



Ain Shams University / Faculty of Engineering

CSE Department

Software Engineering (1) CSE128

Hospital Management System

Submitted By

Mostafa Mahmoud El-Rosasy 16P8113

Omar Magdy Shaaban 16P6034

Peter Nabil Zaghloul 16P8100

Seif El Din Abdel Hakim 15P8111

Seif El Din Mohamed 16P8109

Group 2 Section 1

Submitted To

Dr. Gamal Abdel Shafy

Cairo, 2018

Abstract

This project aims to deliver a software that is fulfilling the requirements of building one from scratch using engineering principles and disciplines. Teamwork is indispensable for conveying an idea to creating a successful software project. In this project, we will demonstrate our software, its capabilities, constraints, diagrams, and user guide with snapshots to provide full documentation of the software and assure easy and practical use for the users.

Contents

Figures.....	3
1. Introduction	4
1.1 Purpose	4
1.4 Overview	4
2. General Description	5
2.1 Product Perspective	5
2.2 General Capabilities	5
2.3 General Constraints	5
2.4 User Characteristics	5
2.5 Environment Description	6
2.6 Assumptions and Dependencies.....	6
3. System Requirements	6
3.1 Functional Requirements.....	6
3.2 Non-functional Requirements	7
4. Use Case Diagram	7
5. Narrative Description of Use Cases.....	8
6. Requirements Validation	16
7. Class Model	17
8. State Diagram.....	18
9. Sequence Diagram	19
10. Detailed Class Diagram	20
11. User Interface Design.....	21
15. Estimated Project Cost.....	21
16. User Guide	23

Figures

Figure 1: Use Case Diagram	7
Figure 2: Abstract Class Diagram	17
Figure 3: State Diagram.....	18
Figure 4: Sequence Diagram	19
Figure 5: Detailed Class Diagram	20
Figure 6: HMS window	23
Figure 7: Admin window	24
Figure 8: Patients window	24

1. Introduction

1.1 Purpose

This document aims to provide the users, mostly unexperienced ones, with a simple user guide that enables the users to use the software correctly and efficiently. Also, documentation is provided for all the software process features and steps, to trace every detail during the design and development of the software.

1.2 List of Definitions

- Use case: Required functionality of the system.
- Extend: Performs another use case then return to the current use case.
- Include: Performs another use case that can be considered as a step inside the current use case.
- Client: A class that sends a message to another class to perform a certain function.

1.3 Scope

The software name is “Hospital Management System”. The software mainly adds new patients to the system providing needed information like name, address, age ... etc. and also allows editing of the information of certain patients and eventually remove some patients’ from the records of the system.

1.4 Overview

In this document, we will present a general description of the software describing its functionalities and its constraints, then showing functional and non-functional requirements, use case diagram of the system in UML along with narrative description of all the use cases, user requirements validation using source and traceability techniques, class model using both Noun Extraction and CRC methods, state diagram and sequence diagram, then again class model but more sophisticated showing the interaction between all classes in the software, user interface design, client-object relation diagram, pseudocode implementation of the software, test cases, estimated cost of the project, and a user guide with screenshots to enable a proper and smooth use of the software.

2. General Description

2.1 Product Perspective

The software will depend on a server that will store and retrieve patients' data and records upon existence of an authorized account (successfully logged in account). Collaboration with other systems like "Archive" or "Employees Database" won't be included in the implementation of the software, as it might complicate the way the functionalities will be performed by the system. So, the system functionalities will be based only on the server.

2.2 General Capabilities

The system can do the following:

- 1) Add a patient to the system along with their information (ID, name, age, address, e-mail, disease, treatment, date of birth, date of entry, and date of exit), provided that the current patient isn't already registered on the system.
- 2) Search for a certain patient.
- 3) Edit a patient's information.
- 4) Remove a patient from the system.

2.3 General Constraints

Certain constraints are implied on the system including implementing language, execution speed, reliability, budget, delivery time, and ease of use (these constraints will be clarified in the non-functional requirements in Section 3). Also, the system is required to have a user-friendly interface with the capability of showing adequate error messages such that unexperienced users can understand the problems encountered and know how to solve such problems.

2.4 User Characteristics

The system doesn't use any difficult terminologies for users. The only related fields to the medical terminology are "Disease" which simply asks for the disease name, and "Treatment" asks for the drug name and dose perhaps if the user wanted to enter such information. So, the system is adequate for the users including the doctors and the nurses.

