VIA University College, ICT S17Y

SEP2

Project Report

Hand in 02/06/2017

Group 3

Supervisors

**Table of contents**

[List of figures. 4](#_Toc484100705)

[Abstract 5](#_Toc484100706)

[Introduction 6](#_Toc484100707)

[Evidence Manager System 7](#_Toc484100708)

[Analysis 7](#_Toc484100709)

[UseCase Diagram and Descriptions 7](#_Toc484100710)

[Connection Model 8](#_Toc484100712)

[Design 9](#_Toc484100713)

[Model 9](#_Toc484100714)

[Client-Server (RMI) 11](#_Toc484100715)

[Database 12](#_Toc484100716)

[GUI 13](#_Toc484100717)

[Implementation 16](#_Toc484100718)

[Model 16](#_Toc484100719)

[GUI 18](#_Toc484100720)

[Connection Client-Server 20](#_Toc484100721)

[Database 23](#_Toc484100722)

[Testing 25](#_Toc484100723)

[Test case #1 25](#_Toc484100724)

[GUI 26](#_Toc484100725)

[Results 27](#_Toc484100726)

[Discussion 28](#_Toc484100727)

[Conclusion 29](#_Toc484100728)

[Perspective: 29](#_Toc484100729)

[References and expected sources: 30](#_Toc484100730)

[Appendices 31](#_Toc484100731)

# List of figures.

[Picture 1 UseCase diagram 5](#_Toc484036668)

[Picture 2 Search/view UseCase description 6](#_Toc484036669)

[Picture 3 Analysis of RMI 7](#_Toc484036670)

[Picture 4 MVC design pattern 8](#_Toc484036671)

[Picture 5 Design of database adapter 8](#_Toc484036672)

[Picture 6 Design of client-server architecture 9](#_Toc484036673)

[Picture 7 Design of database 10](file:///C:\Users\Tomas\Desktop\GIT%20GUD\SEP2-GitHub\Project\Project%20Report.docx#_Toc484036674)

[Picture 8 Design of mediator package 10](#_Toc484036675)

[Picture 9 Design of Login scene 11](#_Toc484036676)

[Picture 10 Design of combobox 12](file:///C:\Users\Tomas\Desktop\GIT%20GUD\SEP2-GitHub\Project\Project%20Report.docx#_Toc484036677)

[Picture 11 Design of User view 12](#_Toc484036678)

[Picture 12 Design of Admin view 12](#_Toc484036679)

[Picture 13 Design of add scene 13](#_Toc484036680)

[Picture 14 Implementation of Item constructor 14](#_Toc484036681)

[Picture 15 Implementation of ItemList 14](#_Toc484036682)

[Picture 16 Button controller 15](#_Toc484036683)

[Picture 17 Implementation of button controller 15](#_Toc484036684)

[Picture 18 Implementation of ModelManager 15](#_Toc484036685)

[Picture 19 Implementation of buttons using JavaFX FXML 16](#_Toc484036686)

[Picture 20 Controller package 16](#_Toc484036687)

[Picture 21 Controller overview 16](#_Toc484036688)

[Picture 22 Implementation of controller 17](#_Toc484036689)

[Picture 23 Implementation of ServerInterface 17](#_Toc484036690)

[Picture 24 Server overview 18](#_Toc484036691)

[Picture 25 Making a registry binding. 19](#_Toc484036692)

[Picture 26 Client overview 19](#_Toc484036693)

[Picture 27 Connection to server 20](#_Toc484036694)

[Picture 28 Implementation of database adapter 20](#_Toc484036695)

[Picture 29 Implementation of database adapter 21](file:///C:\Users\Tomas\Desktop\GIT%20GUD\SEP2-GitHub\Project\Project%20Report.docx#_Toc484036696)

# Abstract

*This project aims to show a possible solution regarding proper evidence management and tracking. The reason for its creation is to make an effective system that allows the proper management and tracking of evidence in police stations. The system can differentiate between a user and an administrator. Both can view all the evidence data collected or search to find requested information. Only an administrator has access to more features: adding, removing, and updating evidence data. The system takes advantage of the client-server architecture using Java’s RMI, as well as a PostgreSQL database containing all the data. The achieved result is a system containing the core features that come with evidence management, while providing a layout to something that can turn into a much larger project. Overall, it will allow the proper storing for critical evidence with the opportunity for further expansion.*

# Introduction

Evidence Management has always been an important task for police stations. As more evidence is required to be stored, a need to keep track of it and manage it has appeared. The management of any kind of evidence has proven to be a very important, but difficult task. Given the variety of different evidence items (Evidence handling, 2008), properly storing them is vital to the success of the police forces.

While in most developed countries crime has gone down (Crime and Criminal Justice Statistics, 2016), that doesn’t mean the need of storing evidence has decreased, as it is often needed to keep in store evidence of crimes committed in the past (Laudan & Allen, 2013).

Evidence management is critical to the outcome of criminal prosecutions. The ability to obtain DNA evidence from many items has also resulted in a large increase in the amount of evidence being stored into evidence rooms. Losing meaningful evidence (Zuckerman, 2011) is also a problem that police forces often must deal with. The reason it even occurs is because of disorganized evidence storing can be. This very often leads to legal cases falling apart.

An efficient way to handle the galore of evidence is to keep it stored in a very organized fashion (Property Room Standards, 2008). But even so, without a way to keep track of all different objects, managing them properly can become hectic. Keeping track of evidence data on paper has become obsolete and rather inefficient given the large quantities of evidence.

Storing data of each individual evidence item inside of a system, can, however, makes keeping track of evidence more organized and thus reduce the number of mistakes when it comes to storage within the police departments. Important part of safe evidence management is keeping track of the chain of custody (Evidence, Chain of Custody, 2017) which means having a secure custodianship and handover of responsibility, from the time and place of collection to the time and place of presentation as elements of a proof

Nowadays, there is a large need for good evidence management systems for police stations. Keeping track of a databases with evidence, while providing meaningful information about each item over its entire individual lifetime. Digitizing physical evidence (Stebick, Divonna, & Jonelle Pool, 2006) can reduce the need of handling it until it is time to present it, which reduces the likelihood of it being tampered.

