Criminal Investigation Tracker with Suspect Prediction
Analysis and Design Document

Student: Han Ana-Maria
Group: 30433

Revision History

Date	Version	Description	Author
01.04.2018	1.0		Han Ana-Maria
06.05.2018	2.0		Han Ana-Maria
28.05.2018	3.0		Han Ana-Maria

Table of Contents

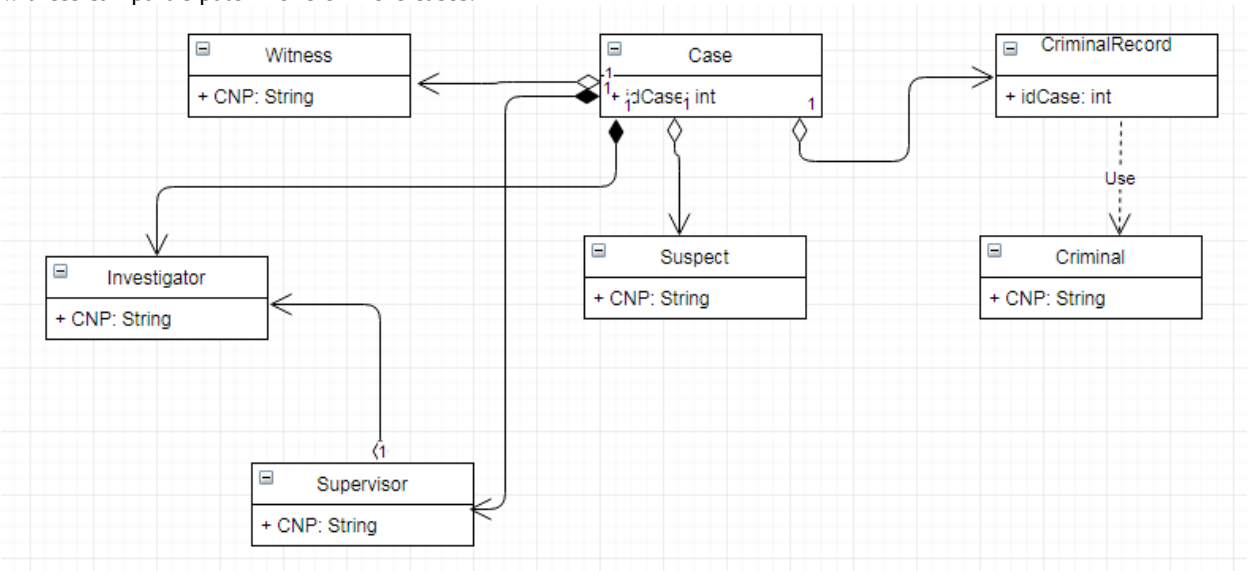
I. Project Specification

The problem criminal investigation agencies are facing is finding possible suspects as soon and efficient as possible, with a minimum amount of data. While human reasoning and intuition are very important and powerful factors in crime investigations, the forensic investigators might be biased and lose some aspects out of sight when trying to put the pieces together to discover the perpetrator. Thus, there is an urgent need to have a kind of technology to help ease and speed up the process of finding suspects, even only with some guidance towards the possible criminal, and do it as accurately as possible.

II. Elaboration – Iteration 1.1

1. Domain Model

The domain model consists of the following classes: Investigator, Supervisor, Suspect, Criminal, CriminalRecord, Case, Witness, where a supervisor has a list of investigators to supervise, a criminal can have one or more criminal records, each case can have one or more suspects and criminal, and zero or more witnesses. A criminal record is linked to one case, but a case can result in more criminal records, and a witness can participate in one or more cases.



