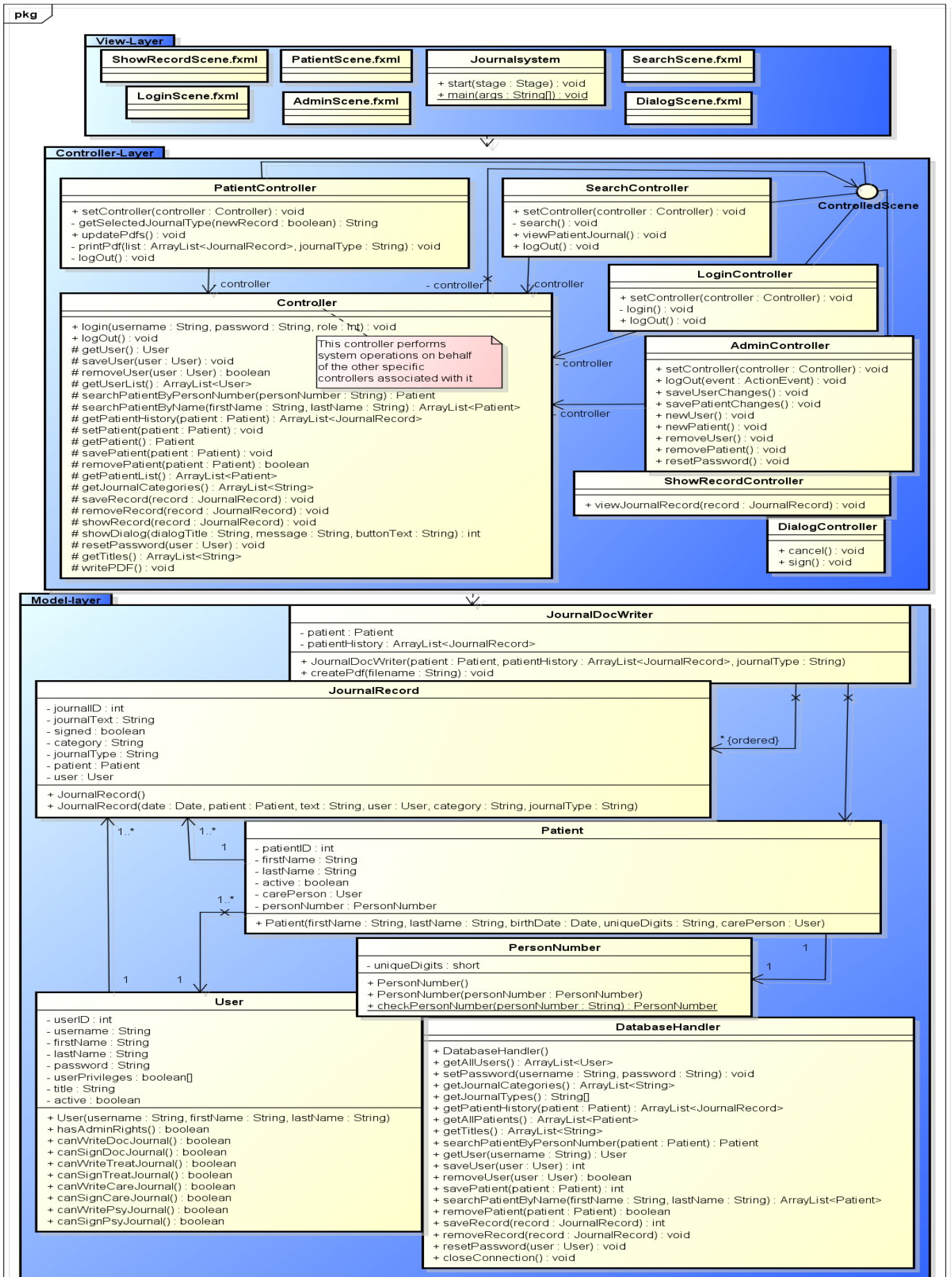


Figure 6.11: The systems logical division

6.3 Database

Since we decided to use MySQL database and Sofiaängen had no previous database set up, we installed and configured an instance of the MySQL community server, which is an open source database. The database was located on their Mac server.

6.4 Documentation

After we developed the system completely to the customer's satisfaction, we prepared System Requirement Specification (SRS) document (see Appendix A), architecture document (see Appendix B), user manual and installation guide document for the developed EMRS. The installation guide is very small and easy so we decided to include it in the user manual.

7 Conclusions

The goal of this degree project was to build an EMRS for a medical clinic (Sofiaängen behandling och skola). The system was developed in accordance with the requirement specifications and it fulfills all the requirements (including the look-and-feel requirement of the GUI) given to us in the form of use cases and descriptions. Of course, as always, some things could have been done better both code- and design-wise if we had more time. However, we are completely satisfied with the system we developed and so are our customers.

We used NetBeans IDE and JavaFX to develop the system. This turned out to be very good choice for us, because eventhough it took us quite some time to learn and to get started with Javafx (JavaFX is relatively new and there aren't many useful tutorials available that could help us in this project), we found JavaFX to be easy to develop applications with.

The iterative work model led to our early development of a product we could test and run. With small incremental changes, we could easily identify bugs in the new versions. We think this model has worked well for this type of project that requires flexibility and adaptation in which many additional requirements came during the implementation phase. The result was successful and we were able to deploy and test the system in the customer's workplace even though we still have to fix the URL issue (mentioned in chapter 5.6) so the system stores the converted PDF files in the specified location rather than in a default location, which we think is a minor problem that can be fixed.

7.1 Validity

We made sure that the system we developed satisfies the specified requirements in the SRS by conducting system tests using test data. System testing was also successfully demonstrated to our customer and our customers were more than happy with the result. Hence we believe that the system meets the customer's requirements.

7.2 Reliability

We can safely say that this system is reliable for a number of reasons. One reason is we have tested all the features of the system on site, in customer's workplace, and found the system to work perfectly without any system failure (except for the issue described in chapter 5.6). Even though we have not conducted *acceptance testing* yet, the customers have been testing and playing with the system for a long time now and have not reported any system failure yet. Second reason is that the system users are very few (less than 10) and MySQL can handle more than that number of concurrent users (threads) very easily [33]. We don't expect any workload related system failures. In case of system failure, the data will not be compromised because it will be securely stored in the MySQL database and will be backed up as PDF separated from the database.

8 Future Development

In this section we will describe some of the areas that have been identified as in need of further work but have been left for future development due to the time limit and priority requirements.