Using databases can also provide a lot of flexibility when working with evidence, because it allows multiple people to access and alter it. The data collected from all the stored evidence can be used for educational and research purposes, such as statistics for the type of evidence found on different crime scenes.

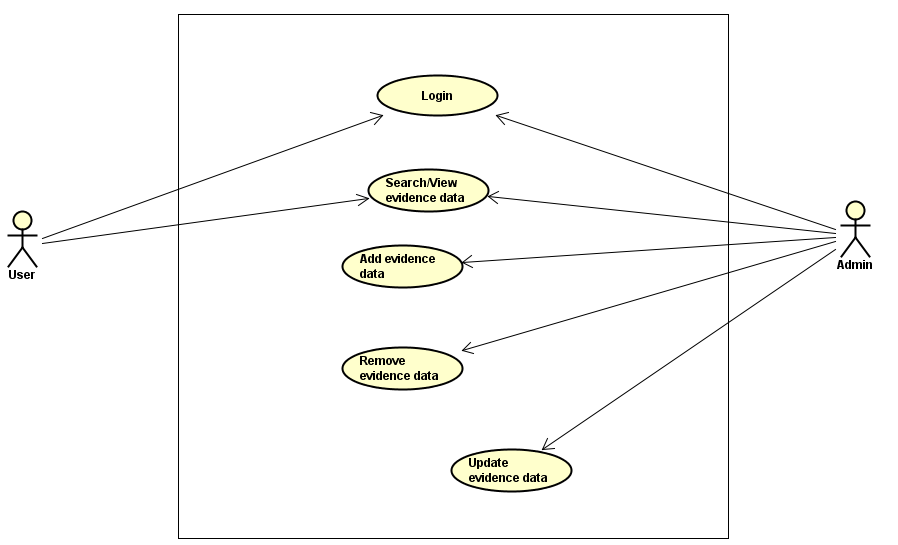
The purpose is to create a system that keeps track of evidence data from various criminal cases.

# Evidence Manager System

# Analysis

## UseCase Diagram and Descriptions

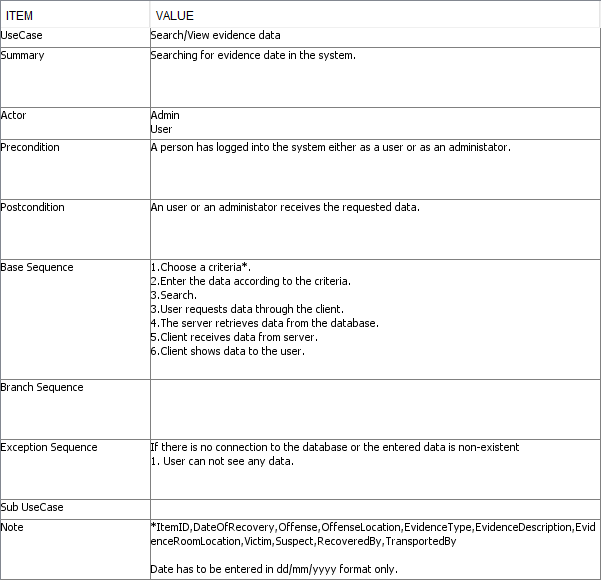
Picture 1 shows the UseCase Diagram for the system. This diagram shows what Evidence Manager system is capable of and what privileges does each actor have.



Picture 1 UseCase diagram

The Administator has access to 3 additional UseCases(Add/Remove/Update evidence data).

Picture 2 expands Search/View evidence date UseCase shown in Picture 1.



Picture 2 Search/view UseCase description

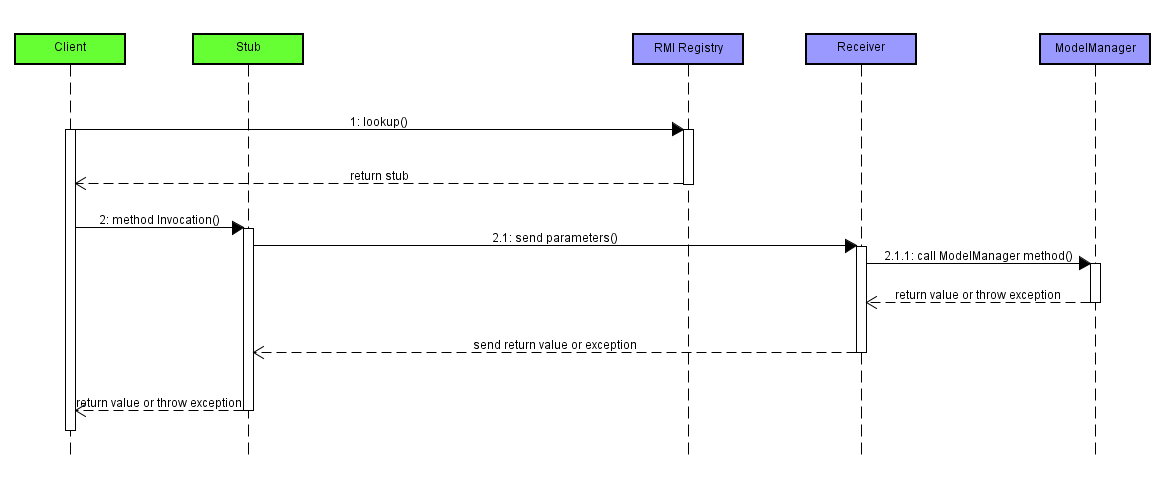
For full list of UseCase descriptions, refer to Appendix A.

## Connection Model

The client communicates with the server using Java RMI API. This enables connection between client and server on the same network.