2. Architectural Design

2.1 Conceptual Architecture

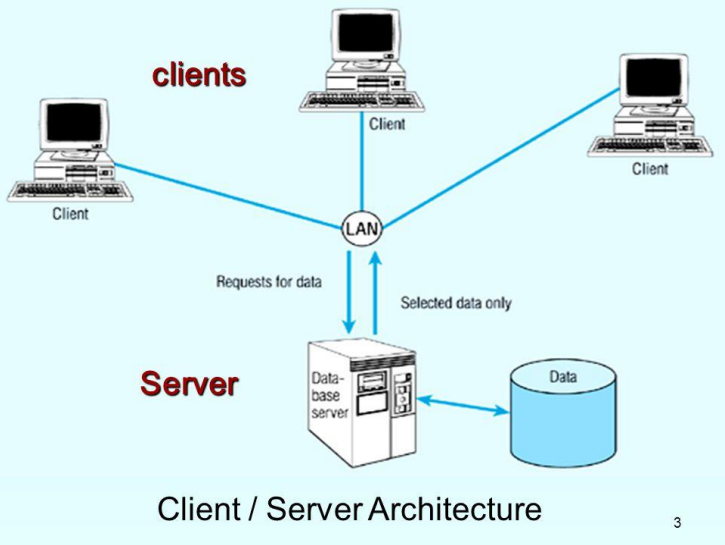
For this project, I am going to use the client-server architecture style, complemented by the MVC architectural pattern.

The **client–server model** is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. [1]

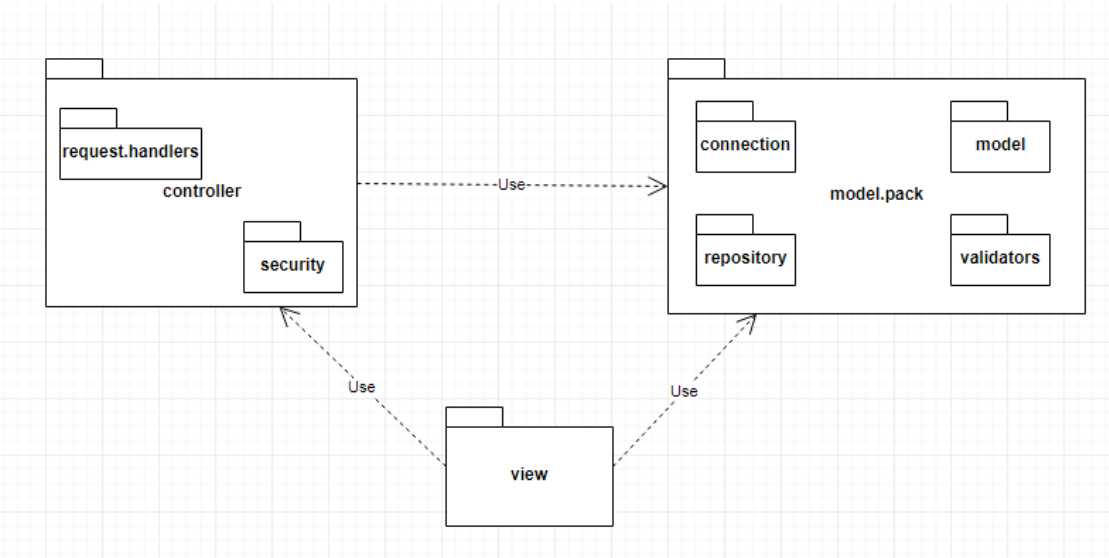
Model–view–controller (**MVC**) is an architectural pattern commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the

user.[1][2] The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development. [2]

I have chosen these two, first of all because my application requires multiple users that would have to enter data from multiple computers for an efficient use of the application, and it is also important that the project is split into several modules that deal with a certain component, such as the database (model), the user (view, interface), and the logic that makes the connection between the two and deals with all the computations needed to yield correct results (controller). Thus, the supervisors and investigators would input data and ask to see the progress of each investigation, as well as request for a list of suspects when sufficient information is entered. All the cases, suspects, and criminal records are stored in a database and will be retrieved, of course, whenever they are requested (an investigator can only retrieve information about the cases they are assigned to, while a supervisor can view all the data in the database).