8.1 Hash or encrypt Passwords

For now passwords in the MySQL database are stored as plain texts since the database itself is protected by the security of the server. However, in the future it would be better to use some encryption algorithm to store the passwords.

8.2 Logging for debugging

We have handled all the pre-defined error scenarios (that we could think of) that can occur by sending informative messages to the user. Still the application can be debugged more easily with some logging tool which can provide more detailed information about the errors.

8.3 Support for the handicapped (Vision-impaired)

Right now the system has a fixed font size but in the future it would be more convenient for the user to be able to change the font size.

REFERENCES

- [1] Martin Heller, "REST and CRUD", <http://www.infoworld.com/d/developer-world/>, Retrieved 2013-08-01
- [2] NetBeans, "NetBeans IDE" <https://netbeans.org/>, Retrieved 2013-11-13
- [3] HIMSS, "Selecting the Right EMR Vendor", <http://www.himss.org>
- [4] Christer Bark and Martin Hardenberger, "Landstingen behåller kritiserade journalsystem", Sjukhusläkaren 1 maj, 2012, <http://www.sjukhuslakaren.se>
- [5] CompuGroup Medical CGM, www.compugroupmedical.com, Retrieved 2013-11-17
- [6] Cambio Healthcare Systems, www.cambio.se Retrieved 2013-11-17
- [7] INFOMEDIX, <http://www.infomedix.com.au/>, Retrieved 2013-11-17
- [8] Norrbottens Läns Landsting, NLL, "VAS-Portalen", <https://www.nllplus.se>
- [9] Cecilia Lundberg, "Jämförelse av elektroniska patientjournalsystem", Uppsala University, Disciplinary Domain of Science and Technology, Master thesis no: ISSN: 1650-8319, UPTEC STS 11 015, 2011, 111 pages
- [10] Addison Wesley SLIT, "IT-stöd inom landstingen i Sverige", 2009, page 9
- [11] PSYKBASE, <http://www.psykbase.se/> Retrieved 2013-11-17
- [12] OpenEMR, <http://www.open-emr.org/>, Retrieved 2013-11-18
- [13] VistA, <http://www.ehealth.va.gov/VistA.asp>, Retrieved 2013-11-18
- [14] GNU Health, <http://www.gnuhealth.org/>, Retrieved 2013-11-18
- [15] Open Source Clinical Applications & Resources (OSCAR), <http://oscarcanada.org/>
- [16] Kirch, Wilhelm, Encyclopedia of Public Health. Volume 2, 2008
- [17] Martin Löfberg, Patrik Molin, "Web vs. Standalone Application", Blekinge Institute of Technology, School of Engineering, Master thesis no: MSE-2005:07, 36 pages.
- [18] Oracle, "JavaFX 2 Documentation", <http://docs.oracle.com/javafx/>, Retrieved 2013-08-12
- [19] Oracle, "javax.swing Package Summary", <http://docs.oracle.com/javase/7/docs>, Retrieved 2013-10-25
- [20] iText, <http://itextpdf.com/>, Retrieved 2014-01-19
- [21] Datainspektionen, "Molntjänster" <http://www.datainspektionen.se/>
- [22] Sornmeville Ian, Software Engineering, 9th Edition, New York, Pearson Education, 2011
- [23] Sornmeville Ian, Software Engineering, 8th Edition, London, Pearson Education, 2007
- [24] B. B. Agarwal, S. P. Tayal, Mahesh Gupta, Software Engineering and Testing, Sudbury (Massachusetts), Jones & Bartlett Learning, 2010
- [25] MySQL, <http://www.mysql.com/>, Retrieved 2013-11-25
- [26] Roger S. Pressman, Software Engineering A Practitioner's Approach, New York, McGraw-Hill, 7th Edition, 2010
- [27] Oracle, "JavaFX Frequently Asked Questions", <http://www.oracle.com/>, Retrieved 2013-08-12
- [28] Infoscience, "Deprecating the Observer Pattern", <http://infoscience.epfl.ch/record/176887>

- [29] MySQL, Why MySQL? <http://www.mysql.com/why-mysql/>, Retrieved 2013-09-02
- [30] Kevin Roebuck, Object-Relational Mapping, Newstead: Emereo Pty Limited.,2011
- [31] MartinFowler.com, "[MF Bliki: POJO](#)" , Retrieved 2013-12-12
- [32] Oracle, "Connecting SceneBuilder edited FXML to Java code", https://blogs.oracle.com/connecting_scenebuilder_edited_fxml_to, Retrieved 2013-08-30
- [33] MySQL, Documentation, <http://dev.mysql.com/doc/>, Retrieved 2013-01-17

Appendix A: System Requirement Specification (SRS)

A1. Introduction

A1.1 Purpose

The purpose of this document is to specify the requirements for an Electronic Medical Record System (EMRS). The intended audience are Sofiaängen's staff (<http://sofiaangen.com/>)

A1.2 Scope

The EMRS shall assist Sofiaängen staff to store and manipulate patient medical records quickly and efficiently. The system shall give access only to authorized personnel. The EMRS shall convert medical record to PDF.

A1.3 Definitions, Acronyms, and Abbreviations

EMRS: Electronic Medical Record System

GUI: Graphical User Interface.

REQ: Requirement

A1.4 References

No formal documents have been referenced in this document

A1.5 Overview

This document provides a high-level description of the EMRS. It identifies the involved users and explains their roles. It describes the general constraints and any assumptions and dependencies there are on the system. The document then describes the specific requirements.

2. General Description

A2.1 Product Perspective

The EMRS is self-contained software that stores and manages patient medical records and is intended for use on any platform that supports java.

2.2 Product Functions

The following lists the main system functions:

- **Patient and User registration:** The system shall allow the registration of new users and patients by an administrator.
- **Medical Record creation:** The system shall allow the creation and management of psychotherapy, health (physical), treatment and nursing (care) record and also shall enable the user to assign category to these records.

- **Report generation:** The system shall convert medical records to Microsoft word.
- **Authentication and authorization:** The system gives access only to authorized personnel.

2.3 User Characteristics

The EMRS will be used by Sofiaängen treatment personnel (behandlingspersonal) and since the system is small and has a user-friendly GUI, no formal training is required but some practice is required by the user to be able to use it properly.

2.4 General Constraints

The system must be user-friendly and intuitive.

2.5 Assumptions and Dependencies

It is assumed that the users have adequate skill with using computers and computer software

3. Specific Requirements

3.1 Functional Requirements

REQ 1: Adding new patients – The system shall collect and display essential demographic patient information such as: first name, last name and social security number.

REQ 2: Adding new users – The system shall enable a new user to choose a username and a password.