Picture 3 shows how the client establishes a connection and communicates with the server (Step 1 and 2)

After the connection has been established between the Client and the Server (the client has received a stub) all the communication is done using the stub and calling methods from it(Step 2)



Picture 3 Analysis of RMI

# Design

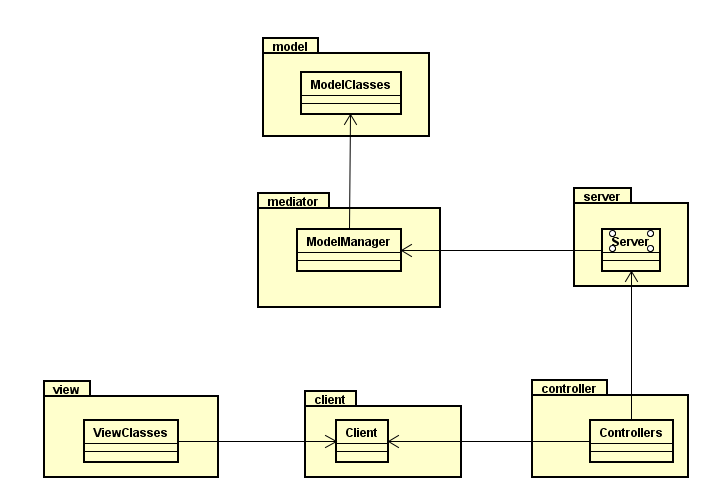
## Model

The system is based on a Model-View-Controller (MVC) design pattern, consisting of the following parts:

The model, which consists of the model package; taking care of storing and processing data. The view classes take input from user and sends information to controllers which pass requests to the server.

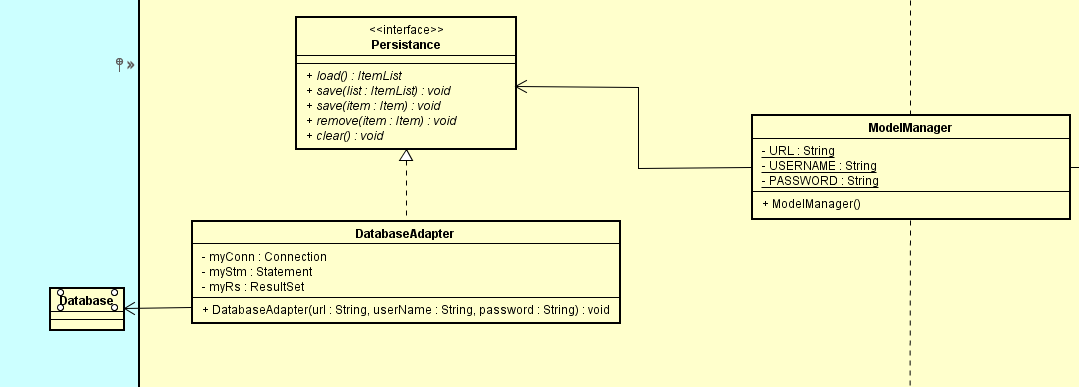
Server sends data to ModelManager which updates the model. It acts as connection between the view and the model.

(Picture 4).



Picture 4 MVC design pattern

One of the most important parts of mediator package is a DatabaseAdapter (Picture 5) Database adapter communicates with ModelManager using SQL statements. This feature is the key element in this system.



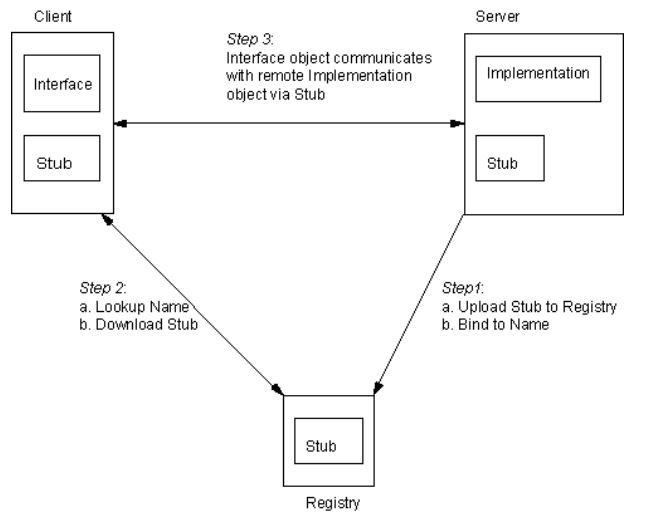
Picture 5 Design of database adapter

For full class diagram, refer to Appendix B.

## Client-Server (RMI)

The client/server architecture was created using Remote Method Invocation.

The figure (Picture 6) below shows how a client first establishes a connection with the server and then communicates. All the communication is done through the stub and the RMI registry as shown in the diagram below.



Picture 6 Design of client-server architecture

The client is the computer that call the remote method and the server is the host computer that processes the call.

When a client invokes a method on the remote object it actually calls an ordinary method on a proxy object called a stub.

The stub sends the information to the server where a receiver object checks the information passed, and calls the desired method passing the supplied parameters. The receiver captures the data from the desired method and sends it back to the stub on the client computer

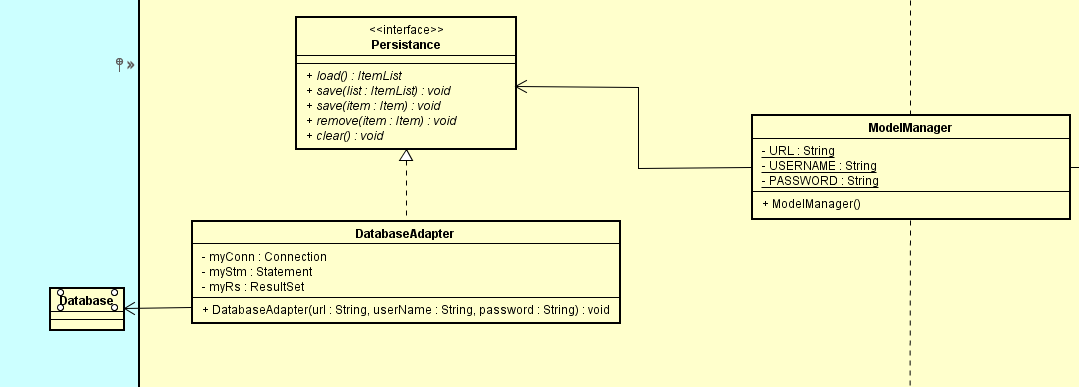
## Database

The database was developed in the way to coincide with the design of our application including the model and the GUI, but also most important following the client's requirements. Because, we were developing a system for a police department we created the fields that could describe a crime in as many details as possible, the same thing needed to be implemented when designing a database. Figure below (Picture 7) expands database design.

Picture 7 Design of database

The reason we have so many fields when creating an item and storing it in the database is because of how important detailed information is in crime reports. The system was made in a way that will not allow the user to input null values, this means that each field must be mandatory written to have a better understanding of crime scene. The reason we don't have more tables is because normalization is mostly used to to avoid data duplication, in our scenario when an item is created each field needs to be filled this means that it pointless to separate the table in more than one and having a complex relationship between them.