2.5 Environment Description

The software runs on the following Windows operating systems: XP, Vista, 7, 8, 8.1, and 10. Also, it can run on MacOSx. The software needs as well JRE (Java runtime environment) to operate and an Intel/AMD/Nvidia processor of speed no less than 1GHz. Finally, it will be required a free space of at least 25 MB.

2.6 Assumptions and Dependencies

The implementation of the software is based on the need of the user to find a system that is fast in execution, with minimal number of errors, early delivery, and short budget. Also, providing a user interface design that will make the utilization of the software easier. The system is based on the assumption that only the server will perform the functionalities of the system without needing external systems like “Archive” for example.

3. System Requirements

3.1 Functional Requirements

1. The “Admin” can add “Users” and give them their usernames.
2. The “User” can login into the system using a username and a password. An error message should appear if the username/password/both are incorrect.
3. The “User” should be able to search for a patient with the patient’s ID. If the patient is not found, an error message should appear.
4. The system will show the “User” the patient’s information including his name, ID, address, DOB, phone number, date of entrance.
5. The “User” can add new patients to the list of patients, provided that the new patient isn’t already registered in the system.
6. The “User” can remove patients from the list.
7. The “User” can edit the information of a certain patient.

3.2 Non-functional Requirements

1. The system must not exceed 80 MB of RAM, 25MB in storage.
2. The system must use Form fill-in interaction style.
3. The implementation of the program should be written in Java.
4. The system delivery is in a month.
5. The budget doesn't exceed 15,000 L.E.
6. The training requires no more than 30 minutes, and the average number of errors made by not experienced users shall not exceed one error per week.

4. Use Case Diagram

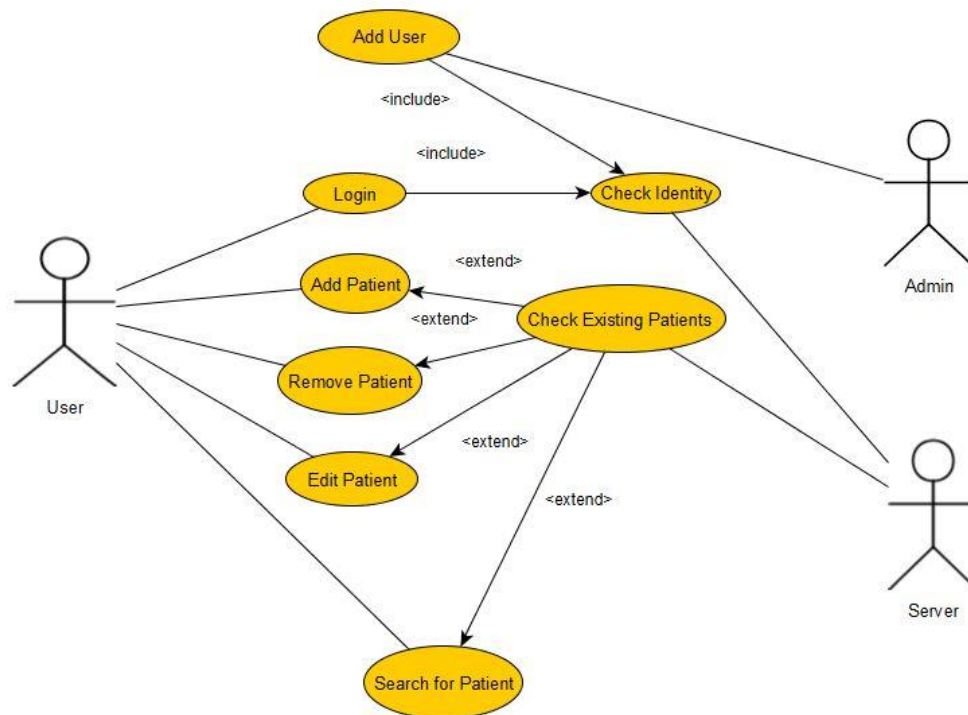


Figure 1: Use Case Diagram

5. Narrative Description of Use Cases

1) Use case name:

Add Patient.

Related requirements:

None.

Goal:

The system adds a certain patient.

Preconditions:

- 1) "Admin" must be logged into the system.
- 2) "Patient" must not be already registered in the application.

Successful end condition:

"Patient" is added to the database.

Failed end condition:

"Patient" isn't added to the database.

Primary actors:

"User".

Secondary actors:

"Server".

Trigger:

"Patient" data is entered for the "server" to be added in the application.

Main flow:

- 1) "Admin" presses the "Add patient" button.
- 2) "Admin" enters the "patient" data.
- 3) "Server" checks if the "patient" is saved in the application, if not then the "server" adds the "patient" in the application.