REQ 3: Privileges – The system shall allow the allocation of different patient record creation privileges to users.

REQ 4: New patient records – The system shall allow a user (based on his/her privileges) to create psychotherapy, treatment, medical and nursing records

REQ 5: Category choice – The system shall enable the user to choose a category from a list for each new record being created.

REQ 6: Storing patient records – The system shall allow the user to save and sign patient records. A saved only record can be edited but a signed record can never be edited again.

REQ 7: Report generation – The system shall convert all patient records to Microsoft Word.

3.2 Non-Functional Requirements

3.2.1 Security

REQ 8: Authentication and authorization – The system shall provide a means to authenticate a user identity using a username and password before enabling the user to perform any functions based on the user's privileges.

REQ 9: Backup – The system shall ensure data protection by setting up a backup system. Backup of EMR data should be automated within the system wherever possible to ensure consistency

REQ 10: Control of data – The system shall provide full control over own data.

3.2.2 Reliability

REQ 11: Availability – The system shall be available at all times. (And there is no need for accessing the system remotely through the Internet)

3.2.3 Economic Constraint

REQ 12: Expense – The system shall not incur recurring expenses

Appendix B: Architecture Document

Architecture Document

Electronic Medical Record System

Version 1.0

B1. Introduction

B1.1 Purpose

This is the Software Architecture Document (SAD) of the Electronic Medical Record System (EMRS) developed for Sofiaängen. It describes the software's architecture, i.e. it gives a comprehensive architectural overview of the EMRS. It is intended to capture and convey the significant architectural decisions which have been made on the system.

B1.2 Scope

The scope of this SAD is to outline the overall design of the system considered to be architecturally significant. It contains information relating to the architectural design of the software, the Structure of the Database, and of the physical server hosting the database.

B1.3 Definitions, Acronyms and Abbreviations

EMRS: Electronic Medical Record System

SAD: Software Architecture Document

UML: Unified Modeling Language

MVC: Model View Controller

MySQL: Relational database management system (RDBMS)

ACID: Atomicity, Consistency, Isolation, Durability

SSN: Social Security Number

FXML: XML-based language for describing user interfaces in JavaFX platform [2]

B1.4 References

[1] Sornmeville Ian, *Software Engineering*, 9th Edition, New York, Pearson Education, 2011

[2] Oracle, "JavaFX 2 Documentation", <http://docs.oracle.com/javafx/>

B2. Architectural Representation

This document details the EMRS architecture using the use case view, logical view and deployment view, using UML diagrams where appropriate.

B3. Architectural Goals and Constraints

B3.1 Technical Platform

The EMRS will be deployed as a platform-independent java jar-file on each Sofiaängen computer and will be connected to a MySQL database installed on Sofiaängen server.

B3.2 Transaction

Transaction will be handled by MySQL's storage engine, InnoDB. MySQL 5.5 and later versions use it by default. It provides the standard ACID-compliant transaction features.

B3.3 Security

Sofiaängen currently have a secure network which allows access to authorized personnel only and this EMR system will reside within the security blanket of this network but even so, the application will also implement some basic security features, such as

- Authentication: Login using at least a user name and a password
- Authorization: according to their privileges, users must be allowed to perform some specific actions.

B3.4 Persistence

Persistence of data will be handled by creating a relational database in MySQL.

B4. Use Case View

This view describes the use case view of the software architecture, that is, it describes the set of scenarios and/or use cases that represent some significant functionality of the system.

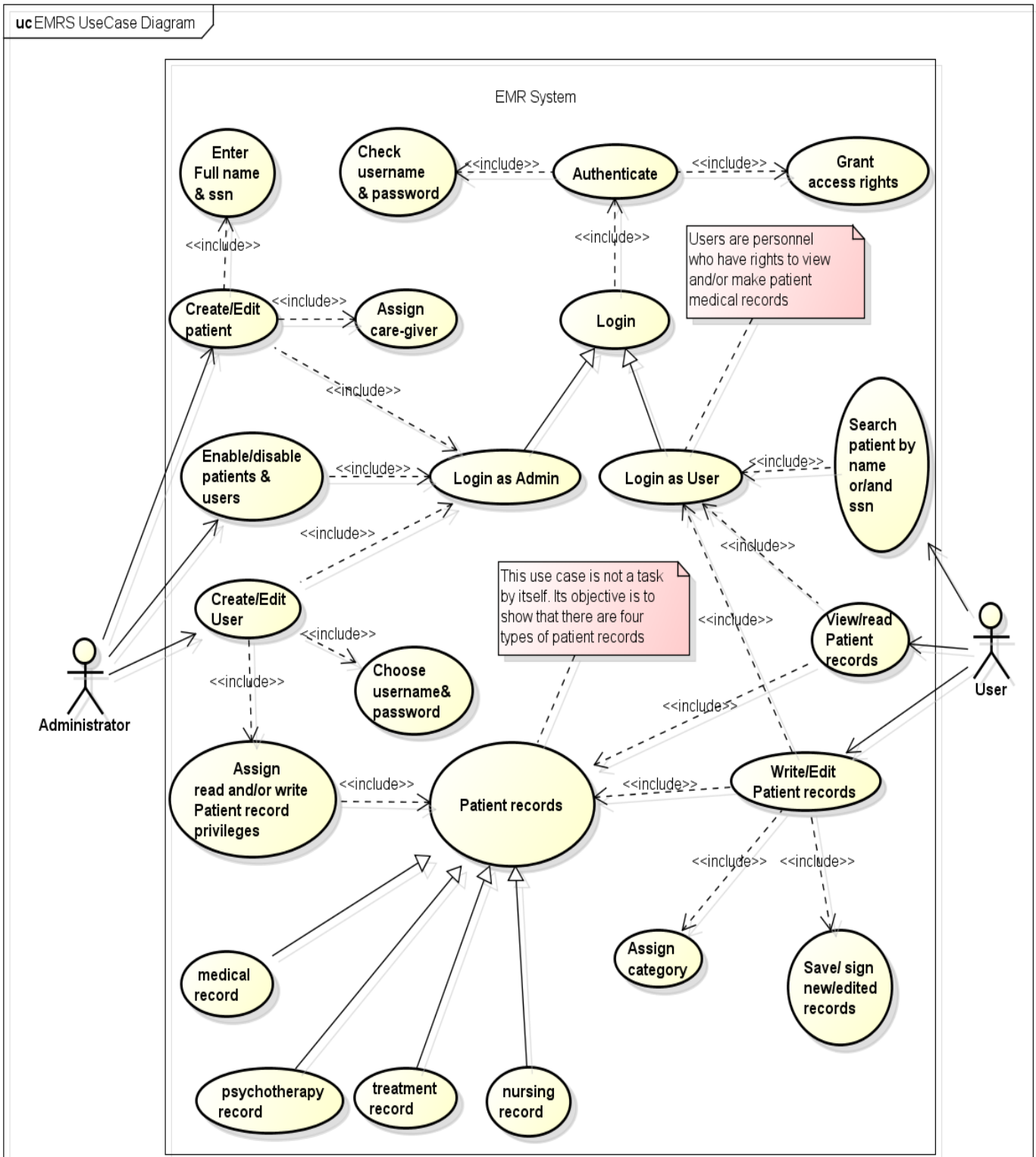
B4.1 List of Use Cases

The use cases in this system are listed below:

1. Login (as admin or as a user)
2. Create/edit patients
3. Enter full name and ssn
4. Assign care-giver
5. Create/edit users
6. Choose username and password
7. Assign, read and/or write Patient record, privileges
8. Enable/disable patients and users
9. Search patients by name and/or ssn
10. View/read patient records
11. Write/edit patient records
12. Assign category
13. Save/sign, create new or edit existing records

B4.2 Use Case Diagram

The following diagram depicts the use cases in the system.



B4.3 Use Case Description

The architecturally significant Use Cases are described below.

1. Login

This Use Case occurs when the administrator or a user wants access to the system. The system requires a username and a password to authenticate the user and give access to the system based on his/her privileges.

2. Create/edit patients

This Use Case occurs when an administrator wants to create a new patient or wants to edit a patient's profile. A patient's full name and social security number is required for a new patient. A care-giver or responsible personnel is assigned to the newly created patient.

3. Create/edit users

This Use Case occurs when an administrator wants to create a new user or wants to edit a user's profile. This requires choosing a username and password by the user being created.

4. Assign read and/or write Patient record privileges

This Use Case occurs when an administrator is creating/ editing a user and wants to give the user different writing and/or reading privileges regarding patient records (medical, psychotherapy, treatment and nursing/care)

5. Search patient by name and/or ssn

This Use Case occurs when a user wants to find or search for a patient using either name/ssn only or using both name and ssn.

6. View/read patient records

This Use Case occurs when a user has already searched for and found a certain patient and wants to see or view this patient's records.

7. Write/edit patient records

This Use Case occurs when a user has already searched and found a certain patient and wants to write new patient records for this patient or when the user wants to edit a saved patient record. For each new patient record created, a category must be assigned from a list of categories.

8. Save/sign, new/edited, records

This Use Case occurs when a user has already searched and found a certain patient and has written a new record or edited a saved record for this patient and wants to save or sign the new/edited patient record. A saved patient record can be edited while a signed record can never be edited.

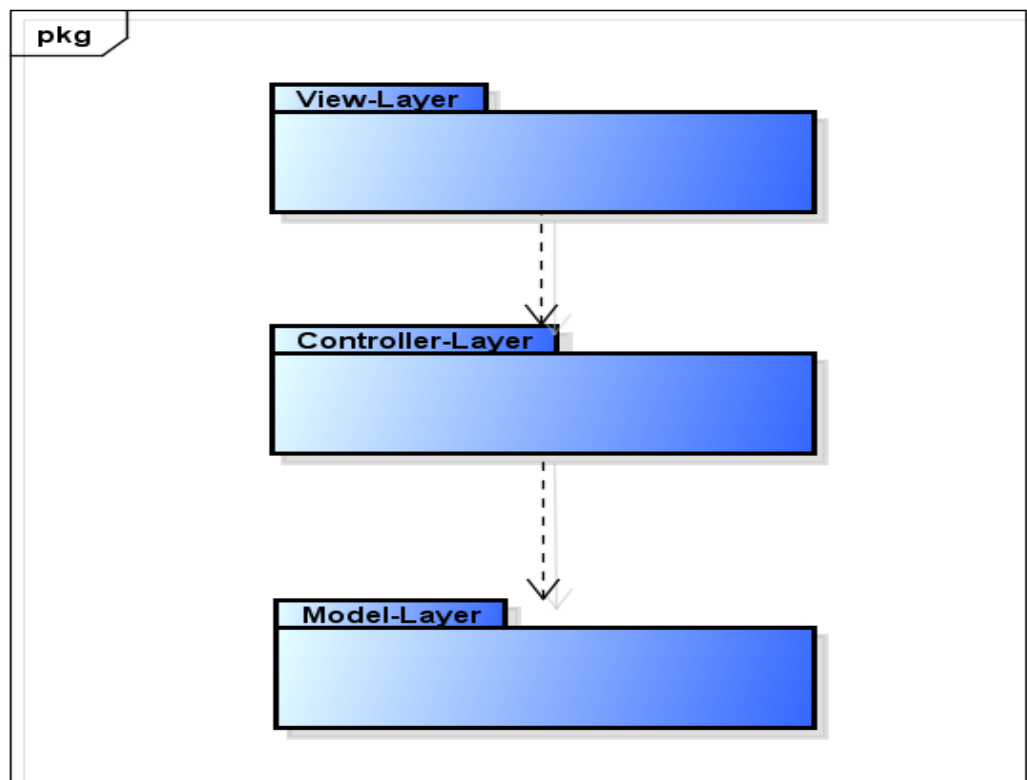
B5. Logical View

This view shows and describes the composition of the most important classes in packages and subsystems, and the organization of these subsystems into layers. The logical view of the EMRS is made up of the 3 main Layers: View layer, Controller layer, and Model layer based on MVC architecture pattern [1]. The MVC pattern separates elements of the system, allowing them to change independently. For example, adding a new view or changing an existing view can be done without any changes to the underlying data in the model. The layers are implemented as packages.

Class diagrams are also included to illustrate the relationships between architecturally significant classes and packages (layers).

B5.1 Architecture Overview – Package Layering

The diagram below shows the system (functionality) organized into separate layers, and each layer only relies on the facilities and services offered by the layer immediately beneath it.