This is the mediator package (Picture 8), which is responsible to connect the model, database, RMI and GUI all together. The interface called Persistence is responsible to load all items saved in database and import them in temporary array located in the Model Manager, using this logic the Model Manager can manage all data stored in the database.

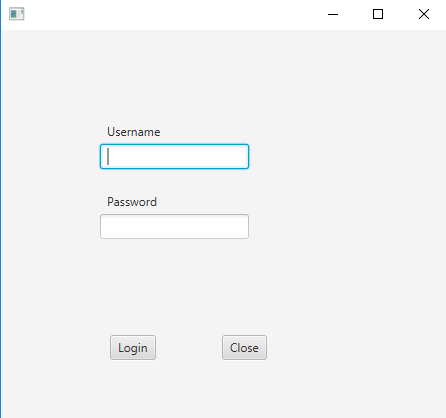


Picture 8 Design of mediator package

GUI

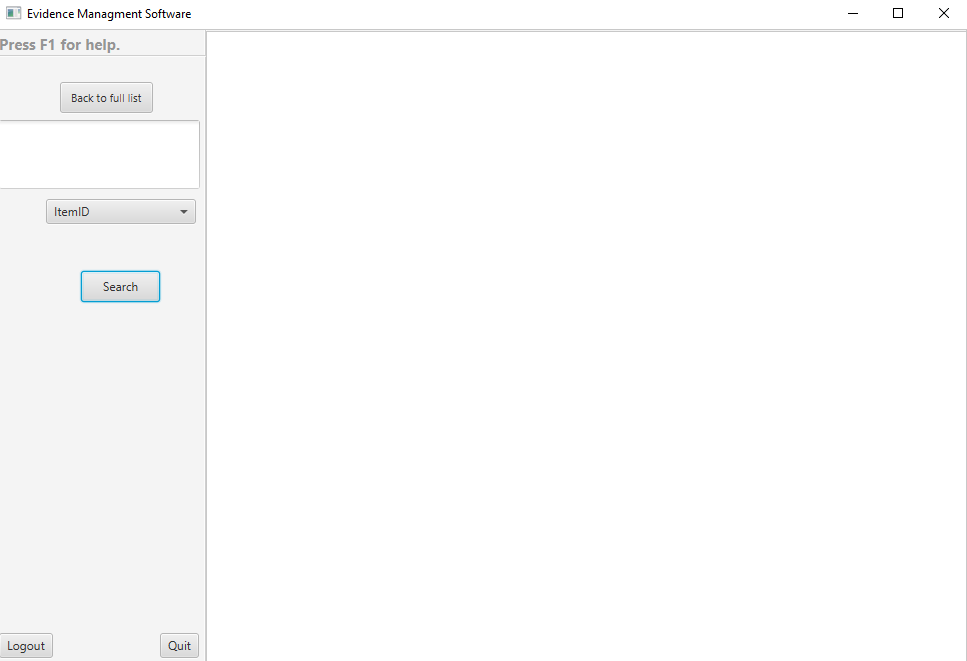
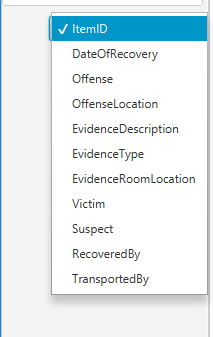
The graphical user interface of Evidence Manager system was designed using Java library called JavaFX. GUI works by changing the scenes. It consists of 2 buttons, 2 textFields and 2 labels.

(Picture 9).



Picture 9 Design of Login scene

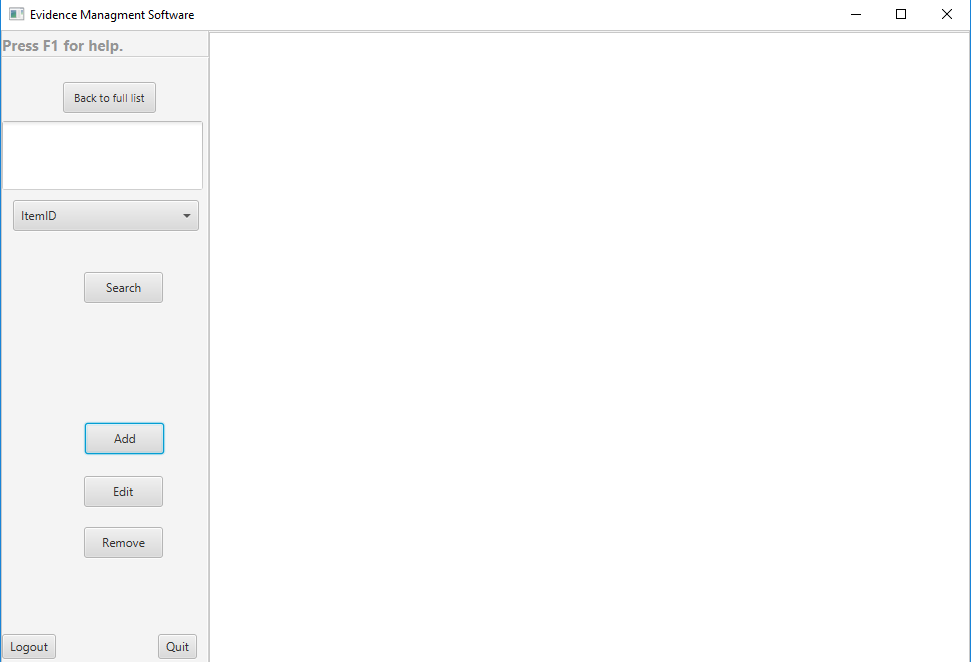
From login scene follows 2 scenes: a user scene and an administrator scene. Picture x and x shows the view for a user. User scene has the following: textField, buttons, combobox with multiple items. (Pictures 10, 11 and 12).