Extensions:

"Patient" is stored in the application so the system returns a message saying that the "patient" is an invalid entry.

2) Use case name:

Search for Patient.

Related requirements:

None.

Goal:

Find a "patient".

Preconditions:

The "user" is logged into the system.

Successful end condition:

The "patient" being searched for, is found.

Failed end condition:

The "patient" being searched for, isn't found.

Primary actors:

"User".

Secondary actors:

"Server".

Trigger:

"User" clicks on the "Search for Patient" button.

Main flow:

- 1) "User" enters the ID of the "patient".
- 2) "Server" searches for the ID of the "patient".
- 3) "Server" returns the "patient" data, if an identical ID is found.

Extensions:

- 1) "Server" doesn't find the ID of the "patient".
- 2) "Server" returns a "Patient not found!" message.

3) Use case name:

Edit patient.

Related requirements:

None.

Goal:

"Patient" data will be edited.

Preconditions:

"Patient" data should be stored in the application.

Successful End Condition:

"Patient" data is edited.

Failed End condition:

"Patient" data isn't edited.

Primary Actors:

"User".

Secondary Actors:

"Server".

Trigger:

"User" clicks on "Edit Patient" button.

Main flow:

- 1) "User" clicks on "Edit Patient" button.
- 2) "User" chooses "patient" to edit.
- 3) "User" edits "patient".
- 4) New edit is stored in the application.

Extensions:

- 1) "Patient" to be edited isn't found.
- 2) "Server" returns "Patient not found!" message.

4) Use case name:

Remove Patient.

Related requirements:

None.

Goal:

Remove "patient" from the application.

Successful end condition:

"Patient" is removed from the application.

Failed end condition:

"Patient" isn't found.

Primary actors:

"User".

Secondary actors:

"Server".

Trigger:

"User" clicks on the "Remove Patient" button.

Main flow:

- 1) "User" clicks on the "Remove Patient" button.
- 2) "User" enters "patient" ID.
- 3) "Server" searches for "patient" ID.
- 4) "Server" deletes "patient" data if "patient" ID is found.

Extensions:

- 1) "Patient" data isn't found.
- 2) "Server" returns a "Patient not found!" message.

5) Use case name:

Check Identity.

Related requirements:

Login, or Add User.

Goal:

"User" logs into the system/ adding a new "user".

Preconditions:

"User" entered his/her login data, or the "admin" will add a "user".

Successful end condition:

"User" data is verified and logs into the system/ a new "user" is added.

Failed end condition:

"User" data isn't verified and doesn't log into the system/ new "user" can't be added.

Primary actors:

"Server".

Secondary actors:

None.

Trigger:

"User" enters login data/ "Admin" is adding a new "user".

Main flow:

- 1) "Server" checks login data/ "user" credentials.
- 2) "User" logs into the system/ a "user" is added.

Extensions:

- 1) "User" data isn't verified.
- 2) "Server" returns a "User not found!" message.

6) Use case name:

Check Existing Patient.

Related requirements:

Search for Patient, Edit Patient, and Remove Patient.

Goal:

"Patient" is to be verified by the "server".

Precondition:

"User" chooses to either add/edit/search for/remove a "patient".

Successful end condition:

"Patient" is found in the application.

Failed end condition:

"Patient" isn't found in the application.

Primary actors:

"Server".

Secondary actors:

None.

Trigger:

"User" chooses to either add/edit/search for/remove a "patient".

Main flow:

- 1) "Server" checks for the "patient" existence.
- 2) "Server" returns the "patient" data.

Extensions:

- 1) "Server" doesn't find the patient.
- 2) "Server" returns a "Patient not found!" message.

7) Use case name:

Login.

Related requirements:

None.

Goal:

“User” logs into the application.

Preconditions:

“User” must have an account username and password to login to the system.

Successful end condition:

“User” logs into the system.

Failed end condition:

“User” doesn't log into the system.

Primary actors:

User

Secondary actors:

Server

Trigger:

“User” presses the “Login” button.

Included case:

Check Identity.

Main flow:

- 1) “User” types his/her username and password.
- 2) “User” login information is checked.
- 3) “User” logs into the system and gets access to the application.

Extensions:

- 1) The system doesn't verify the “user” details.
- 2) The system returns the details as unverified.

8) Use case name:

Add User.

Related requirements:

Check Identity

Goal:

"Admin" adds a user.

Preconditions:

"Admin" must have an account username and password to login to the system and add a "user".