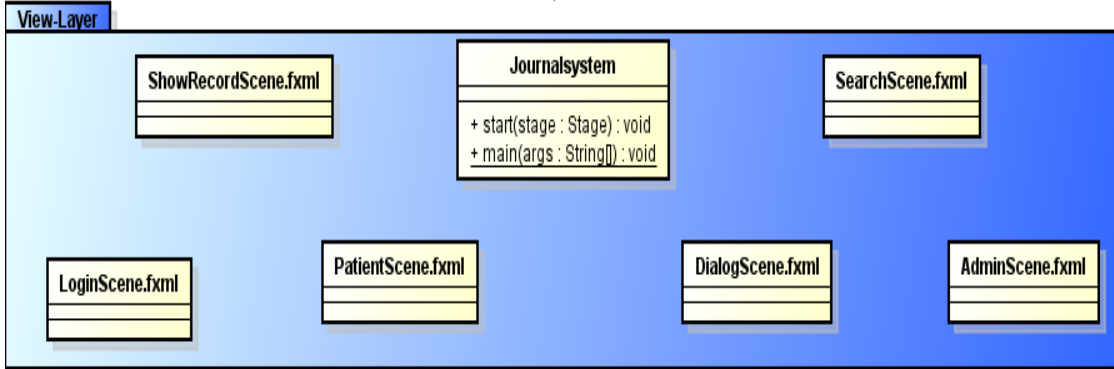
powered by Astah

B5.2 Architecturally Significant Design Packages (Layers)

B5.2.1 View (View Layer)

This layer is responsible for receiving mouse and keyboard input data and/or commands from end users to the system and presenting output results from the system to the end user.

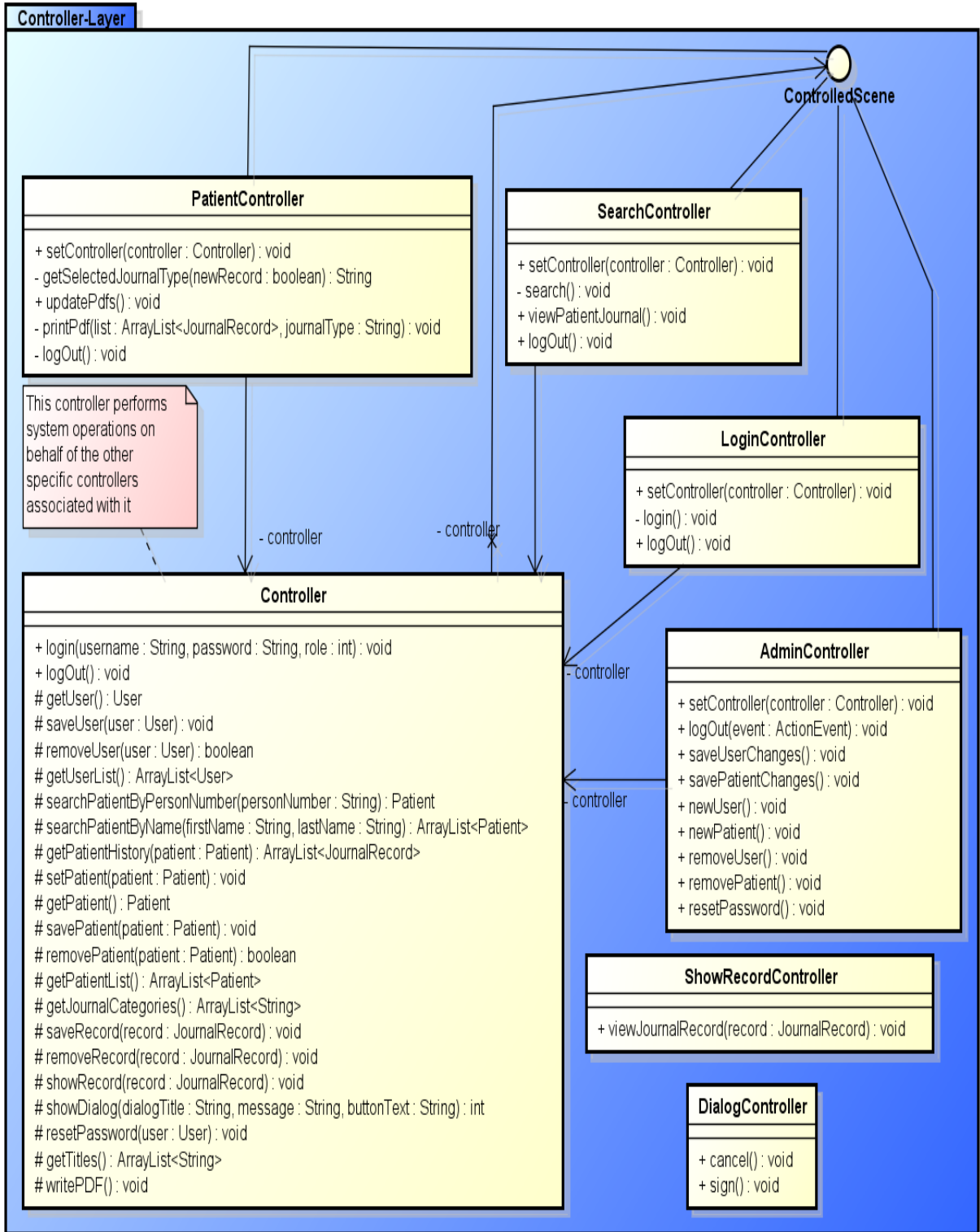
This layer contains 6 fxml files. These files define the user interfaces of the system. It also contains one java class with the main method hence it is responsible for starting the application.



B5.2.2 Controller (Controller Layer)

The controller manages access to the model and the view and coordinates the flow of data between them. The view is updated by the controller with data from the model based on user input.