Picture 10 Design of combobox

Picture 11 Design of User view

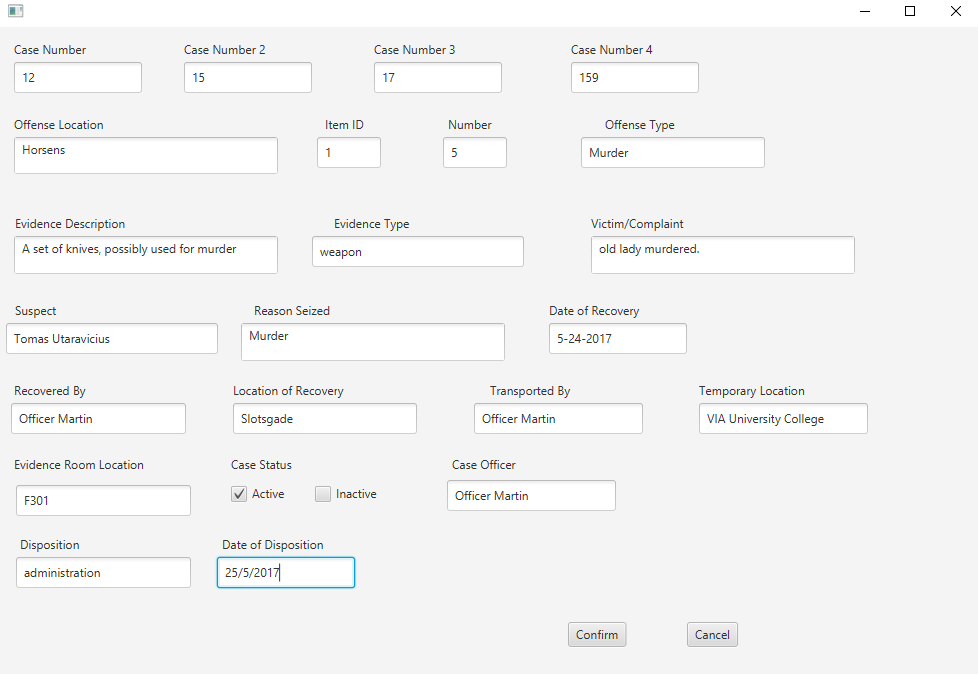
Combobox was choice of a design because it simplifies search function.



Picture 12 Design of Admin view

An administrator scene has access to all the system functions. Buttons “Add” and “Edit” change the scene, so user can add or edit data. In a figure below is shown a pop-up window for Add

scene. The scene corresponds to model design, by having same amount of data fields as Item class instance variables. (Picture 13).



Picture 13 Design of add scene

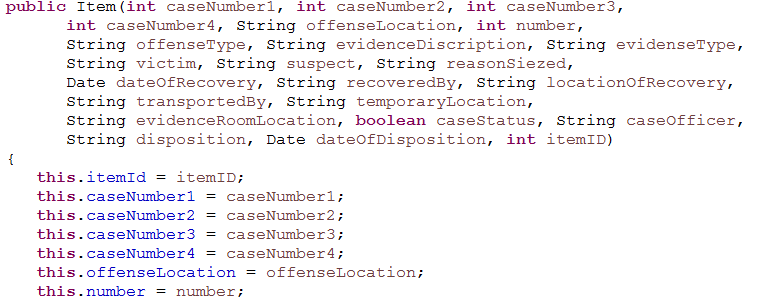
# Implementation

## Model

This part will focus on Add evidence data UseCase and how it was implemented.

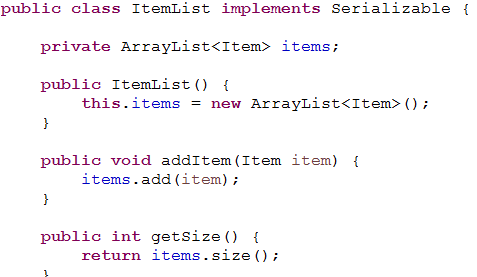
The Item class is implemented in a way, that it has same number of fields as the database.

(Picture 14).



Picture 14 Implementation of Item constructor

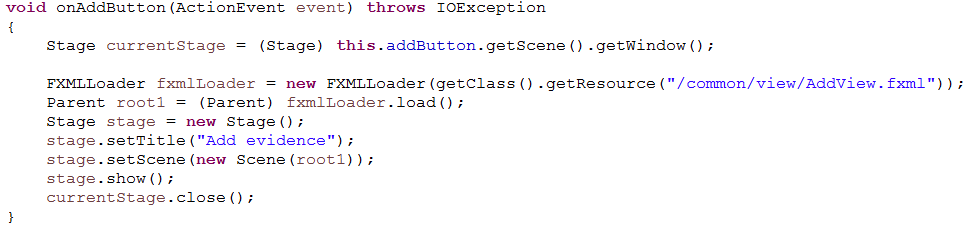
List of parameters in constructor makes sure that when new evidence item is created, all fields are filled. New evidence items are stored in ItemList. (Picture 15).



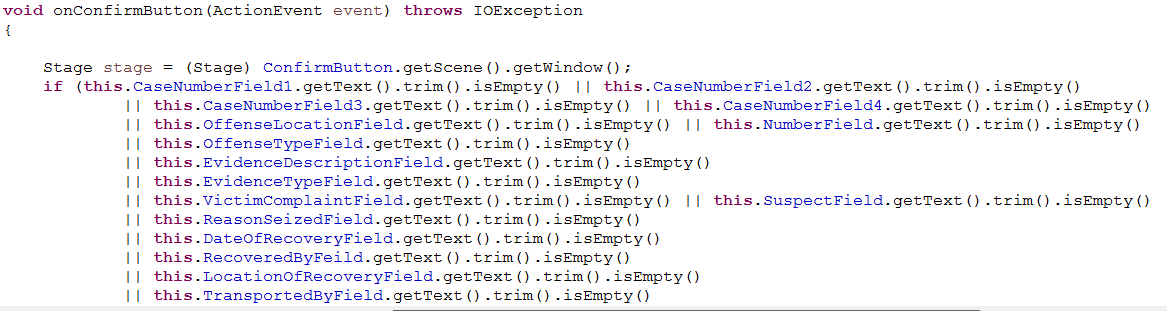
Picture 15 Implementation of ItemList

As shown above, ItemList uses ArrayList to store items.