Successful end condition:

"User" is added to the system.

Failed end condition:

"User" isn't added to the system.

Primary actors:

"Admin".

Secondary actors:

"Server".

Trigger:

"Admin" presses the "Add New User" button.

Included case:

Check identity.

Main flow:

- 1) "Admin" logs into the system.
- 2) "Admin" enters "user" info.
- 3) New "user" is added to the system.

Extensions:

- 1) System doesn't verify the "admin" details/ "Admin" enters wrong user data.
- 2) System returns details as unverified.

6. Requirements Validation

Requirements	Login	Add user	Login error message	Patient database	Add patient	Search patient	Remove patient	Edit patient	Patient not found message
Login			R						
Add user	D		R						
Login error message	D								
Patient database	D								
Add patient	D			D					
Search patient	D			D					
Remove patient	D			D					
Edit patient	D			D					
Patient not found message	D			D	R	R	R	R	

7. Class Model

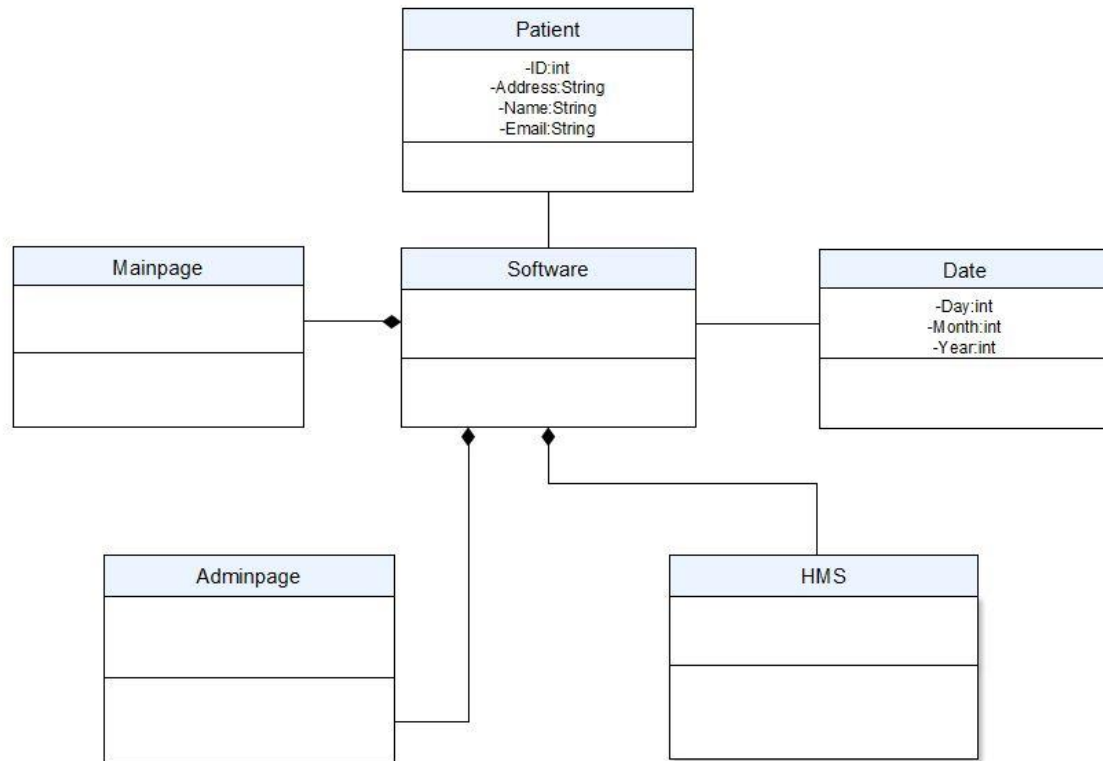


Figure 2: Abstract Class Diagram

8. State Diagram

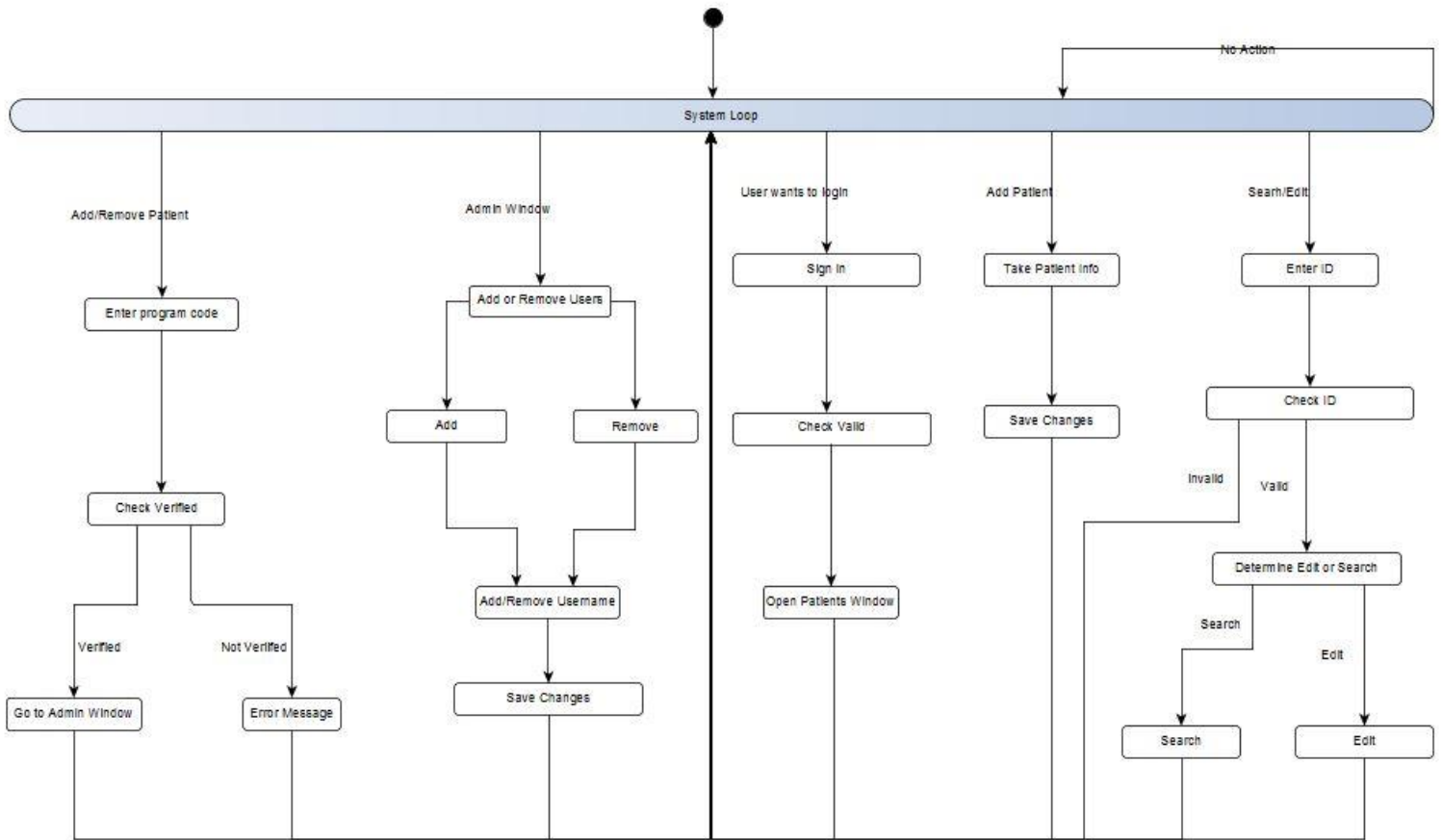


Figure 3: State Diagram

9. Sequence Diagram

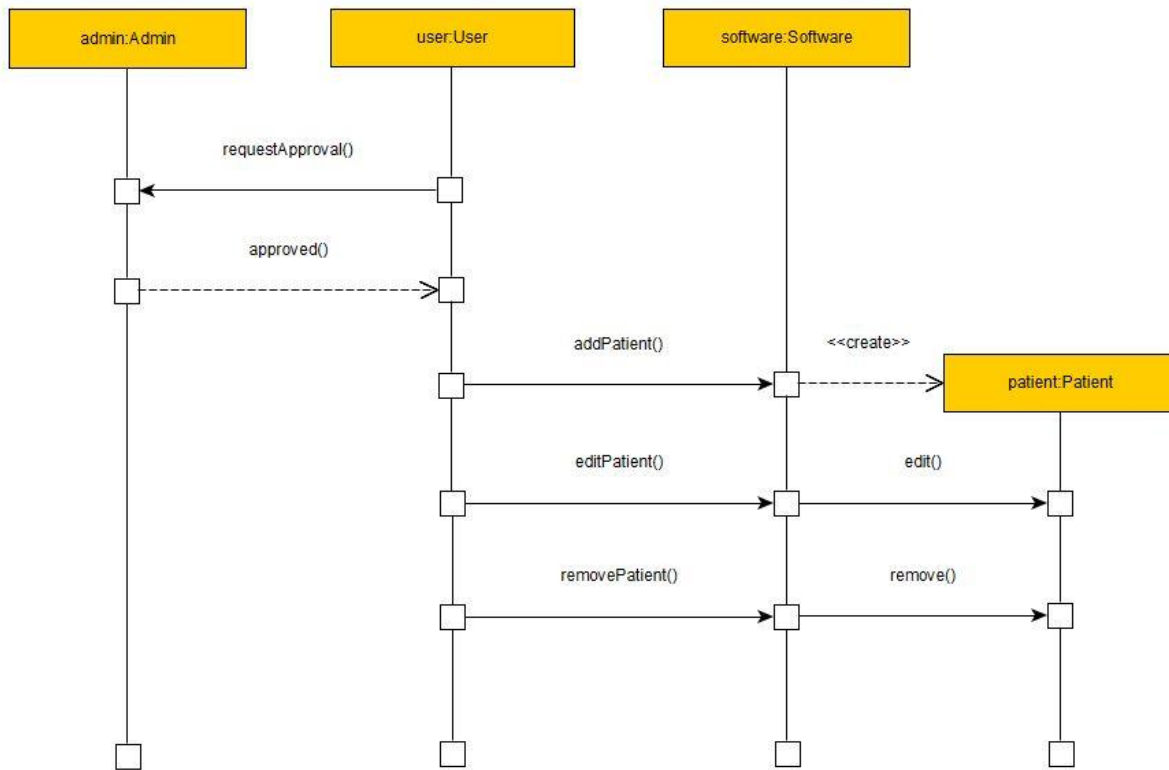


Figure 4: Sequence Diagram

10. Detailed Class Diagram

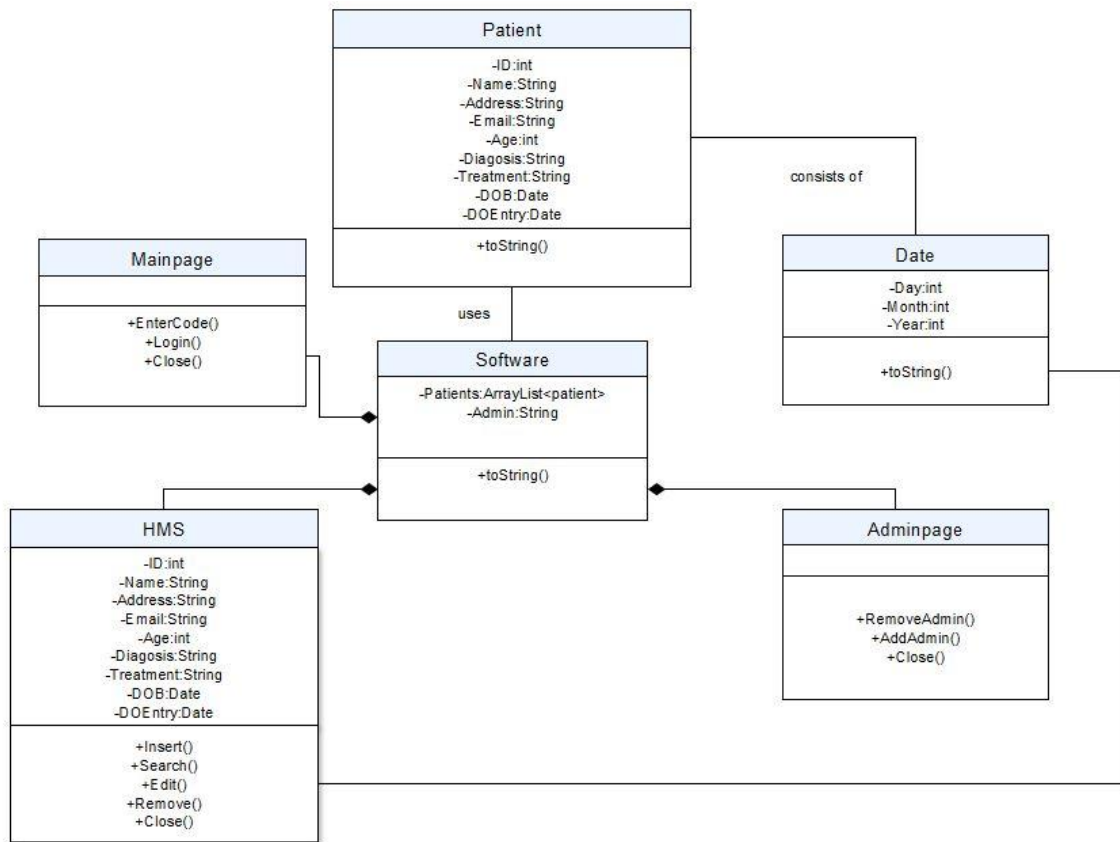


Figure 5: Detailed Class Diagram

11. User Interface Design

The user interface styles used are “Form fill-in” for simple data entry and learning ease. Natural Language is also used for its easy access to casual users.