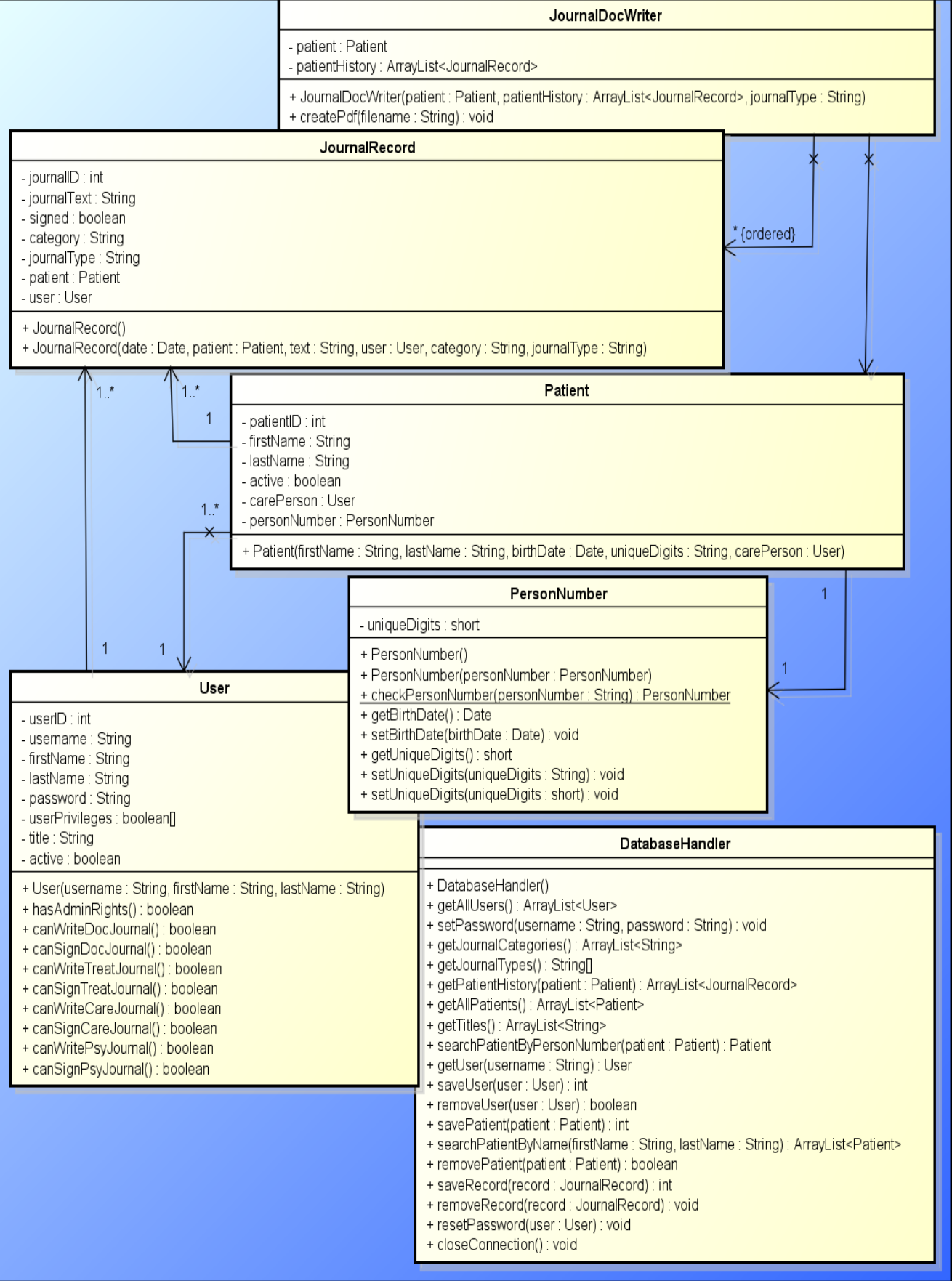
The following diagram shows the class diagram of the classes in this layer and their associations. The classes show only the important methods and fields.



B5.2.3 Model (Model Layer)

The model manages the behavior and data of the application, responds to requests for information about its state from the view, and responds to instructions to change state from the controller. It also accesses the database and the server's file system directly to store and retrieve information. The diagram below shows the important classes in the model layer with their associations

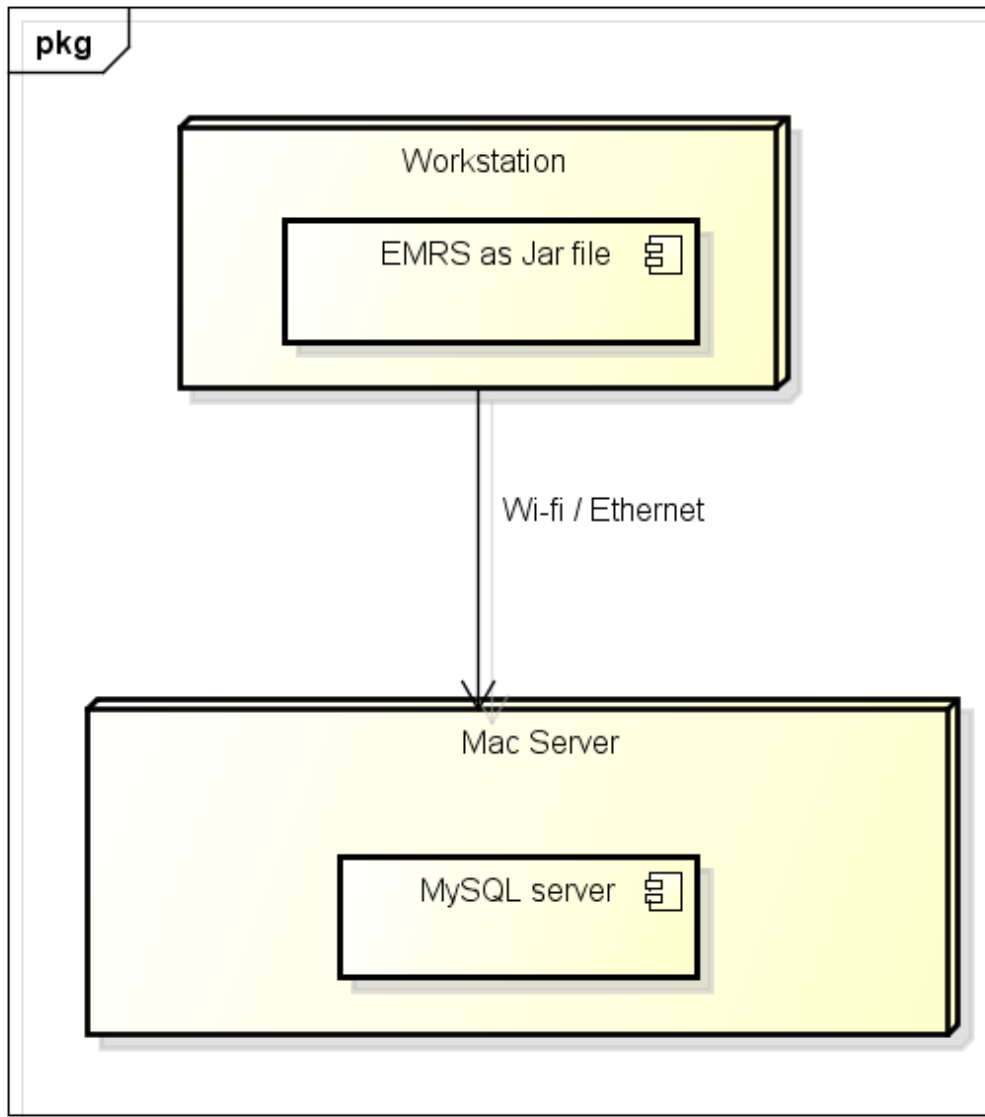
Model-layer



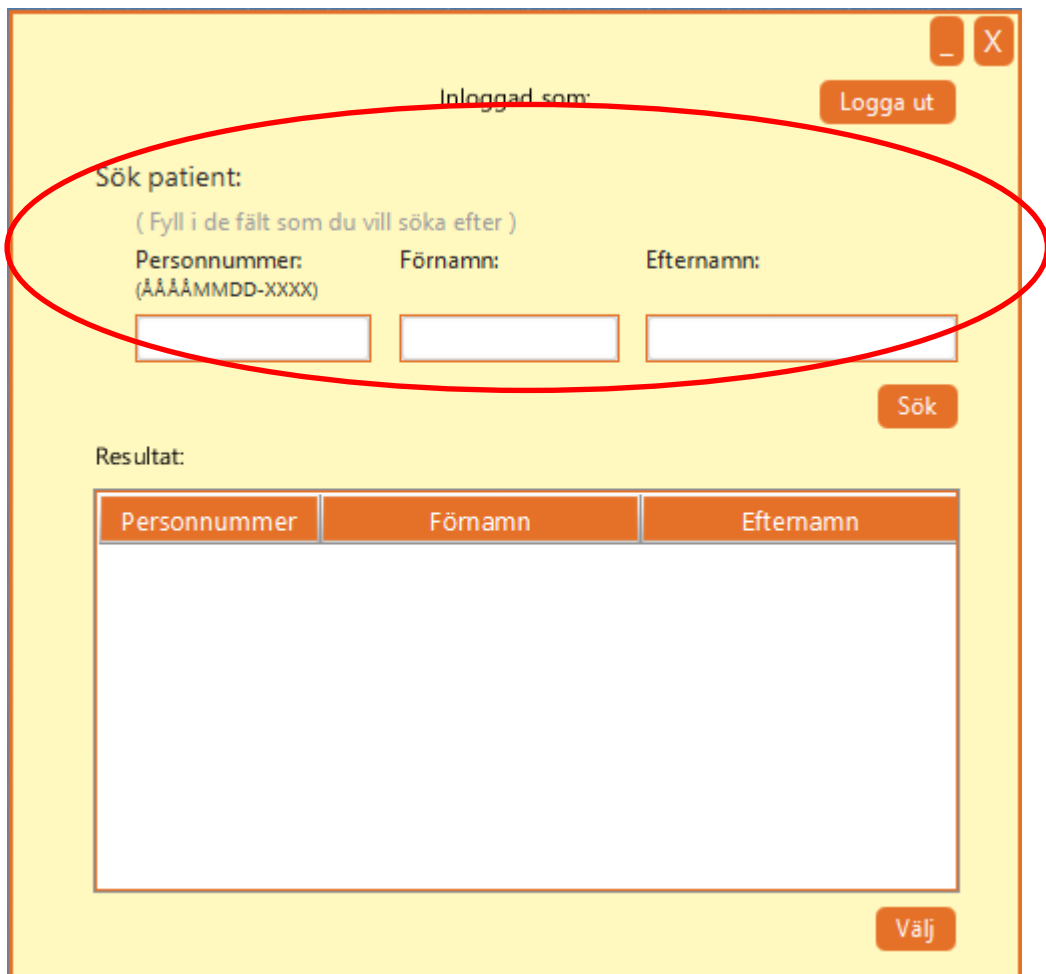
B5.3 Deployment View

The EMR system is deployed as a jar file on every workstation while the MySQL database is installed on a Mac server.

The UML deployment diagram below shows the deployment of the EMRS on the workstations and on the Mac server.



Appendix C: Sample Source Code



The screenshot shows a web interface for searching patients. At the top right, there are window control buttons (minimize, maximize, close) and a 'Logga ut' button. Below this, the text 'Inloggad som:' is followed by a 'Logga ut' button. The main section is titled 'Sök patient:' and includes a sub-instruction '(Fyll i de fält som du vill söka efter)'. There are three input fields: 'Personnummer: (ÅÅÅÅMMDD-XXXX)', 'Förnamn:', and 'Efternamn:'. These three fields are circled in red. To the right of the input fields is a 'Sök' button. Below the search section is a 'Resultat:' section with a table structure. The table has three columns: 'Personnummer', 'Förnamn', and 'Efternamn'. The table body is currently empty. At the bottom right of the table area is a 'Välj' button.

Figure C1 The search scene/view. Here the different search fields are circled.

```

<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.*?>
<?import javafx.scene.text.*?>

<AnchorPane id="AnchorPane" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="484.0"
    prefWidth="505.9998779296875" xmlns:fx="http://javafx.com/fxml" fx:controller="journalssystem.controller.SearchController">
    <children>
        <TextField fx:id="perNumSearchField" layoutX="63.0" layoutY="152.0" onKeyPressed="#keyPressed" prefWidth="118.0" />
        <TextField fx:id="fNameSearchField" layoutX="195.0" layoutY="152.0" onKeyPressed="#keyPressed" prefWidth="110.0" />
        <TextField fx:id="lNameSearchField" layoutX="318.0" layoutY="152.0" onKeyPressed="#keyPressed" prefWidth="156.0" />
        <Button layoutX="434.0" layoutY="187.0" mnemonicParsing="false" onAction="#search" text="Sök" />
        .
        .
        .
        MISSING CODE HERE
        .
        .
        .
    </children>
    <stylesheets>
        <URL value="@patient.css" />
    </stylesheets>
</AnchorPane>

```

Figure C2 The FXML file, SearchScene.fxml, which defines the user interface in figure C1 (the red-circled part)

```

public class SearchController implements Initializable, ControlledScene
{
    @FXML
    private TextField perNumSearchField, fNameSearchField, lNameSearchField;
    @FXML
    private Button chooseButton;
    .
    .
    .
    MISSING CODE HERE
    .
    .
    .
    @FXML
    private void search()
    {
        String perNumInput = perNumSearchField.getText();
        String lNameInput = lNameSearchField.getText();
        String fNameInput = fNameSearchField.getText();

        ArrayList<Patient> searchResults = new ArrayList();
        try
        {
            if (!perNumInput.isEmpty())
            {
                searchResults.add(controller.searchPatientByPersonNumber(perNumInput));
            }
            else if (!lNameInput.isEmpty() || !fNameInput.isEmpty())
            {
                searchResults = controller.searchPatientByName(fNameInput, lNameInput);
            }
        }
        catch (Exception e)
        {
            errorLabel.setText(e.getMessage());
            errorLabel.setTextFill(Color.RED);
        }
        fillSearchTable(searchResults);
    }
    .
    .
    .
    MISSING CODE HERE
    .
    .
    .
}

```

Figure C3 shows the controller file (SearchController.java) of the FXML file, SearchScene.fxml, shown in Figure 11

```

public class DatabaseHandler
{
    .
    .
    .
    MISSING CODE HERE
    .
    .
    .
    public Patient searchPatientByPersonNumber(Patient patient) throws Exception
    {
        String query = "SELECT birthdate, unique_digits, patient.first_name, "
            + "patient.last_name, username, user.first_name, user.last_name "
            + "FROM patient "
            + "JOIN user ON user_id = care_person "
            + "WHERE birthdate = ? "
            + "AND unique_digits = ?";
        PreparedStatement stmt = null;

        Patient resultPatient = new Patient();
        try
        {
            stmt = dbCon.prepareStatement(query);
            stmt.setDate(1, patient.getBirthDate());
            stmt.setShort(2, patient.getUniqueDigits());
            ResultSet result = stmt.executeQuery();
            result.next();
            resultPatient.setBirthDate(result.getDate("birthdate"));
            resultPatient.setUniqueDigits(result.getShort("unique_digits"));
            resultPatient.setFirstName(result.getString("patient.first_name"));
            resultPatient.setLastName(result.getString("patient.last_name"));
            User carePerson = new User(result.getString("username"));
            carePerson.setFirstName(result.getString("user.first_name"));
            carePerson.setLastName(result.getString("user.last_name"));
            resultPatient.setCarePerson(carePerson);
        }
        catch (Exception e)
        {
            stmt.close();
            throw new Exception("Ingen kontakt med databasen");
        }
        stmt.close();
        return resultPatient;
    }
    .
    .
    .
    MISSING CODE HERE
    .
    .
    .
}

```

Figure C4 The database handler file of the system showing only the method that gets a patient by social security number.

Appendix D: Sample generated MR PDF

JOURNALANTECKNINGAR 1
Sofiaängen
BEHANDLING ~ SKOLA
Sofiaängen AB
Bergsunds Strand 43 A
08-428 86 50
Johanes John
19810910-1999

2013-01-01 Bakgrund A new patient has come to our center and this his first entry
jemal Saleh
Konsultläkare

2013-02-02 Beslut the patient seems to adapt quickly to his new school very well.
jemal Saleh
Konsultläkare

2013-03-03 **Daganteckning** Testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing 2013
jemal Saleh
Konsultläkare

2013-05-09 **Aktuellt** testing testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing
testing testing testing testing testing testing testing testing testing testing
testing testing testing testing testing
jemal Saleh
Konsultläkare

Figure D1 A sample patient's medical records in PDF format. The red circled part is the header that appears on every new page.

Appendix E: Survey

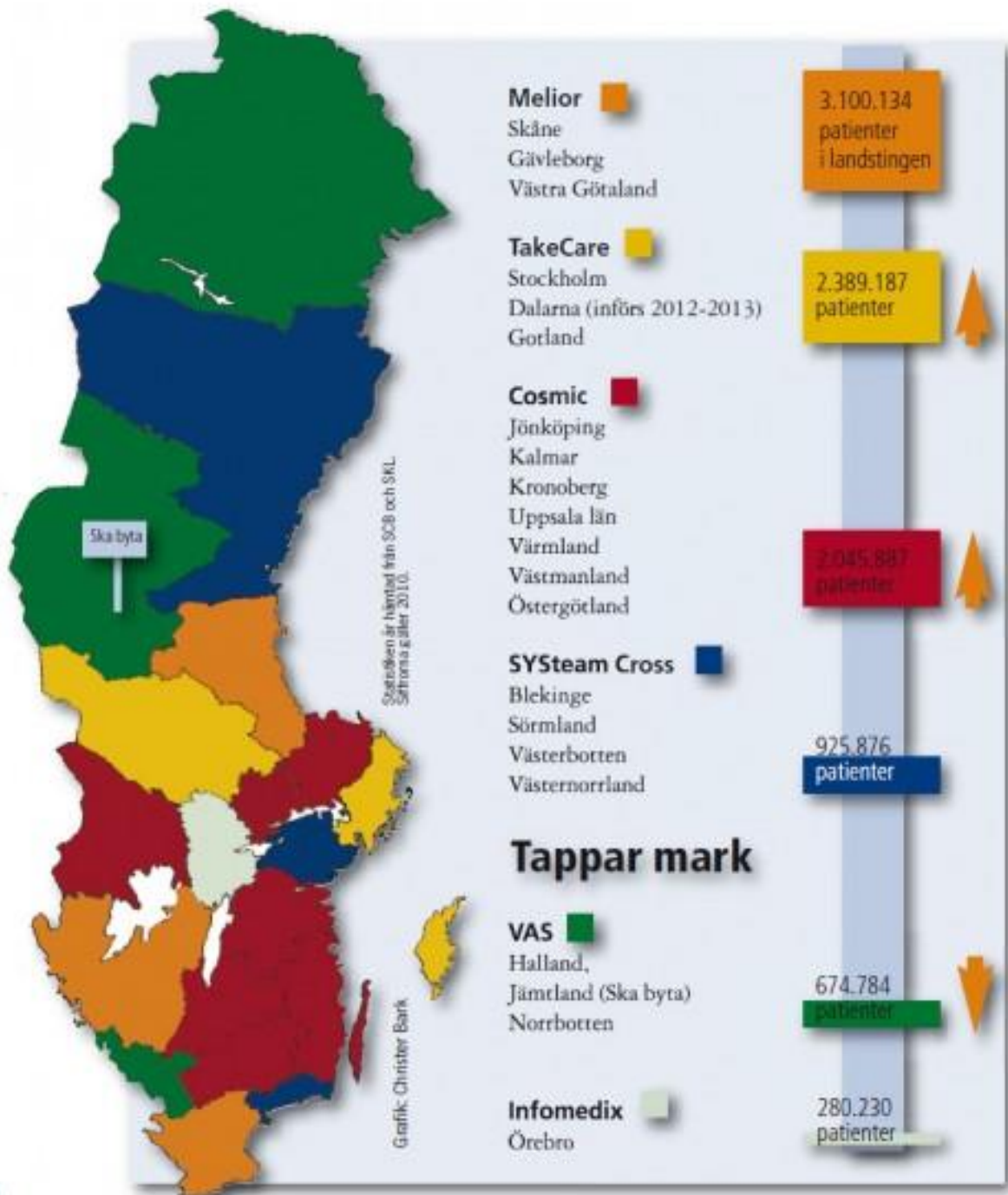


Figure E1 Regions that have chosen the same system for all healthcare documentation in the county.