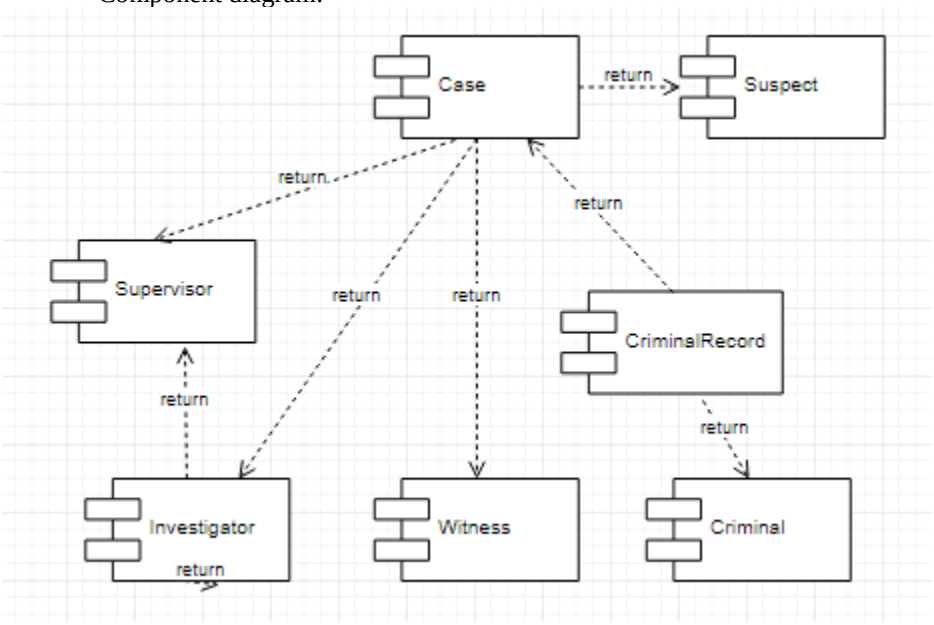


2.2 Package Design

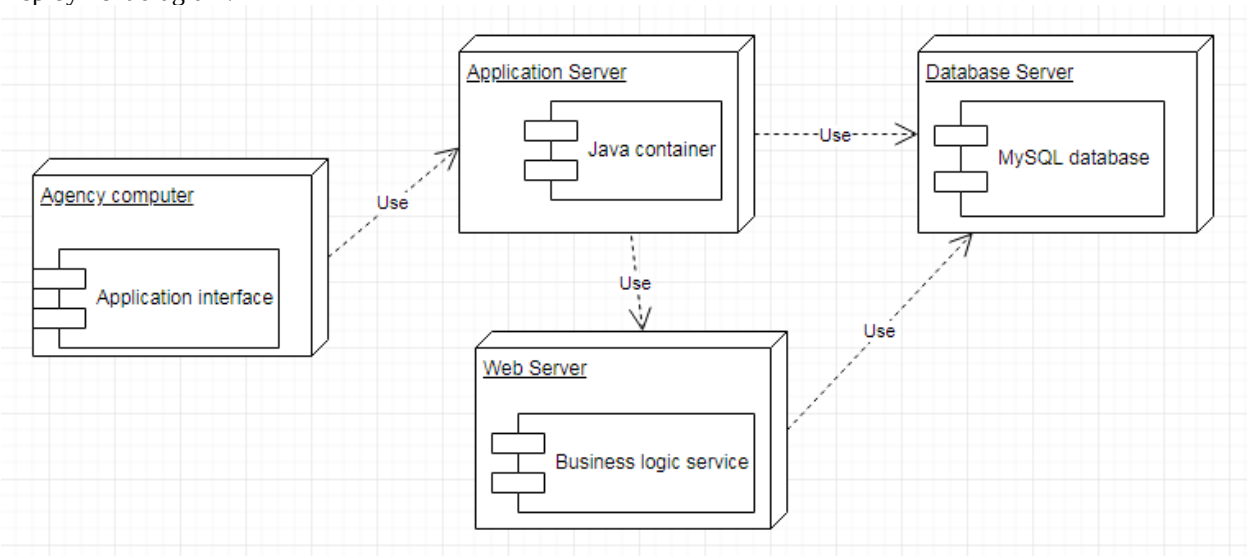


2.3 Component and Deployment Diagrams

Component diagram:



Deployment diagram:

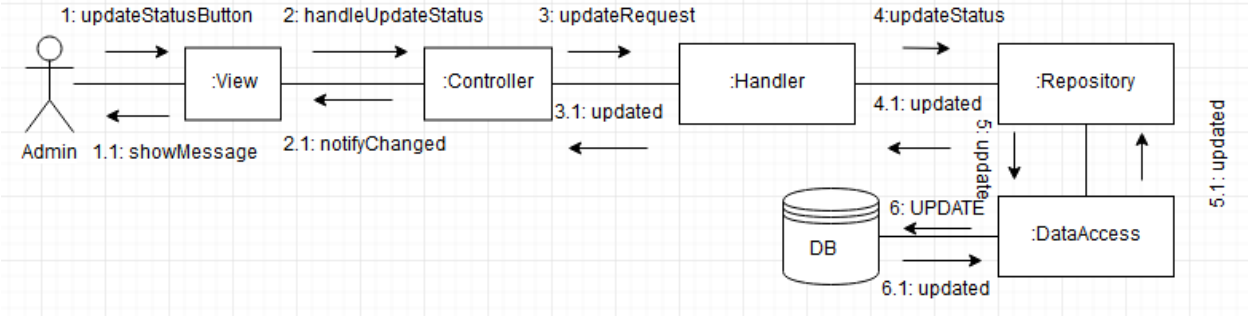


III. Elaboration – Iteration 1.2

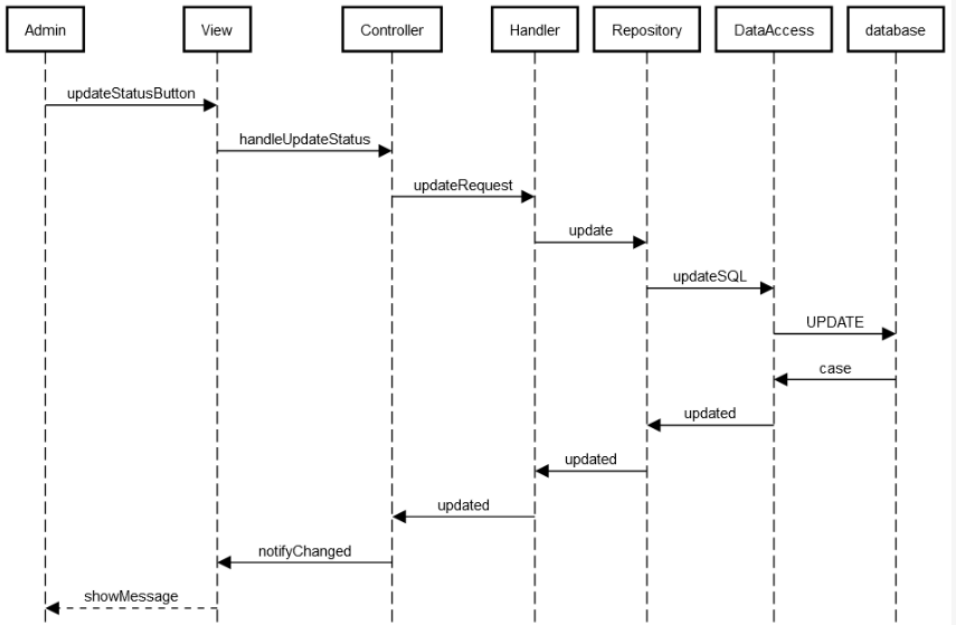
1. Design Model

1.1 Dynamic Behavior

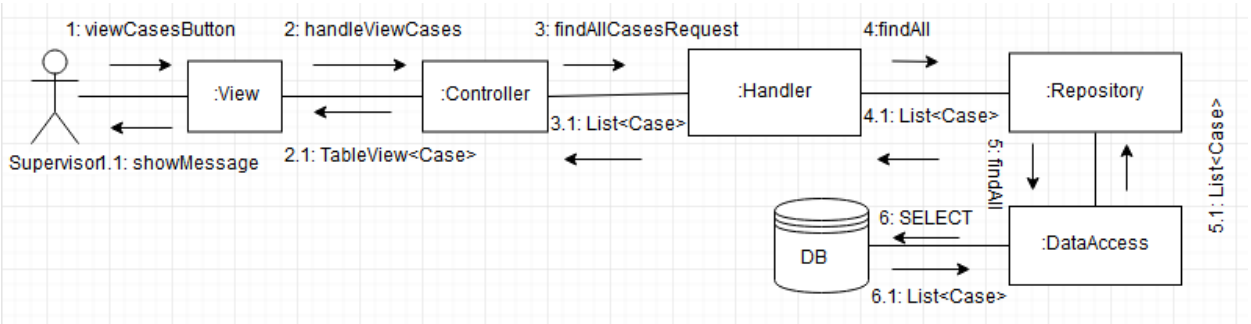
Communication diagram for admin updating crime investigation status:



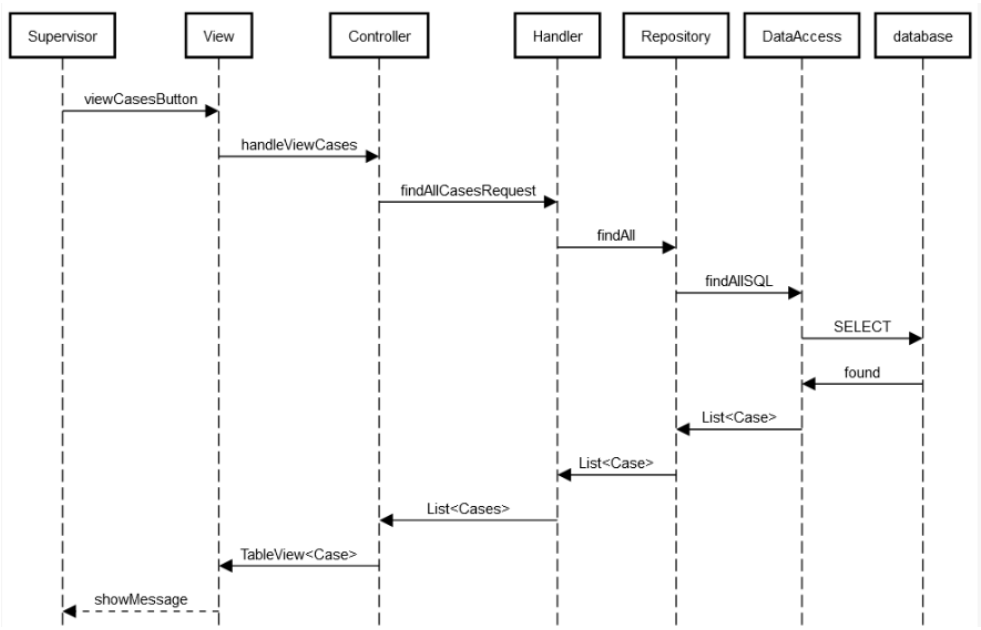
Sequence diagram for admin updating crime investigation status:



Communication diagram for a supervisor who wants to view all existing cases:



Sequence diagram for a supervisor who wants to view all existing cases:

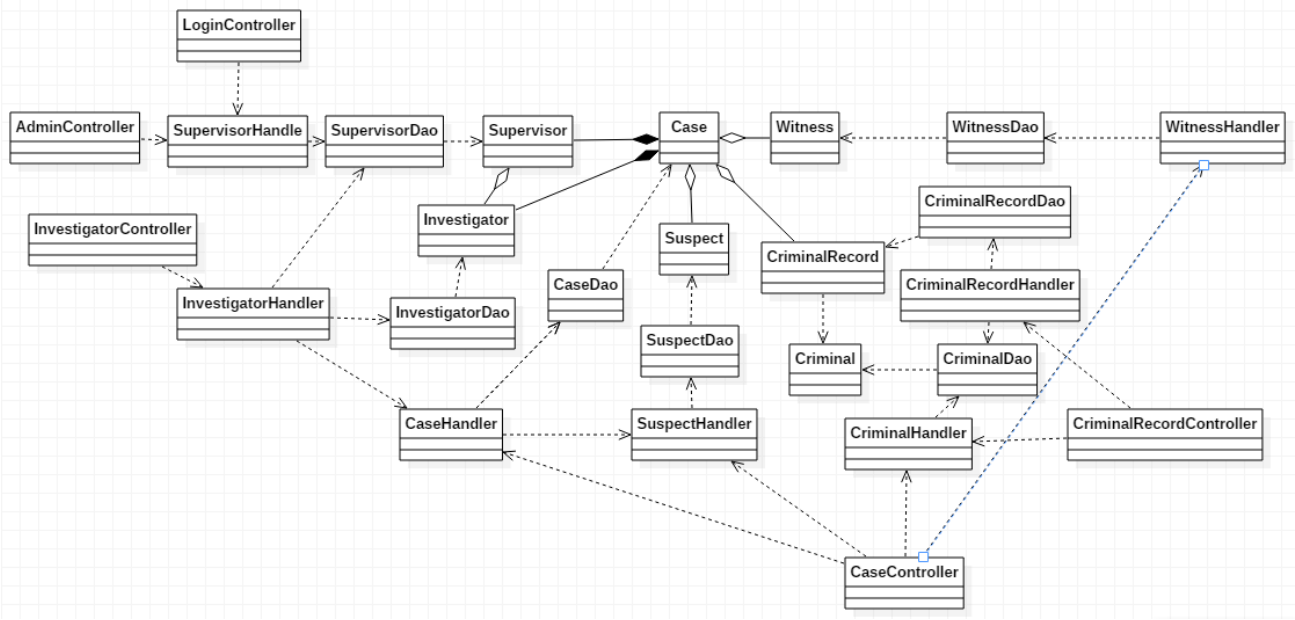


1.2 Class Design

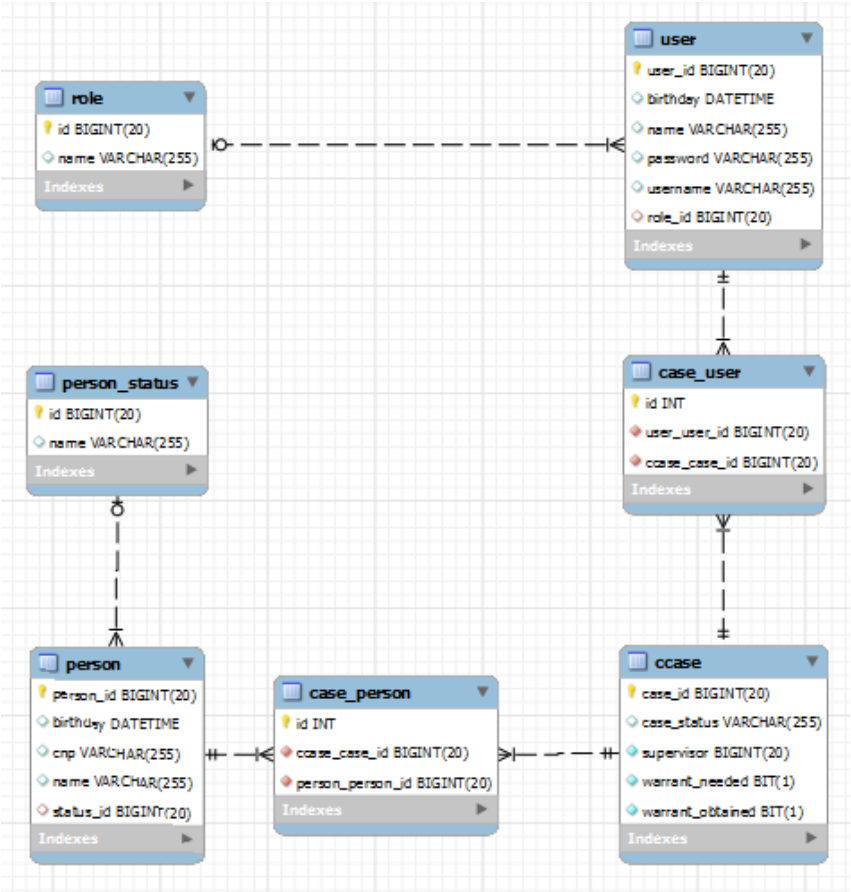
The two design patterns I intend to use are Builder and Iterator.