The user interface changes when any of the following happen:

- The “user” presses the “Login” button in the “Login” window another window opens that contains the HMS.
- The “admin” presses the “Enter Program Code” button in the “Login” window, another window opens that contains the “admin” page.
- The “admin” presses “Create User” button in the “admin” page another window opens with a message indicating that a new “user” has been successfully created or that a “user” with the same name already exists.
- The “admin” presses the “Delete User” button in the “admin” page another window opens with a message indicating that the “user” has been successfully removed.
- The “user” clicks the “Add Patient” button in the HMS window another window opens with a message indicating that the “patient” has been successfully added or that the “patient” already exists in the application.
- The “user” clicks the “Remove Patient” in the HMS window button another window opens with a message indicating that the “patient” has been successfully removed.
- The “user” clicks on the “Search” button in the HMS window the “patient” data is displayed on the label if the correct data is entered.

The “user” may interrupt the state of the system if any of the following happen:

- The “user” enters wrong login data, an error message appears indicating the problem.
- The “user” enters wrong “patient” data, an error message appears indicating the problem.

15. Estimated Project Cost

Software cost components and pricing:

Hardware and Software cost: None

Travel and training cost: 500 LE for travel expenses.

Effort cost: 7000 LE.

A sum of five engineers were involved in this project. Each with a salary of 1000 LE.

2000 LE of social and insurance costs is included.

Effort cost overheads: 600 LE.

300 LE for building, heating and lighting.

200 LE for networking and communications.

100 LE for shared facilities.

Total Cost: 8100 LE

Person months were calculated using the standard formula for algorithmic models.

$$PM = A * Size^B * M$$

Where A = 2.94 in initial calibration.

Size = 1 KLOC (Kilo Line of Code)

B = 1.2

RCPX: Product Reliability and Complexity	multiplier = 1
RUSE: Product Reusability	multiplier = 1
PDIF: Platform Difficulty	multiplier = 1
PREX: Personnel Experience	multiplier = 1
PERS: Personnel Capability	multiplier = 1
SCED: Required Schedule	multiplier = 1
FCIL: Team Support Facilities	multiplier = 1

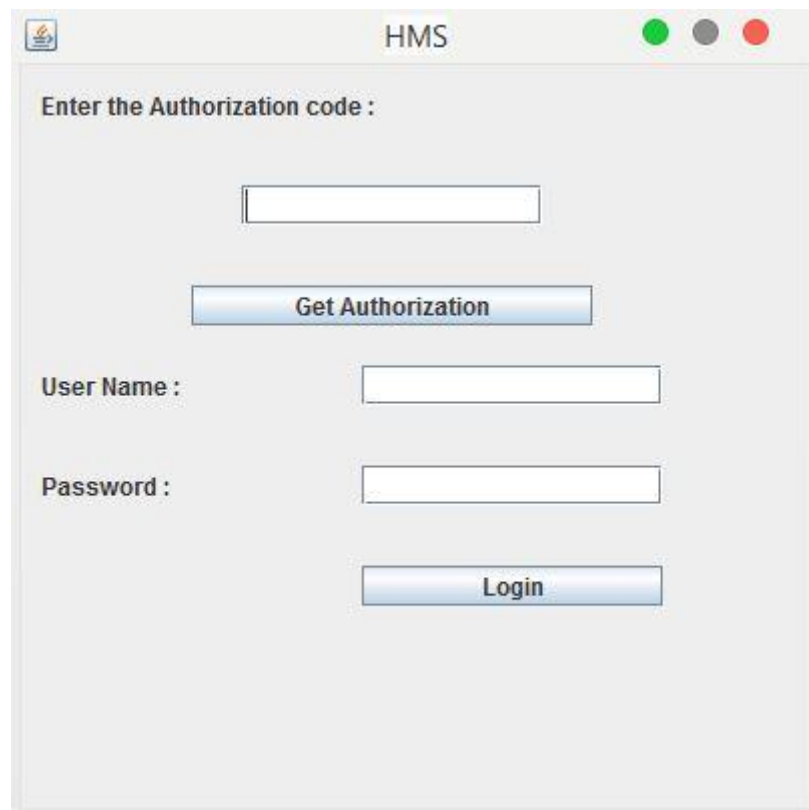
Initial COCOMO estimate without cost drivers: 3 Person-months

RCPX: Product Reliability and Complexity	high, multiplier = 1.2
RUSE: Product Reusability	high, multiplier = 1.2
PDIF: Platform Difficulty	low, multiplier = 1.1
PREX: Personnel Experience	moderate, multiplier = 1.1
PERS: Personnel Capability	high, multiplier = 1.2
SCED: Required Schedule	accelerated, multiplier = 1.2
FCIL: Team Support Facilities	moderate, multiplier = 1.1

Adjusted COCOMO estimate: 9 Person-months

16. User Guide

1) Double click on the application to open, then enter “0” in the “Enter the Authorization code” text field and press “Get Authorization” button to get to the “Admin” window.



The screenshot shows a window titled "HMS" with standard macOS window controls (green, grey, and red buttons). The main content area has a label "Enter the Authorization code :" followed by a text input field. Below this is a "Get Authorization" button. Further down are labels for "User Name :" and "Password :", each followed by a text input field. At the bottom is a "Login" button.

Figure 6: HMS window

2) Enter a number from 0-9 in the “ID” text field then write your own password. When you’re done, press “Add the admin” button then “Back to the main menu” button. If you had already added an admin and you wish to remove him/her, enter the admin’s ID then press “Remove the admin” button.

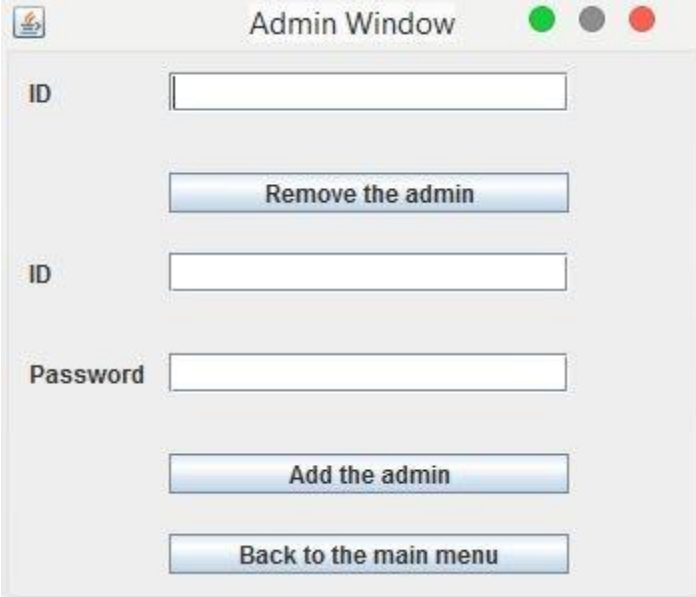
The image shows a graphical user interface window titled "Admin Window". It features a standard macOS-style title bar with a small icon on the left and three colored window control buttons (green, grey, red) on the right. The main content area is light grey and contains two sets of input fields and three buttons. The top set consists of an "ID" label followed by a text input field, a "Remove the admin" button, and another "ID" label followed by a text input field. Below these is a "Password" label followed by a text input field. At the bottom of the window are two buttons: "Add the admin" and "Back to the main menu". All buttons have a blue gradient and a slight shadow effect.

Figure 7: Admin window

3) After returning back to the main menu, enter your registered ID and password then press “Login”, the “Patients” window will appear. You can now enter the patient’s information in the proper fields then press “Insert a patient”. If you want to search/edit/remove a patient, enter the patient’s ID then click on the desired operation button.

The screenshot shows a window titled "Patients Window" with a standard macOS-style title bar (green, grey, and red buttons). The window is divided into two main sections. The left section contains a vertical list of input fields: "Name", "ID", "Age", "Address", "Email", "Diagnosis", "Treatment", "Date of Birth", "Date of Entry", and "Date of Exit". The "Date" fields are implemented as date pickers showing "1", "1", and "1901". At the bottom of this section is a button labeled "Insert a patient". The right section is titled "Enter the patient's ID:" and contains a single input field. Below this field are three buttons: "Search", "Edit", and "Remove". At the bottom of the right section is a large, empty rectangular box. At the very bottom of the window is a button labeled "Back to the main menu".

Figure 8: Patients window