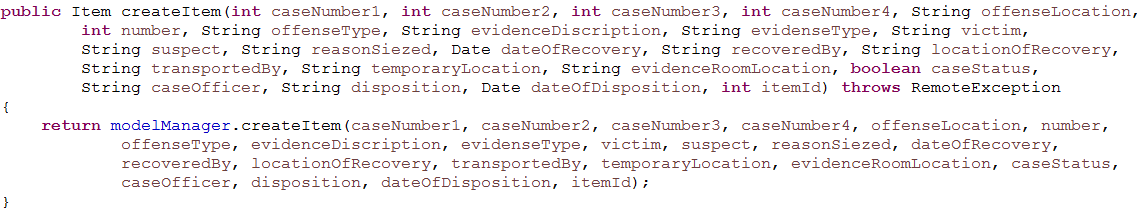
The controller classes get input from user through the client. The onAddButton method loads new scene, that enables user to create a new evidence item. (Picture 16 and 17).



Picture 16 Button controller



Picture 17 Implementation of button controller

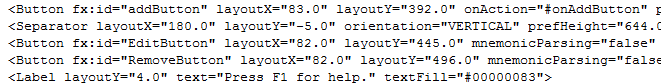
After data fields are filled, request is sent to the server to create a new item, using ModelManager. (Picture 18).

Picture 18 Implementation of ModelManager

After new evidence item is created, it is added to the database.

New item in the database is correctly displayed in the Client.

## GUI

Graphical user interface was implemented using JavaFX FXML, which is an XML-based language that provides the structure for building a user interface separate from the application logic of your code. The view user sees is separated into several scenes, which show depending on what user input Client receives. (Picture 19)

Picture 19 Implementation of buttons using JavaFX FXML

Creation, initialization, and addition are done in one line of code.

This part creates “back to full list button” and assigns a method to it.

Creation:



Initialization:

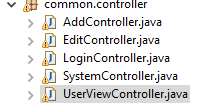


Adding a method to the button:

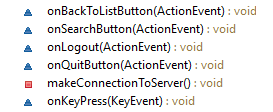


Addition in FXML does not require any extra code.

For each scene, a controller was created: (Picture 20).

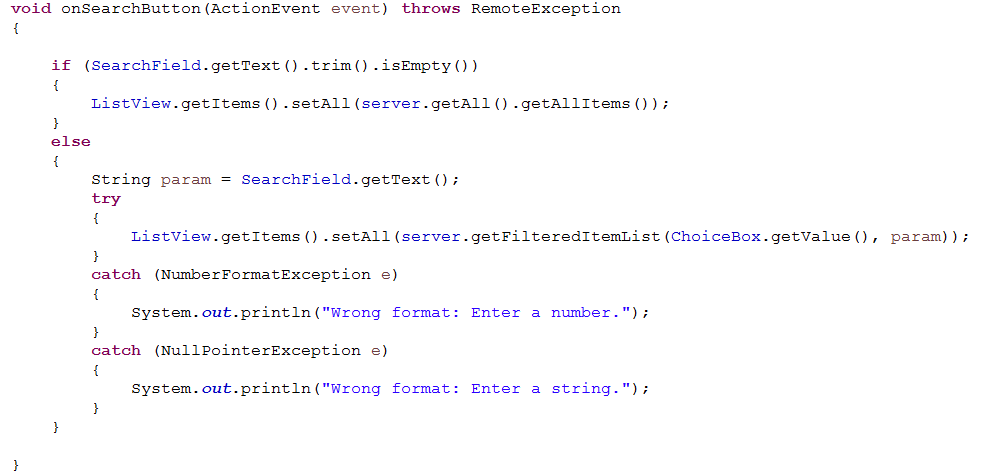


Picture 20 Controller package

Each controller has actionListeners for all the buttons in the scene. (Picture 21).

Picture 21 Controller overview

The following picture expands onSearchButton method. (Picture 22).



Picture 22 Implementation of controller

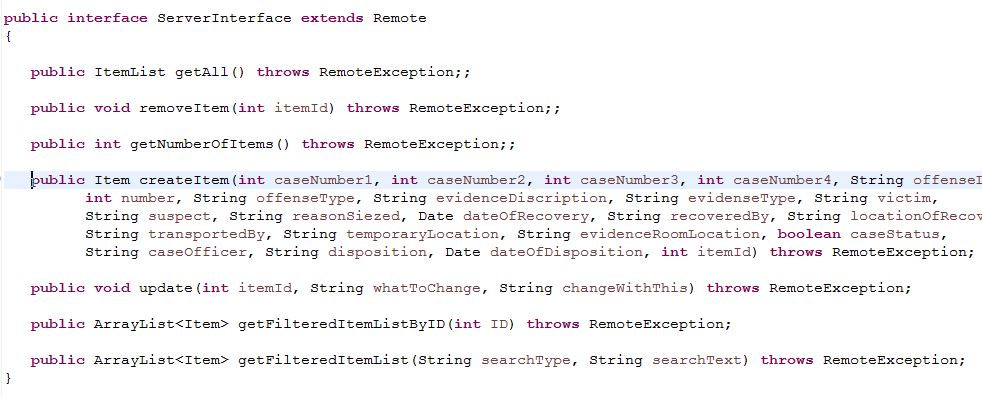
Every time an action in client is taken, client refreshes the list of evidence data, in order for user to see the latest version of it.

## Connection Client-Server

RMI programming model.

The capabilities of remote objects are expressed through interfaces on the client/server side and they must extend Remote defined in the java.rmi package.

Since the server holds all the information e.g. the list of evidence items all the remote methods are on the ServerInterface class as per below. (Picture 23).



Picture 23 Implementation of ServerInterface

They can be summarized into the below:

getAll() – retrieves the complete list of items;

getNumberOfItems() – returns the number of items;

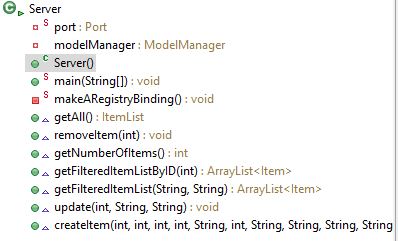
createItem() – creates a new evidence item, add it to the list;

update()- edit a specific item;

getFilteredItemListById() – find a specific item and return it;

getFilteredItemList()- create a custom list by a specific criteria;

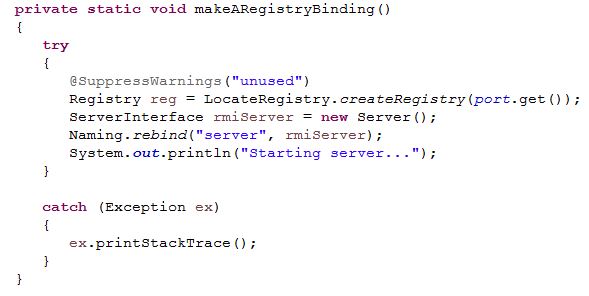
All the methods have been implemented in the Server class. The Server has to extend UnicastRemoteObject and have to implement the ServerInterface. (Picture 24).



Picture 24 Server overview

For the client/server architecture to function the Server class has to be started.

The RMI connection was implemented with the makeARegistryBinding() private method. (Picture 25).



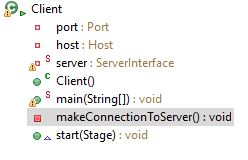
Picture 25 Making a registry binding.

The RMI registry is created and the name of the server is bound to a stub object.

The methods work with the ListOfEvidenceItems provided by the ModelManager class which is implemented as a facade to the ItemList class.

The methods implementation is basic and a full description of them can be found in the javadoc.

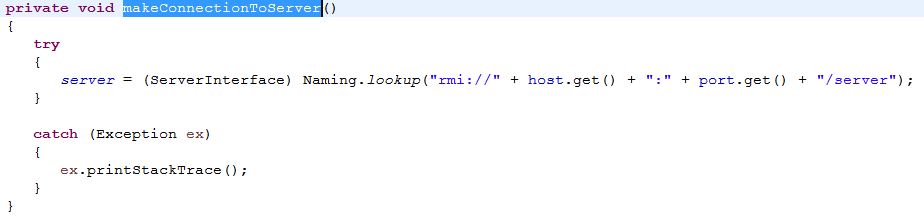
The Client class was implemented to extend the Remote package from the java.rmi and the Client class to implement it. (Picture 26).



Picture 26 Client overview

The Client class was also implemented and to invoke the methods from the remote object on the Server, the connection must be established and the stub with the connection information obtained. This is accomplished through the makeConnectionToServer() private method.

(Picture 27).



Picture 27 Connection to server

The lookup for the Stub is called on first time the Client is created and thus creating the connection. Once established all remote method invocation pass through the stub.

## Database

This is the constructor for the class DatabaseAdapter, first of all, it initializes the temporary ArrayList object, where all the data stored in the database will be saved in that object, that can be used later on by other classes. Next, there is a connection, with three parameters, which is essential to connect it to the database.

The method getAll() inside the class DatabaseAdapter assures that all the items stored in the database are retrieved and saved in the ArrayList object. (Picture 28 and 29).



Picture 28 Implementation of database adapter

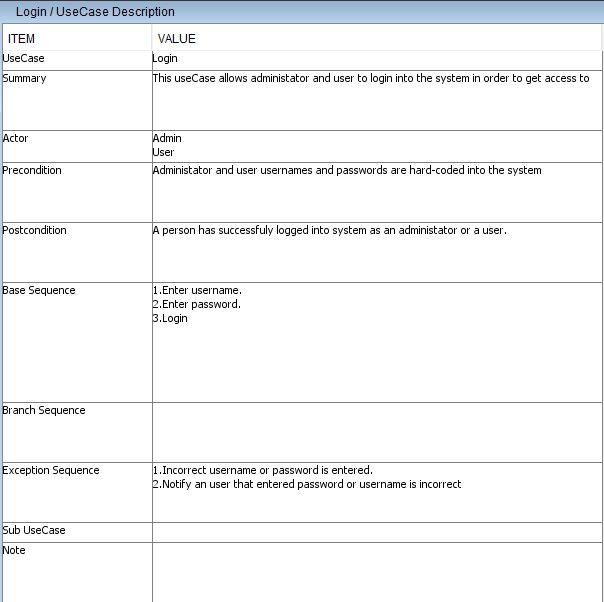
To create one more row in the database, there is another method called create the item, in this. Using SQL queries and class Statement it is possible to add more data from specified parameters.

Picture 29 Implementation of database adapter

# Testing

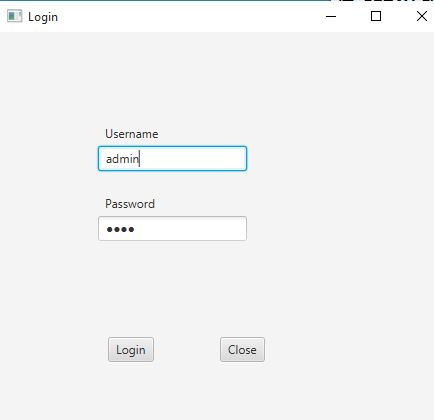
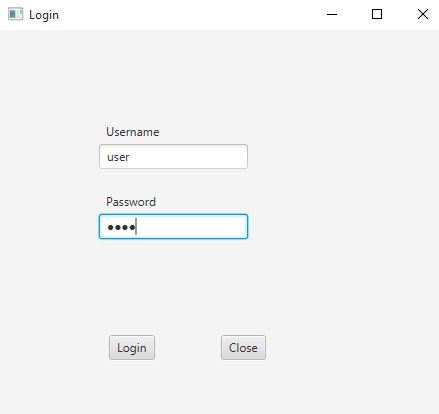
## Test case #1

Test Case Login



Subcase #1 -testing legal values e.g. User/Admin

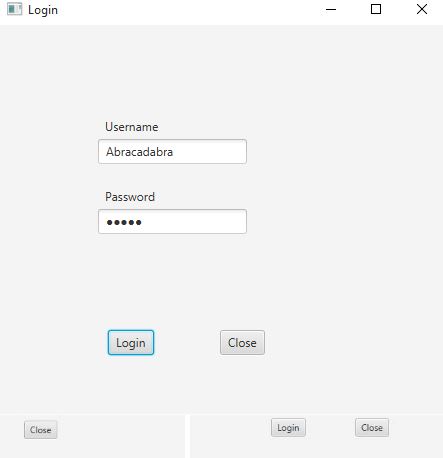
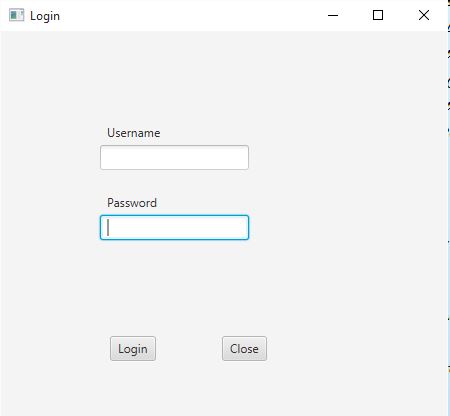
Expected result-login successful



Result: using the correct credential allows the user to login successfully.

Subcase #2 -testing illegal values e.g. Incorrect username/password/empty fields.

Expected result-login failed, does not show an error message.



Result: using wrong username or password or just trying to login with blank fields does not allow the user to login and nothing happens.

Improvement: System to show error message.

## GUI

The GUI was tested by making sure all the buttons worked properly, scene were being shown correctly, the whole design looked properly, and the action events were being executed.

# Results

The Evidence Manager system is a tool that was created with the purpose of let a police department can keep track and properly manage the entirety of its evidence storage. The system was made using a Client-Server architecture, as well as a database containing all the stored data.

On the client side, there are two parts – administrator and user. The user has a limited set of functions to use, while the administrator has a full access to the functions of the whole system. Tasks that can be performed from the client:

-Search/View Evidence(User/Administrator)

-Add Evidence Items(Administrator)

-Remove Evidence Items(Administrator)

-Update Evidence Items(Administrator)

A Graphical User Interface was created, as well, in order give a more user-friendly way to use the program and allow non-technical-savvy people to work with it.

The overall result is a system that is functional, however with a lot of room for improvement, such as scalability.

# Discussion

The system in its core is fully functional and that becomes apparent after going through a thorough testing.

It is very important to mention that a lot of features that would make sense in the program were not included due to the limit of time. This piece of software was made with the idea of it being scalable. What that means, while a lot of features were made for a small system, it can effectively transform and expand into something much larger.

The system, while functional in its core, does have problems and things that can be further fixed. Scalability is an important topic in system development, so it makes sense to point out several places that are not scalable and can be improved. Specifically, the list of items that is always shown at the beginning, when opening the program, is not scalable, because there is a possibility of having an enormous list and that would impact the efficiency of the program. While this makes sense in a small, isolated system, in the future, if it is to be expanded, a possible solution is to not show a list at all, until a user request one. As mentioned in the database documentation, it is also not scalable, since normalization was excluded but it can be further implemented if the system was to be improved in the future.

It was also important to create the system in a very structured, well organized manner, such that following work on it is possible to do without much hurdle.

# Conclusion

Overall, the system can execute its core functions, but it has several minor issues. The purpose of the project was to create a piece of software that enables police departments to properly manage their evidence. The goal was accomplished, but a lot of room for improvement is left.

The Client-side GUI is intuitive and straight-forward, The Client-Server architecture, using RMI, lets more people be connected to the server and thus having access to the Evidence Manager from different computers.

The management part the system, which is its center, is implemented, but is imperfect. The basics are working, however there are situations where different design would make more sense – such as the making the ID auto-generated, instead of taking the input by the user

What the system is also lacking is scalability, as mentioned previously in the report and is the main thing that could be improved in future updates, along with adding extra features.

## Perspective:

The system does not currently consider security and encryption, which is apparent from the way the Login system that was implemented. Additionally, the server does not include GUI, so working with it extensively is not possible. Validity of data checking also is not considered and while in the real-world storage space is a problem, our system does not try to solve it.

# References and expected sources:

1. Crime and Criminal Justice Statistics (May 2016). Retrieved from http://ec.europa.eu/eurostat/statistics-explained/index.php/Crime\_and\_criminal\_justice\_statistics

2. Laudan, L., Allen, J. R, The Devastating Impact of Prior Crimes Evidence and Other Myths of the Criminal Justice Process, 101 J. Crim. L. & Criminology 493 (2013). http://scholarlycommons.law.northwestern.edu/jclc/vol101/iss2/4

3. "Evidence, Chain of Custody" World of Forensic Science. Retrieved March 26, 2017 from Encyclopedia.com: http://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/evidence-chain-custody

4. Gulick, G. (2008), “Storage Technology”, Evidence Technology Magazine(Volume 6, Number 1): http://www.evidencemagazine.com/index.php?option=com\_content&task=view&id=92

5. Duerr, T. E., Beser, N. D., Staisiunas, G.P.(2004), "Information Assurance Applied to Authentication of Digital Evidence". Forensic Science Communications. Federal Bureau of Investigation. Archived from the original on March 12, 2008: https://web.archive.org/web/20080312121928/http://www.fbi.gov/hq/lab/fsc/backissu/oct2004/research/2004\_10\_research01.htm

6. "Property Room Standards". International Association for Property and Evidence. Archived from the original on 2008-02-04. Retrieved 2008-03-31. https://web.archive.org/web/20080204172810/http://www.iape.org/Standards\_7-03/index.htm

7. "Property Standards #7". International Association for Property and Evidence. Archived from the original on 2008-04-23. Retrieved 2008-03-26. https://web.archive.org/web/20080423052936/http://www.iape.org/Standards/PS07/index.html

8. "Evidence handling". Minnesota Department of Corrections. Archived from the original on 2008-05-31. Retrieved 2008-03-31. https://web.archive.org/web/20080531155547/http://www.doc.state.mn.us/DocPolicy2/FormsWorking/NSI/107.055RC.htm

9. Zuckerman, M. A., Yes, I Destroyed the Evidence – Sue Me? Intentional Spoliation of Evidence in Illinois (April 5, 2011). John Marshall Journal of Computer & Information Law, 2010. Available at SSRN: <https://ssrn.com/abstract=1536805>

10. Stebick, Divonna, and Jonelle Pool. "Piloting a Digitized Evidence-Based Assessment System," American Association of Colleges for Teacher Education, 2006.

<http://cupola.gettysburg.edu/cgi/viewcontent.cgi?article=1013&context=edfac>

# Appendices

Appendix A – Use Cases and Activity Diagrams

Appendix B – Class Diagram UML

Appendix C – Test Cases

Appendix D – Project Description

Appendix E – Product and Sprint Backlog

Appendix F – Scrum Meetings