The Builder DP will be implemented for the domain model objects creation in order to make it easier to create entity objects by avoiding parametrized constructors that need many fields to be instantiated. Each entity object encapsulates a Builder class, which initializes their fields and returns the newly created object, thus separating the construction and representation of the complex entities. The creation of objects is delegated to the Builder classes.

The Iterator pattern will be used to traverse a container and access the container's elements, such as the criminal records of a criminal or the list of suspects for an investigation case, without exposing details of their implementation. The solution it proposes is defining a separate (iterator) object that encapsulates accessing and traversing an aggregate object. Different iterators can be used to access and traverse an aggregate in different ways. New access and traversal operations can be defined independently by defining new iterators.



2. Data Model



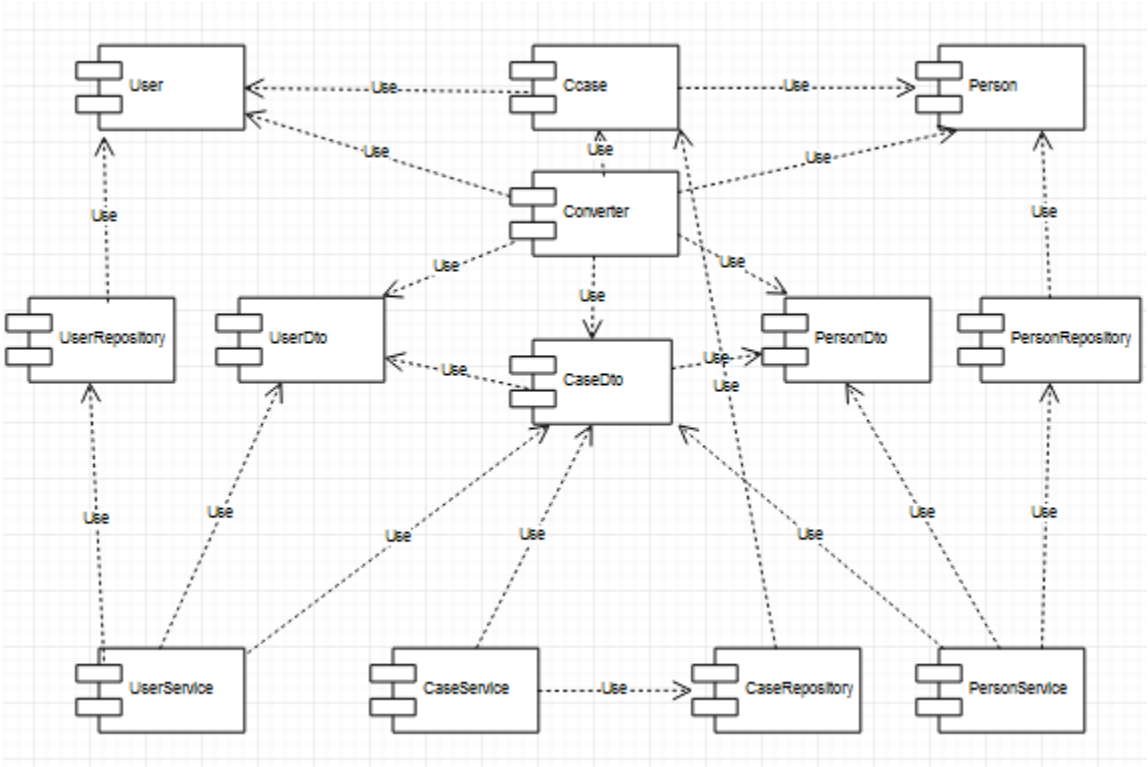
3. Unit Testing

The test case scenarios I want to take into consideration are:

- The application must not allow the investigators and supervisors to make any change to a closed case.
- The supervisors can view all the cases, but can only write reports and provide information to those they are assigned to.

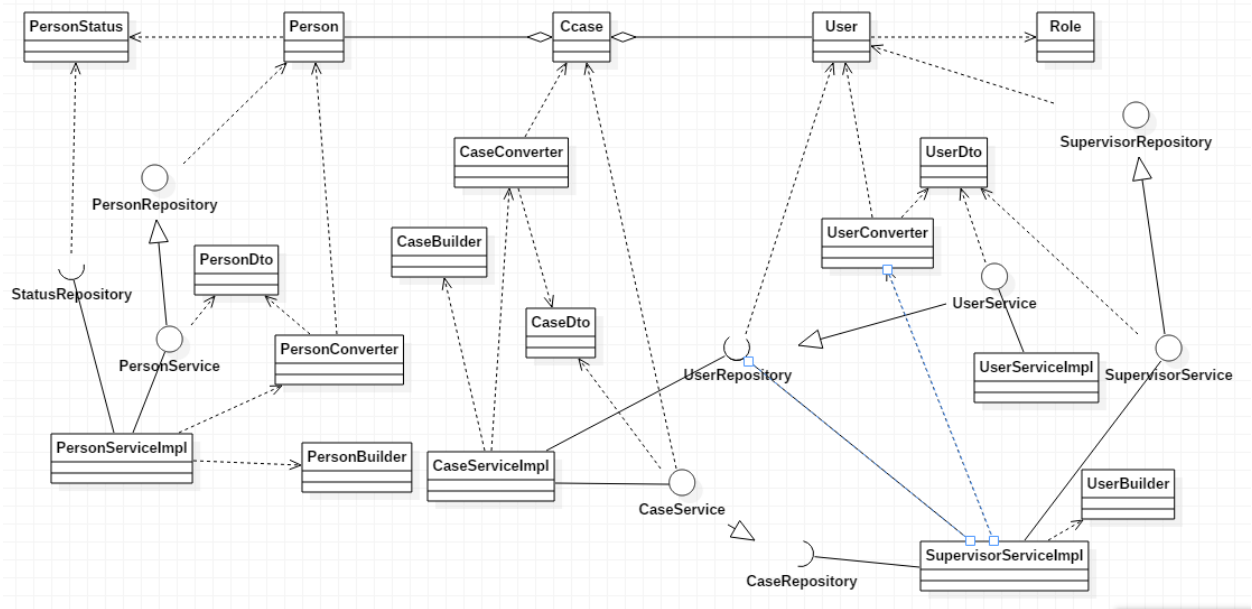
IV. Elaboration – Iteration 2

1. Architectural Design Refinement



Other than the component diagram and the fact that I haven't implemented the client-server architecture, the previous specifications remained pretty much the same.

2. Design Model Refinement



The most important change that occurred within this project was the change of the domain model, which

no longer consists in separate classes for the users and the other stakeholders involved, but instead groups all users under one class (User, having a field of enum type for the role), and does a similar thing for the other people involved (criminals, witnesses, suspects). Next, for each entity there is a corresponding repository that extends the JpaRepository, making handling CRUD operations a lot easier. Also, for each entity there is a corresponding Service class which implements its specific operations by implementing the Service interface that extends the corresponding repository mentioned above. I have also implemented DTO classes for handling the actions performed in the HTML pages, and some classes for converting from the entity classes to the DTO ones and vice-versa.

V. Construction and Transition

1. System Testing

The system testing was done by using **System.out.println** throughout the entire application in order to make sure that the values retrieved from or inserted into the database are the expected ones, but also that the intermediate results inside the application propagate correctly.

2. Future improvements

As future improvements, I would like to enable the investigators to add evidence to the cases they are involved in, each piece of evidence helping to identify the suspects, which would be automatically added or removed from the list of suspects of a case as more evidence is inserted. Each time an investigator adds a piece of evidence to a case, the supervisor and all the other investigators on that case are notified. The supervisor must request search warrants and send reports of the cases they are assigned to to the judges. Once a case is closed, no more evidence can be added to that case, unless specifically requested by its supervisor. The investigators can only see the cases they are assigned to, while the supervisors can see all the cases.

VI. Bibliography

[1] “Client-server model”: https://en.wikipedia.org/wiki/Client–server_model
[2] “Model-view-controller”: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller