



# Kalafong Provincial Tertiary Hospital

## Gynaecological Patient Information management System:

### Master Documentation

#### Team Pentec:

Ruth Ojo 12042804  
Liz Joseph 10075268  
Trevor Austin 11310856  
Maria Qumayo 29461775  
Lindelo Mapumulo 12002862



Final Version

October 29, 2015

# Contents

<b>1</b>	<b>Vision and Scope</b>	<b>4</b>
1.0.1	Project Background . . . . .	4
1.0.2	Project Vision . . . . .	5
1.0.3	Project Scope . . . . .	5
<b>2</b>	<b>Application requirements and design</b>	<b>5</b>
2.1	PIMS Login Module . . . . .	5
2.1.1	Scope . . . . .	5
2.1.2	Use cases . . . . .	6
2.2	PIMS Statistics Module . . . . .	7
2.2.1	Scope . . . . .	7
2.2.2	Use cases . . . . .	7
2.2.3	Equations . . . . .	9
2.3	PIMS Artificial Intelligence Module . . . . .	10
2.3.1	Scope . . . . .	10
2.3.2	Use cases . . . . .	10
2.3.3	Neural Network Training . . . . .	11
2.3.4	Neural Network Testing . . . . .	13
2.4	PIMS Notification Module . . . . .	14
2.4.1	Scope . . . . .	14
2.4.2	Use cases . . . . .	14
2.5	PIMS Space Module . . . . .	16
2.5.1	Scope . . . . .	16
2.5.2	Use cases . . . . .	16
<b>3</b>	<b>Architectural requirements</b>	<b>20</b>
3.1	Access and integration requirements . . . . .	20
3.1.1	Human access channels . . . . .	20
3.1.2	System access channels . . . . .	20
3.1.3	Integration channels . . . . .	20
3.2	Quality requirements . . . . .	21
3.2.1	Usability ( - priority:critical) . . . . .	21
3.2.2	Scalability ( - priority:critical) . . . . .	21
3.2.3	Reliability and Availability ( - priority:critical) . . . . .	22
3.2.4	Integrability ( - priority:critical) . . . . .	23
3.2.5	Security ( - priority:important) . . . . .	23
3.2.6	Maintainability ( - priority:important) . . . . .	24
3.2.7	Monitorability/Auditability ( - priority:important) . . . . .	25
3.2.8	Testability ( - priority:important) . . . . .	25
3.2.9	Performance ( - priority:important) . . . . .	26
3.3	Architectural responsibilities . . . . .	27
3.4	Architecture constraints . . . . .	28
3.4.1	Data exchange format . . . . .	28
3.4.2	Neural Network . . . . .	28
<b>4</b>	<b>Architectural Patterns and Styles</b>	<b>29</b>
4.1	Model-View-Controller (MVC) . . . . .	29

<b>5</b>	<b>Architectural Design</b>	<b>30</b>
5.1	Technologies	30
5.1.1	Node.js	30
5.1.2	ExpressJS HTTP Server	30
5.1.3	AngularJS	30
5.1.4	Broadway plug-in framework	30
5.1.5	Crypto	30
5.1.6	Express-Validator	31
5.1.7	Node-mailer	31
5.1.8	MongoDB Database	31
5.1.9	Mongoose Object Data Model	31
5.1.10	Database Hosting Server	31
5.1.11	Unit.JS	32
5.1.12	Mocha	32
5.1.13	Chai	32
5.1.14	Should JS	32
5.1.15	Super test	32
5.1.16	Jade Templating Engine	33
5.1.17	Heroku Dev Center Deployment Server	33
5.1.18	Operating Systems	33
5.1.19	Dependency Management	33
5.1.20	PassportJS	33
5.1.21	MeldJS AOP	33
5.2	Package Managers	34
5.2.1	NPM	34
5.2.2	Bower	34
5.2.3	Winston logging	34
5.2.4	Synaptic Neural Network	34
5.3	Tactics and Strategies Addressing Quality Requirements	35
5.3.1	Aspect Oriented programming	35
5.3.2	Contracts-driven Development	35
5.3.3	Indexing	35
5.3.4	Templating	35
5.3.5	Database Connection pools	36
5.3.6	Logging	36
5.3.7	Client-side Rendering	36
5.3.8	Asynchronous processing	36
5.3.9	Framework for UI elements	36
5.3.10	Dependency Injection	36
5.4	Development Architecture	38
5.4.1	Version control	38
5.4.2	IDE	38
5.4.3	Builds	38
5.4.4	Unit Testing	38
5.4.5	Documentation	38
5.4.6	Issues and Bugs	38

<b>6</b>	<b>TESTING INTRODUCTION</b>	<b>39</b>
6.1	Objectives . . . . .	39
6.2	Testing Strategy . . . . .	40
6.3	Scope . . . . .	40
6.4	Reference Material . . . . .	40
6.5	Definitions and Acronyms . . . . .	40
<b>7</b>	<b>TEST ITEMS</b>	<b>40</b>
7.1	Program Modules . . . . .	40
7.2	Job Control Procedures . . . . .	40
7.3	User Procedures . . . . .	40
7.4	Operator Procedures . . . . .	40
<b>8</b>	<b>FEATURES TO BE TESTED (Functional Requirements)</b>	<b>40</b>
8.1	PIMS Login . . . . .	40
8.2	PIMS Notifications . . . . .	41
8.3	PIMS Edit Profile . . . . .	43
	8.3.1 Pims Login . . . . .	43
8.4	PIMS Add User . . . . .	43
8.5	PIMS Statistics . . . . .	44

# 1 Vision and Scope

## 1.0.1 Project Background

The system should allow data to be entered by medical students, junior doctors and registrars (specialists in training) working in the department. People who enter data should have usernames and passwords (student numbers and HPCSA registration numbers can be used) and they should only be able to enter information and should not have access to data. They should be able to enter data using smartphones, tablets or personal computers in an environment where the hospital is not computerized and computers are not available in the hospital for this purpose.

For purposes of entering data smartphone and tablet applications (apple and android) should be developed to allow smartphone and tablet access to the system.

The data should be securely stored on a site on the Website of the University of Pretoria.

Data will be entered into the system by different employees working in the Department of Obstetrics & Gynaecology at the Kalafong Provincial Tertiary Hospital.

Data of patients entered should include both the patient's hospital number as well as RSA Identity number.

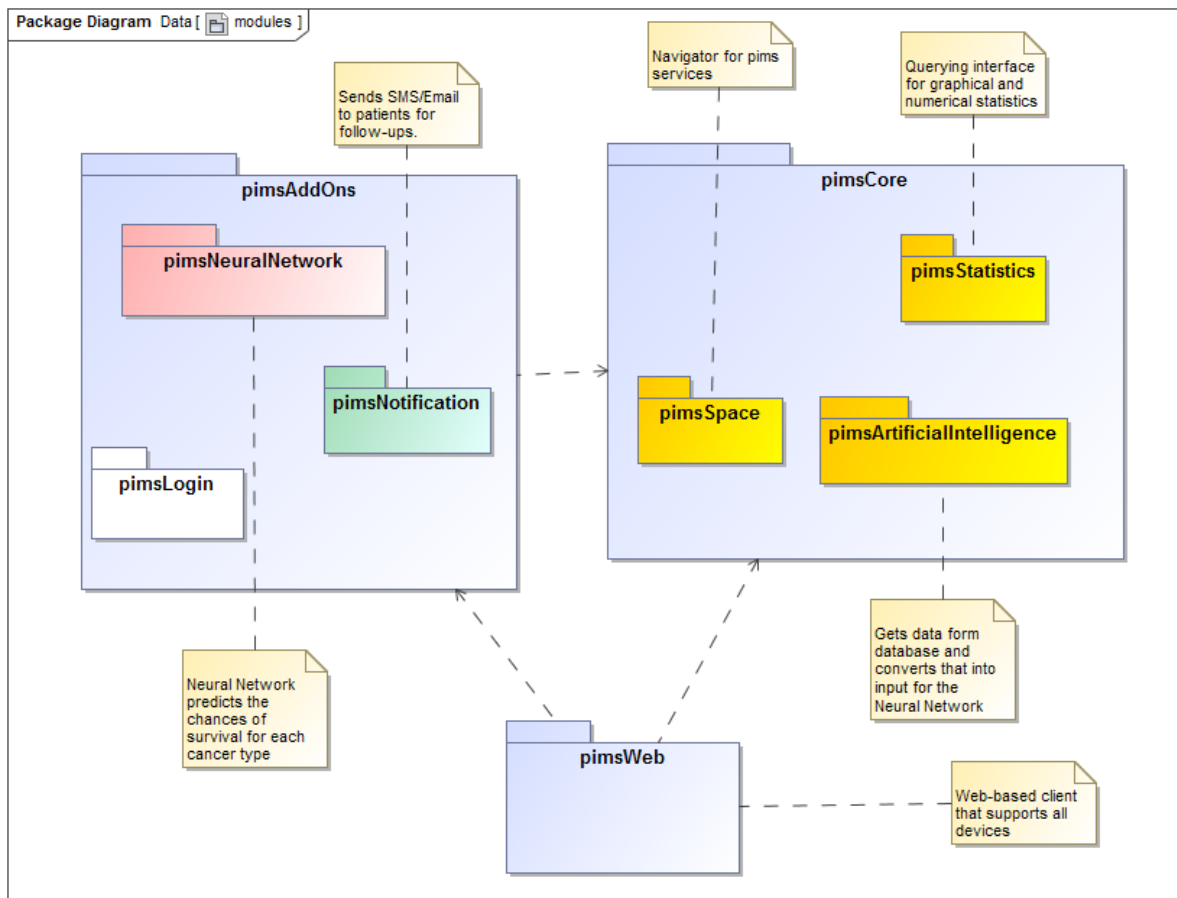
The possibility of linking this system to that of the National Health Laboratory System (NHLS) should be investigated to make it possible to access laboratory results through this system. The NHLS has an online accessibility.

The administrator of the system should have access to all relevant data. The different levels and specifications of data output will be defined upfront and the ability should exist to add or edit these specifications as required.

Patient information and data are highly confidential and the website and information should be secure. All users will have to use a username and password. Medical students can use University of Pretoria student numbers and medical interns, medical officers and registrars can use their Health Professions Council of South Africa (HPCSA) unique registration numbers. Doctors and students rotate through the department for different time periods and the usernames and passwords should expire depending on the different categories. Students rotate for four weeks, medical interns for four months and medical officers and registrars for up to five years. Administrative staff and consultants are more permanent and for security reasons should perhaps update information yearly.

## 1.0.2 Project Vision

## 1.0.3 Project Scope



# 2 Application requirements and design

## 2.1 PIMS Login Module

This module is responsible for allowing a user to login into and logout out of Pentec Patient Information Management System. A user should login with the credentials they were assigned upon registration. If the user is not assigned a user name then an exception is thrown and the user is redirected to the login page.

To avoid "Spambots", a reCAPTCHA challenge is used.

### 2.1.1 Scope

The scope is shown in the use case diagram below:

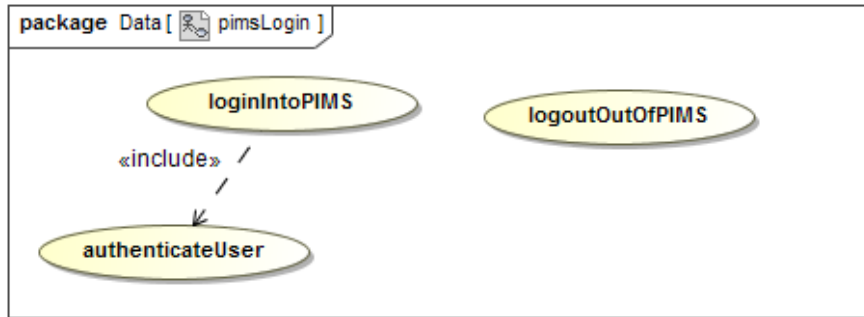


Figure 1: Login module scope

### 2.1.2 Use cases

**loginToPIMS** – [priority: critical] This use case is to cater for the logging in of user into the system. It entails the authentication of each user as well as their authorization, which is based on their access rights.

**Service Contract** The service contract for loginToPIMS is shown below.

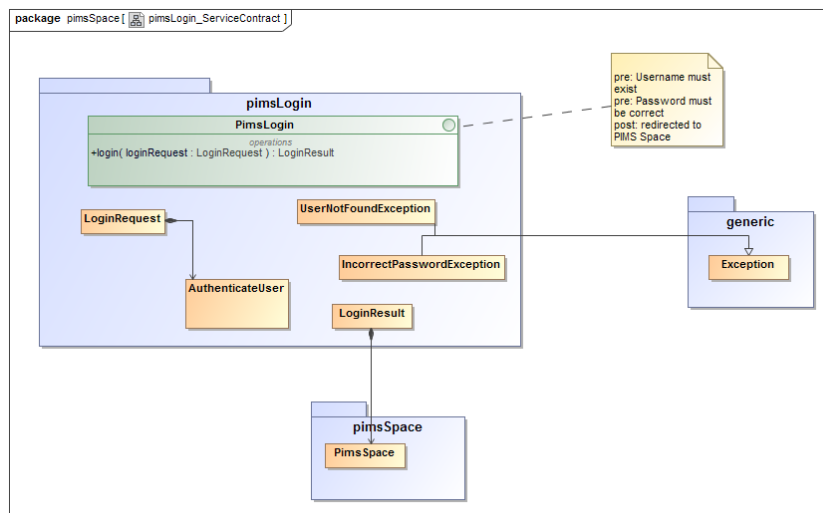


Figure 2: Service contract for pimsLogin

**logoutOfPIMS** – [priority: critical] This use case logs a user out of the system by destroying the session. The user will not be allowed access unless they log back in.

**Service Contract** The service contract for logoutOfPIMS is shown below.

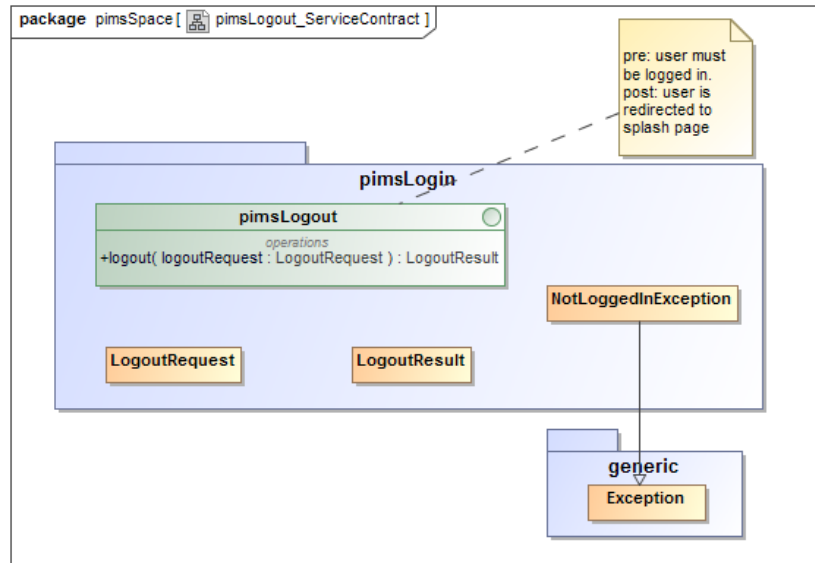


Figure 3: Service contract for logoutOfPIMS

**authenticateUser** – [priority: critical] This use case makes sure that only registered users can use PIMS. It also manages the user rights so that users only access what's intended for them. If a normal user tries to access admin pages, then they are redirected to a page that applies to their access rights.

## 2.2 PIMS Statistics Module

This module is responsible for displaying statistics from the database. The user has an option to choose the type of graph (either a bar chart or line graph).

### 2.2.1 Scope

The scope is shown in the use case diagram below:

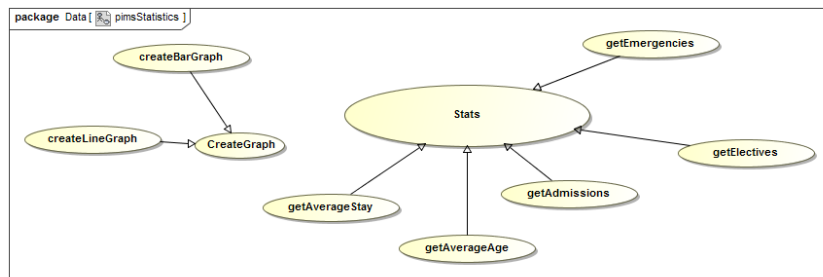


Figure 4: PIMS statistics module scope

### 2.2.2 Use cases

**viewStats** – [priority: critical] This use case is a generic version of all statistics to be obtained. Sub-use cases follow the same contract, but return different data.

**Service Contract** The generic service contract for all statistics is shown below.



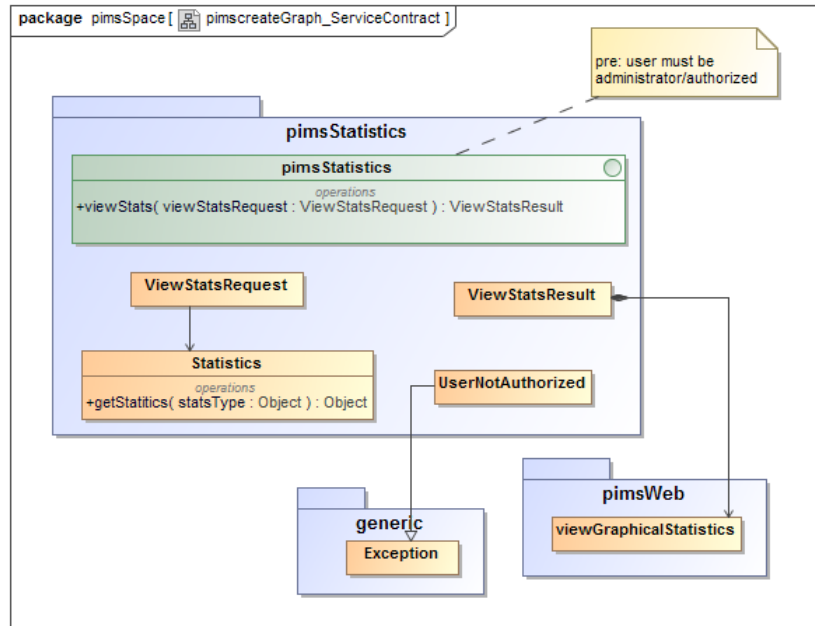


Figure 5: Service contract for viewStats

**Process Specification** The generic process specification for all statistics is shown below.

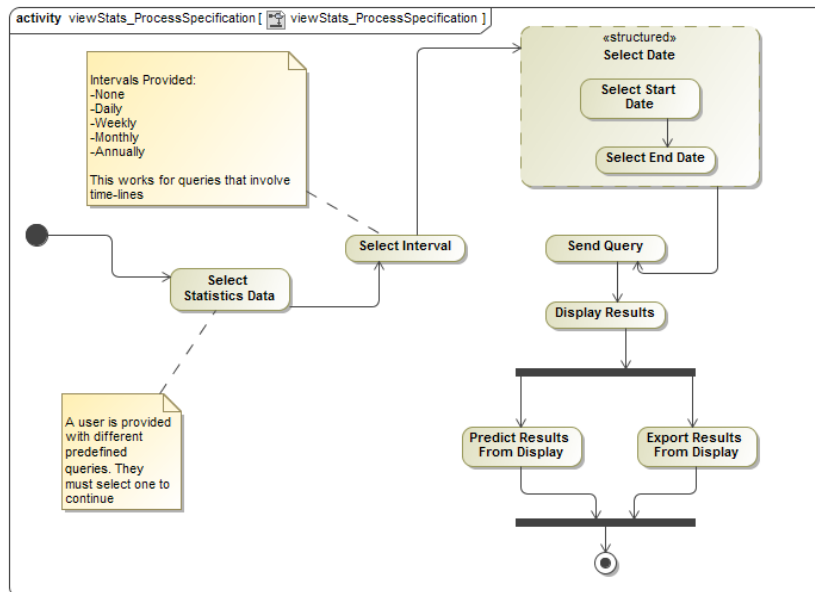


Figure 6: Process specification for viewStats

**createGraph** – [priority: critical] This use case creates and displays a graph. This graph can be viewed by the client or downloaded for record keeping.

### 2.2.3 Equations

#### Statistics Average Equation

$$AM = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^x a_j$$

#### Graph Equation

$$f(x) = \sum_{i=1}^n x_i$$

## 2.3 PIMS Artificial Intelligence Module

This module is responsible for predicting the chances of survival for each cancer type. Parameters for each patient are considered to compute this prediction. These parameters are converted into a single value that will be used by the PIMS Neural Network Module as input.

### 2.3.1 Scope

The scope is shown in the use case diagram below:

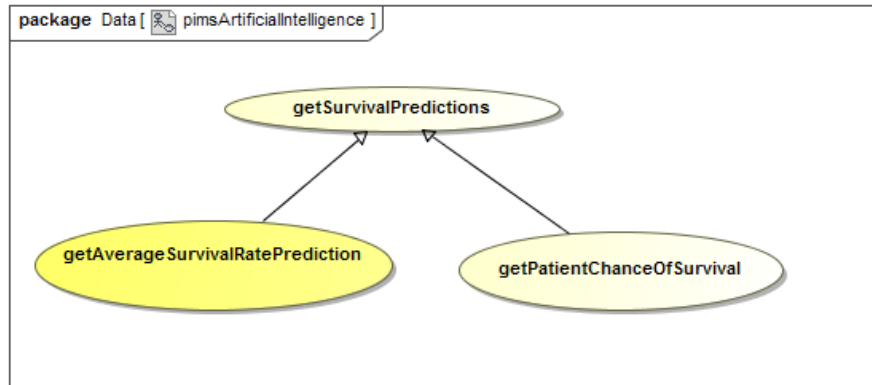


Figure 7: The global scope for PIMS Artificial Intelligence Module

### 2.3.2 Use cases

**getAverageSurvivalRatePrediction** – [priority: critical]

**Service Contract** The generic service contract

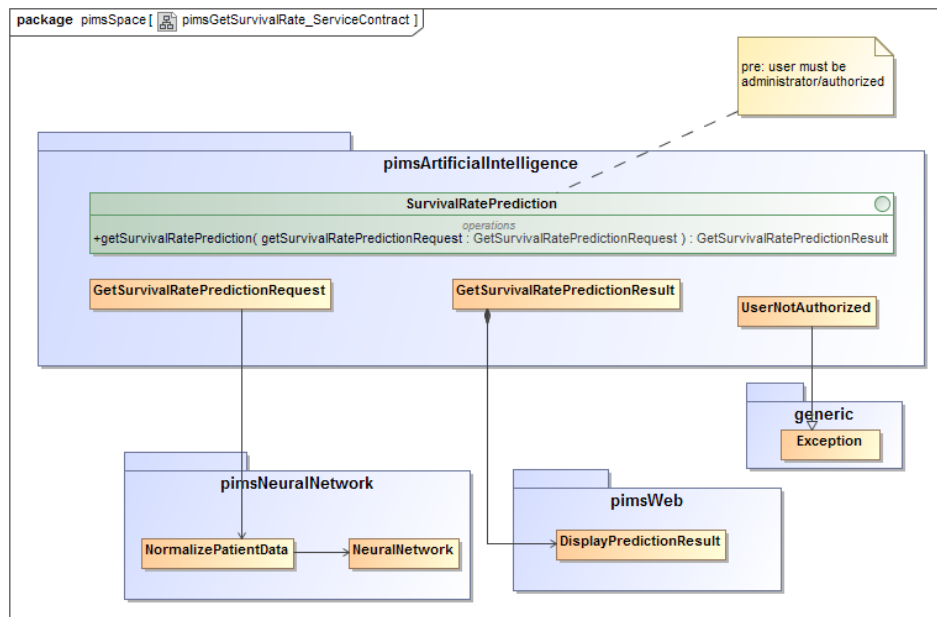


Figure 8: Service contract for getAverageSurvivalRatePrediction

getPatientChanceOfSurvival – [priority: critical]

**Service Contract** The generic service contract

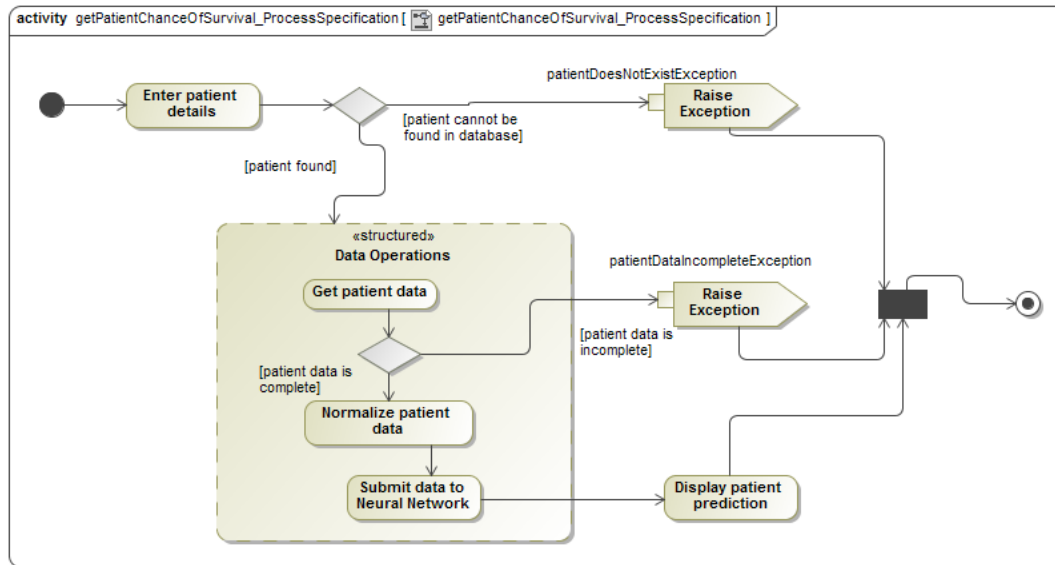


Figure 9: Process specification for Service contract for getPatientChanceOfSurvival

### 2.3.3 Neural Network Training

Patient data was analysed for possible risk factors for cancer susceptibility. These risk factors are:

- Age - No normalization was required for this field. However, decimal scaling was used for the age. Patients are expected to be within the age range of [16 - 100] so the age is scaled-down to two decimal places.
- HIV status - Is a binary field (either negative or positive). Negative HIV statuses were given the numerical equivalent of 0.9 while a positive status holds 0.1.
- CD4 count (If HIV status is positive) - If the patient is HIV negative, the numerical values becomes 0.9998. Otherwise, decimal scaling (4 places down) is used since the CD4 range is 0 - 1500.
- Figo Stage - Since this value is neither binary nor numeric, the number of figo stages determines how the patient's stage is normalized. If there are n stages then the kth stage gets the numerical equivalent of :  $\frac{k-stage}{n}$
- Site of distant metastase - Same as the Figo stage.
- Histology - Also uses the same concept as Figo stage.
- Differentiation - Also uses the same concept as Figo stage.
- Primary treatment - Also uses the same concept as Figo stage.
- Type of surgery - Also uses the same concept as Figo stage.

- Type of radiotherapy - Also uses the same concept as Figo stage.
- Response to treatment - Also uses the same concept as Figo stage.
- Relapse - Also uses the same concept as Figo stage.
- Last known vital status - Also uses the same concept as Figo stage.

The Neural Network makes use of the back-propagation algorithm for the purpose of learning.

- The equations used for the neural network are as follows:

- Input Function:

$$net = \sum_{i=1}^n x_i w_i$$

- Hidden layer Input Function:

$$net_{y_i} = \sum_{i=1}^{I+1} z_i w_j i$$

- Hidden layer Sigmoid Activation function:

$$y_j = \frac{1}{1 + e^{-net_{y_j}}}$$

- Output layer Input Function:

$$net_{o_k} = \sum_{j=1}^{I+1} y_j w_k j$$

- Output layer Sigmoid Activation function:

$$o_k = \frac{1}{1 + e^{-net_{o_k}}}$$

- Sigmoid Activation function:

$$f(net) = \frac{1}{1 + e^{-net}}$$

- Training error:

$$Error = (t_k - o_k)$$

- The equations used for the back propagation phase are as follows:

- \* Output layer error propagation:

$$\delta_{o_k} = -(t_k - O_k)(1 - O_k)O_k$$

- \* Output layer weights propagation:

$$w_{k_j} o_k += -(\delta_{o_k} y_j$$

- \* Hidden layer error propagation:

$$\delta_{y_j} = -(w_{k_j} \delta_{o_k} (1 - y_j) y_j$$

- \* Hidden layer weights propagation:

$$v_{j_i} += \delta_{y_j} z_i$$

### 2.3.4 Neural Network Testing

When testing the network, a the critical cancer parameters are normalized given a patient name. When classifying the patient as likely to survive or die form cancer the principles of a statistical probability density function,  $\chi^2$ -distribution are applied; wherein the null hypothesis is ( $H_0$ ):

- A patient is likely to survive cancer

in order to obtain a confidence interval for the survival prognosis. A 50% significance level is used for the confidence interval; such that 50% of the time, the output node value is likely to be a false positive and as for the other percentage, one can be confident in it as being accurate. The choice of the application of the  $\chi^2$ -distribution is plainly for it's simplicity and it is a well known probability density function. The choice of the confidence interval was aided by the application of artificial neural networks in survival analysis when compared with other survival analysis statistical models [?].

- The below  $\chi^2$ -distribution formulas are used:

– Probability: <sup>1</sup>

$$\chi^2 = \frac{1}{d} \sum_{k=1}^n \frac{(O_k - E_k)^2}{E_k}$$

– Confidence interval: <sup>2</sup>

$$\pm \frac{(n-1) \times s^2}{\chi_{\frac{\alpha}{2}}^2 \times (n-1)}$$

– Degrees of freedom:

$$(n-1)$$

- The  $\chi^2$ -distribution table of critical values will be used to test ( $H_0$ )

The shaded area is equal to  $\alpha$  for  $\chi^2 = \chi_{\alpha}^2$ .

<i>df</i>	$\chi_{.995}^2$	$\chi_{.990}^2$	$\chi_{.975}^2$	$\chi_{.950}^2$	$\chi_{.900}^2$	$\chi_{.100}^2$	$\chi_{.050}^2$	$\chi_{.025}^2$	$\chi_{.010}^2$	$\chi_{.005}^2$
1	0.000	0.000	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	19.812	22.362	24.736	27.688	29.819

Figure 10: Snippet of Chi-squared distribution critical value table

<sup>1</sup> $n$  is the number of patients to test,  $E$  is the target,  $O$  is the output node value,  $k$  is the individual patient

<sup>2</sup> $s$  is the mean square error

## 2.4 PIMS Notification Module

This module is responsible for sending email/SMS notifications to a patient. This email/-text message could be a reminder to a patient about follow up visits to the doctor.

### 2.4.1 Scope

The scope is shown in the use case diagram below:

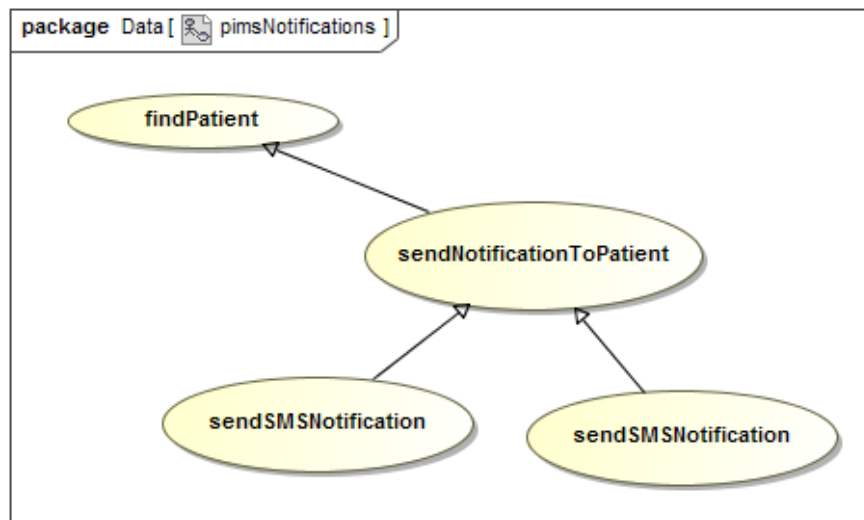


Figure 11: The global scope for PIMS Notification Module

### 2.4.2 Use cases

**findPatient** – [priority: nice-to-have] This use case is to cater for the retrieval of a patient ID/name form the database so as to obtain the contact details of the patient, if any.

**Service Contract** The service contract for findPatient is shown below.

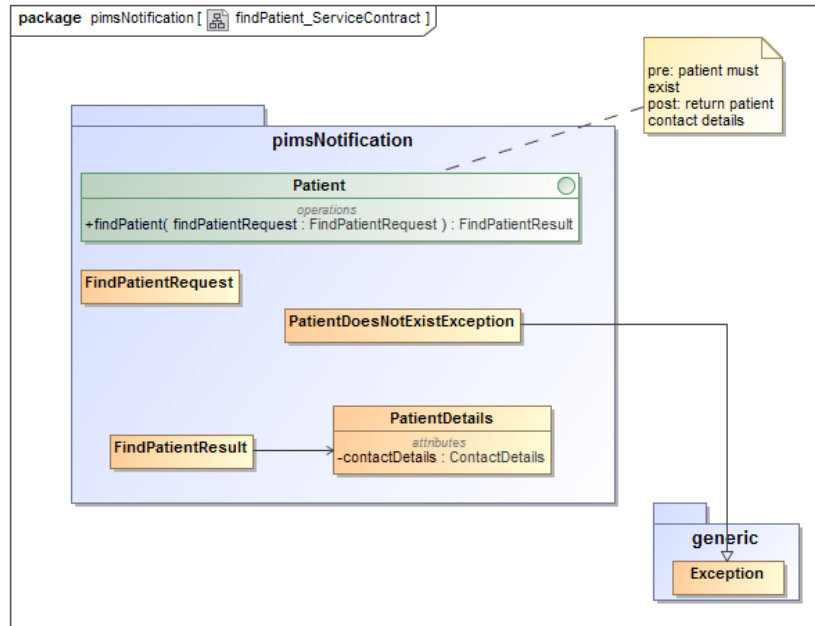


Figure 12: Service contract for findPatient use case

**sendNotification** – [priority: nice-to-have] This use case is to cater for the sending follow-up notification messages via SMS or email depending on whether or not the patient has an existing cellphone number or email.

**Service contract** The service contract for sendNotification is shown below.

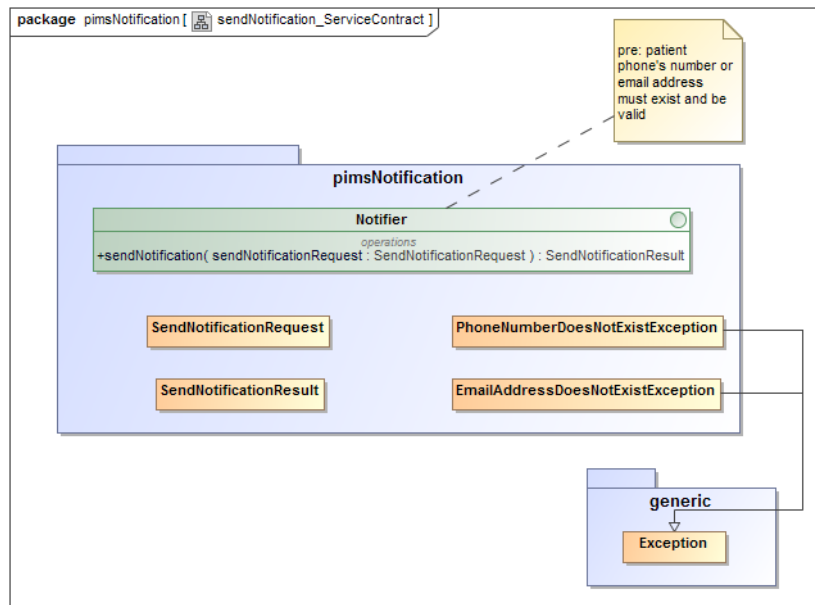


Figure 13: Service contract for sendNotification

**Process specification** The process specification for sendNotification is shown below.



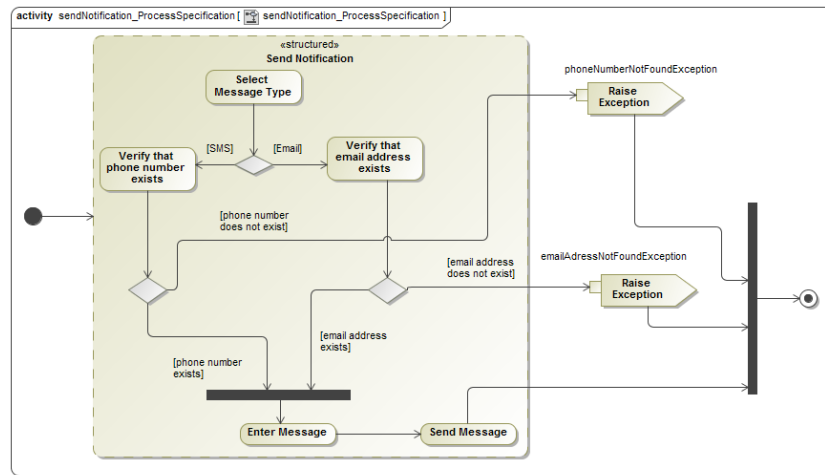


Figure 14: Process specification for sendNotification

## 2.5 PIMS Space Module

This module is responsible for providing all the core functionality of Patient Information Management System. The front-end component displays all the available services to the user.

### 2.5.1 Scope

The scope is shown in the use case diagram below:

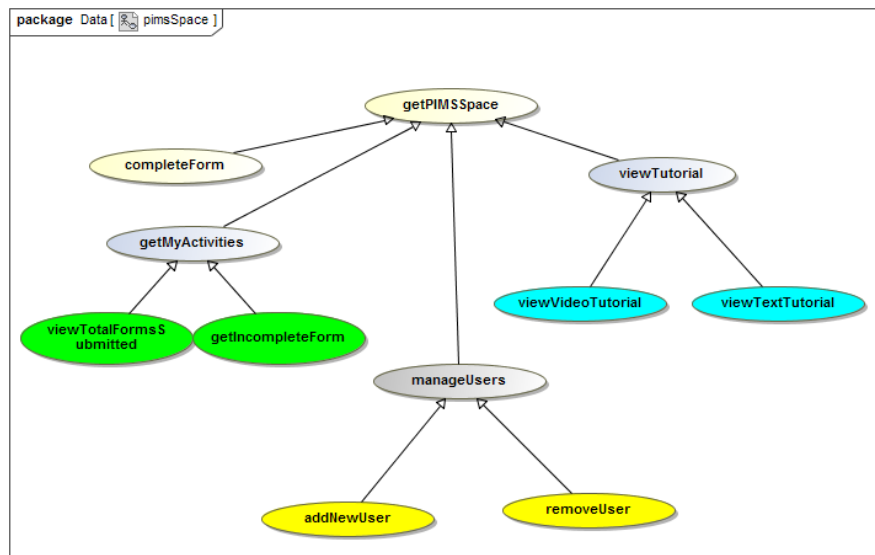


Figure 15: Scope for PIMS Space Module

### 2.5.2 Use cases

#### getPIMSSpace:

Provides the user with the front-end of the PIMS; two spaces exist given the user's privileges. If a user is an administrator, they are provided with a space

with features designed for them. The same applies for normal users, who have fewer features than the administrator.

**Service Contract** Below is a detailed service contract of how a PIMS Space is provided to the user and the actions that follow their requests.

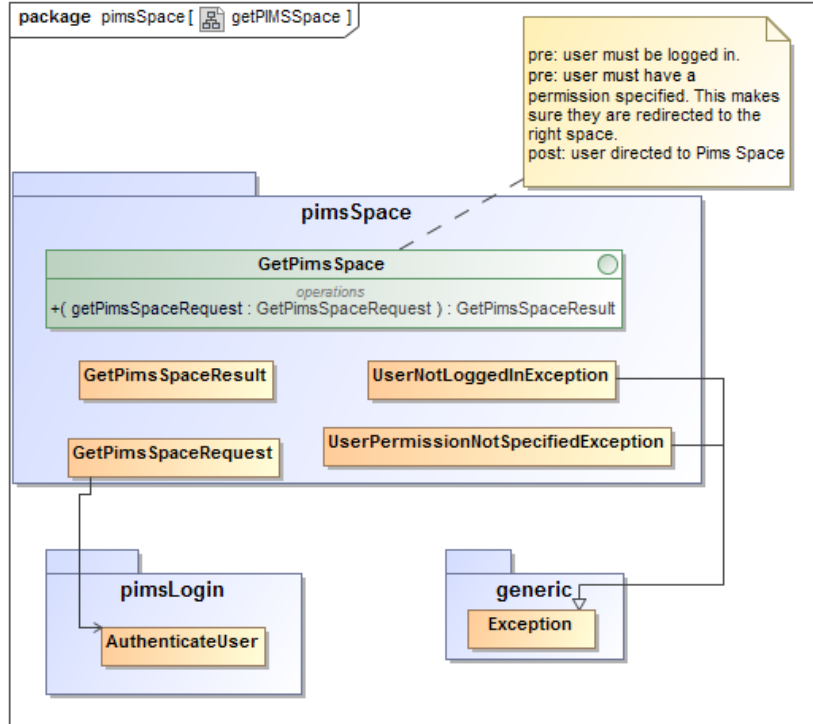


Figure 16: Service contract for getPIMSSpace

**completeAndSubmitForm:** Allows the user to complete and submit forms to the database, provided all the required fields are complete and the data type match the required ones.

**Service Contract** The service contract below further explains how completeAndSubmitForm works.

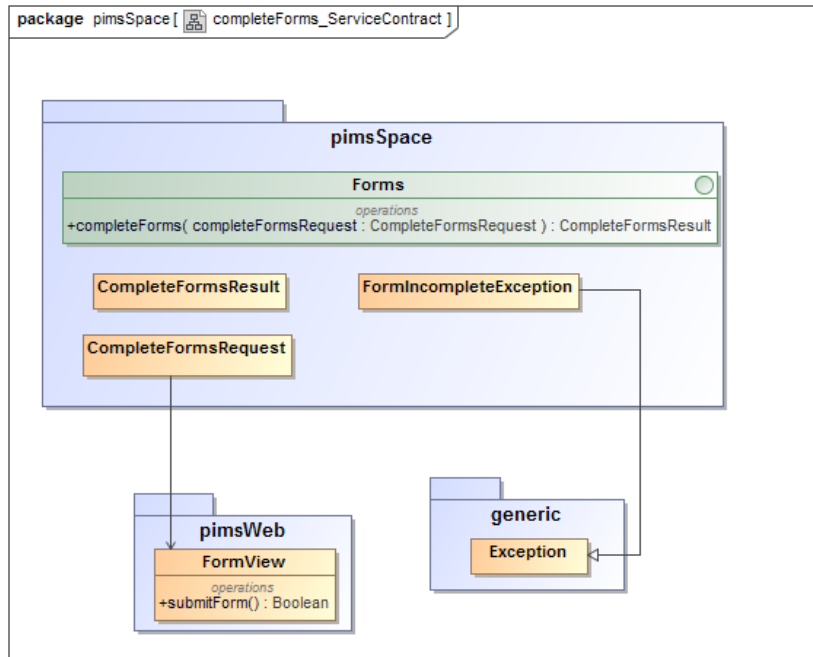


Figure 17: Service contract for completeForm

`viewTutorial`: This provides users with a different tutorials: a video tutorial that covers all the core functionality, an image tutorial with the most critical functionality, and a user-manual that provides a detailed explanation of PIMS.

`addNewUser`: This functionality is only for the administrator; normal users cannot access this page and are redirected to their respective PIMS space. This allows for the addition of new users to the system; access rights are also

**Service Contract** Below is the service contract, outlining the aforementioned.

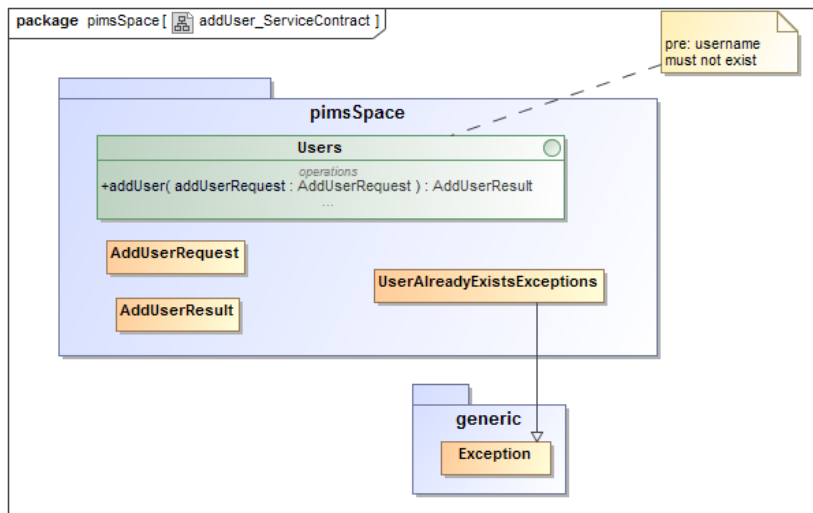


Figure 18: Service contract for addNewUser

`removeUser`: This functionality is also limited to the administrator, which means normal users cannot use this. The administrator can remove users from the system.

**Service Contract** Below is the service contract for the removeUser use-case.

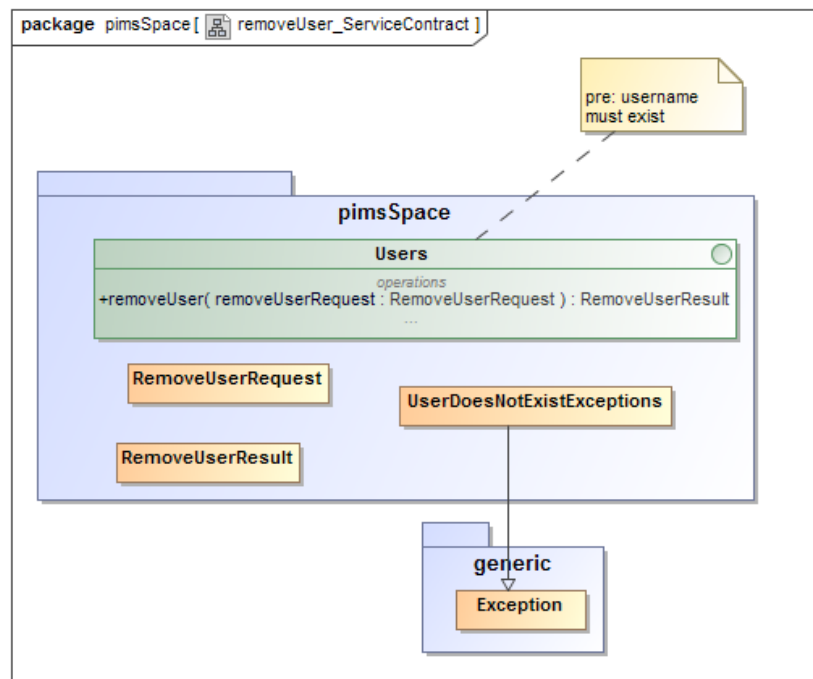


Figure 19: Service contract for removeUser

## 3 Architectural requirements

### 3.1 Access and integration requirements

#### 3.1.1 Human access channels

Human Access Channels are all the various ways a user may interact with and access the PIMS system.

1. Mobile Phone:

- This mode of access will allow for better mobility and portability, as the user can fill in information as they perform procedures.
- The user will have to use their own mobile data bandwidth to access the system, as the Hospital does not have Wi-Fi access.

2. Tablet:

- This mode of access, similar to the mobile phone, will also allow for better for mobility and portability.

3. Desktop Computer:

- This will be the least common way for the user to access the PIMS system, as the Hospital does not have a centralized computer.
- The user will have to make use of a personal modem to access the Internet as the Hospital does not have Wi-Fi access.

4. Laptop:

- This mode of access is more portable than the desktop computer.
- The user will have to make use of a personal modem to access the system, as the Hospital does not have Wi-Fi access.

#### 3.1.2 System access channels

System Access Channels are the means by which other systems will access the services offered by the PIMS.

At the moment, no system access channels are required.

#### 3.1.3 Integration channels

The PIMS will need to be integrated with:

- The NHLS website in order to retrieve diagnostic codes from the website
  - This functionality is to be executed at a later stage, outside the scope of the COS 301 Main Project.
  - The University of Pretoria website by linking the two websites
  - The existing departmental Microsoft Access database
  - The individual datasets that will be part of the system

## 3.2 Quality requirements

### 3.2.1 Usability ( - priority:critical)

#### Description

This ensures that a user will be able to use the system, with ease. The system should provide support to the user.

#### Justification

The Patient information management system is user-oriented. How the users interact with the system is critical, and this should be done with little to no effort. The system should appear easy to use and should not, at any point, baffle the users.

#### Mechanism

- A tutorial on how the patient information management system works. A user can be initiated into the system, the first time they use it. Or they can enable the tutorial until they're familiar with the functionality.
- Enable the user to troubleshoot their problems. Frequently asked questions or frequent problems could assist with this aspect. A user will be provided with predefined help options such that they will not need to contact the system's administrator, for assistance.
- Provide descriptive headings that make navigation easier. Headings should not be ambiguous. A user should know what to expect when they select a certain heading.
- Error signals should be displayed to the user, if some user-inflicted error occurs. The necessary steps to rectify this problem must be provided.
- A user should be able to undo their action, should they be aware of their mistake.
- Model-View-Controller: This separates the user interface from the rest of the system (Bass and John). A user should only interact with a simple interface that was designed for them. This is describable for patient information management system because the users don't necessary have an adept understanding of the lower levels of the system.

### 3.2.2 Scalability ( - priority:critical)

#### Description

Scalability is an essential aspect of a system and is the ability of a system to be easily enlarged in order to accommodate a growing amount of work.

## Justification

The PIMS should allow for hundreds of concurrent users, as such the system must be able to handle such a number without breaking down or reducing performance.

## Mechanism

### 1. Strategy:

- Clustering: using more resources by running many instances of the application over a cluster of servers or instances, to ensure system resources are not strained by a high workload.
- Efficient use of storage: data storage can be efficiently used through compression of the data (reducing data size to make room for more) paging (ensuring that primary storage is used only for more crucial data) as well as de-fragmentation (organizing the data into continuous fragments and free more storage space). by ensuring that no data duplications occur, storage space can be conserved, thus the load on system resources will be reduced.
- Efficient persistence: through indexing and query optimization, the amount of system power used to persist a database will be reduced, as data retrieval will be quicker and costly queries will be done without, thus also reducing system load. In addition, connections can be grouped and accessed via a central channel in order to aid persistent storage to the database.
- Load Balancing: by spreading the systems load across time or across resources the load on the system can be distributed, therefore no system resource will be heavily strained. In the case that the limit for a server has been reached, a new instance or so will have to be created in order to handle the number of increasing requests. On the other hand, if the usage of a server is way below the capacity, the number of instances will have to be reduced.
- Caching: to ensure no duplication or repeated retrieval of frequent objects or queries; a separate module can facilitate caching; thus system resources will not be used up unnecessarily.

### 3.2.3 Reliability and Availability ( - priority:critical)

#### Description

The PIMS should be accessible at almost all times, in particular during the peak operational hours of the hospital. This accessibility will be limited to the hospital network only, so as to ensure no information can be changed without approval.

#### Justification

The reliability and availability requirements are very important seeing as the information to be kept on the system is highly valuable to the medical staff, as the need up-to-date information concerning the patients they are dealing with (lives

could be at risk). With this in mind, only a downtime of less than 2 hours, at most twice a month will be allowed, so as to allow for the medical staff to maximally use the system. A high reliability rate is recommended to ensure that users do not encounter any errors and/or data corruption in their use of the system. The only leeway that will be given for errors, is to have at most one.

### **Mechanism**

- Clustering: using more resources by running many instances of the application over a cluster of servers or instances; therefore if any server should fail, the reliability and availability of the system will not be compromised.
- For reading from and writing to the database, we will ensure that no parallel updates are possible through enforcing the use of a single object to stream all database transactions; thus reducing inaccuracy that would be a result of data redundancy.
- Use of more resources: This would heavily reduce system downtime, as a temporary server can be run while the other is maintained.

#### **3.2.4 Integrability ( - priority:critical)**

##### **Description:**

The PIMS should be integrated with ease to the University of Pretoria website as well as the NHLS web site.

##### **Justification:**

The PIMS is primarily designed for the Kalafong Hospital medical staff and is required to be accessible on both the University's website and the NHLS website. For security purposes we limit its integration to just these two sites.

##### **Mechanism:**

- A contracts based decoupling strategy is used to implement integrability and extensibility on the system by providing sections and different functionality between the back end system and the host site.
- modularity: The system is primarily

#### **3.2.5 Security ( - priority:important)**

##### **Description**

This is a very important requirement for the PIMS. Patient's information should not only be confidential, but levels of user authorisation must exist. The existing information should not be modifiable by users that have no access to the information; this also applies to outside intruders.



## Justification

The whole system should not be penetrable by an intruder; the general public should not have access to any of the information about the patients. The system should also make sure that each user can only access the attributes that applies to them.

## Mechanism

- **Encryption:** User passwords and patient data are confidential and should be encrypted. The patient data will be anonymized by ensuring the data should not allude to which patient it belongs to. The patient's personal information like names will be encrypted to cater for this.
- **Authentication and Authorization:** A user's identity and access rights needs to be determined before they can access the system resources.
- **Store log information:** Although this applies to auditability 3.2.7, it helps to know the nature of a security threat. If new system threats are noted, more security features can be implemented.

### 3.2.6 Maintainability ( - priority:important)

## Description

The PIMS should be easily maintainable in future; thus making it flexible and extensible.

## Justification

Software always needs new features or bug fixes. Maintainable software is easy to extend and fix, which encourages the software's uptake and use.

## Mechanism

- **Open-source resources:** using open source resources to minimize update costs in the future.
- **Iterative development and regular reviews:** This will help us improve system quality.
- **External code review:** Here we get external people, preferably those more experienced, to review the implementation, to ensure that it is clean and has loose coupling. The cleaner and less coupled the implementation, the better it is to maintain.
- **Version control:** This will help keep our code, tests and documentation up to date and synchronised. It will also help us keep track of progress.
- **Documentation** Relevant documentation will help future developers understand the software and system as a whole.

- **Service Contracts:** 5.3.2 Strict adherence to contract specifications will aid system extensibility.

### **3.2.7 Monitorability/Auditability ( - priority:important)**

#### **Description**

Information logs will be kept on each system request and response as well as error messages, which will allow for one to monitor the system and check how it is operating as a whole, as one will have easy access to the system logs to know what aspects of the system are responding well and which aspects are not.

#### **Justification**

- System monitorability is important, for the fact that the system will have multiple concurrent users and to prevent system failure or incorrect operations, the system must be monitored at regular intervals.

#### **Mechanism**

The audit logs will be in the following format:

- log entry ID
- userID of user requesting a service
- date and time of request
- the service that was requested
- request and response of service

The audit log will be accessible via a *.log* file on the server.

### **3.2.8 Testability ( - priority:important)**

#### **Description**

Testability is a measure of how well system or components allow you to create test criteria and execute tests to determine if the criteria (pre and post conditions) are met. Testability allows faults in a system to be isolated in a timely and effective manner.

#### **Justification**

It is extremely important to conduct test cases for every component that will be integrated into PIMS. This ensures consistency in the system, and enables faults and/or loopholes to be picked up and resolved in good time.

## Mechanism

- Unit testing: Unit testing and integration testing will be conducted using mocha and unit.js
- White-box tests internal structures or workings of an application. This requires the explicit knowledge of the internal workings of the PIMS.
- Layering: Simplify testability since high level issues will be separated from low level issues. This level of granularity makes the system to be easily testable on every layer separately.
- Model View Controller: The PIMS model will be separate from the view and the controller, so that it is simpler to have separate test criteria testing different cases. This separation simplifies the development cycle since the model, view or the controller could be tested independently.

### 3.2.9 Performance ( - priority:important)

#### Description

This requirement pertains to how well the system responds to some action(s) execution within a set time interval. This is measured by system latency (time taken to respond to some event(s)) or throughput (number of events executed within a given amount of time).

#### Justification

Performance is an important requirement, as the lack of it will influence other system quality requirements. For instance, if the system does not respond in a timely manner it would affect system responsiveness and usability 3.2.1. The PIMS system will be used by multiple medical staff at a time, as such it needs to respond to user events with minimal latency.

## Mechanism

- High hardware and software tolerance: the system must continue to operate, even if a server lags or one software component is not working. The lagging of a server may result in performance at a reduced level but the system must still be operational with some level of throughput.
- By using fewer computationally expensive hashing algorithms for data encryption, the effect of security measures on performance will be lessened.
- The system should give user feedback if a background process is operational. For instance if an AJAX POST request is being sent in the background, the user must be given a system busy indicator so that they know their request is still being processed.
- Caching of database objects: The system must cache any frequently used database objects so that any user tasks that require database processing will respond quicker. This is particularly important when there is an increase in database requests.

### **3.3 Architectural responsibilities**

The architectural responsibilities for the PIMS include:

1. Logging in and out of users of the system
2. User authentication
3. Appropriate authorization for access of resources
4. form generation through JSON Schemas
5. statistics calculation and graph generation
6. a web and mobile access channel
7. the persisting and accessing of patient information
8. sending notification messages by email or SMS
9. integration with University of Pretoria website
10. logging of all operations on the website
11. cancer survival rate predictions through a feed-forward neural network

### **3.4 Architecture constraints**

- The architecture is largely constrained to node.js and MongoDB.
- Due to the nature of node.js' single-threaded processing there may be an impact on performance, but this will be balanced out with the use of other performance-driven technologies, such as MongoDB.

#### **3.4.1 Data exchange format**

- The transfer of data between the server and client side as well as from the database will primarily be in JSON format, particularly due to the nature of the MongoDB database data format.

#### **3.4.2 Neural Network**

- The operation of the feed-forward neural network is to allow for background calculations that must be propagated to the user interface in a user-friendly and eye-pleasing manner. As a result, the neural network must have not only a back-end for calculation but a front-end aspect as well. An npm Neural Network package that works in browsers is used to address this aspect.

## 4 Architectural Patterns and Styles

### 4.1 Model-View-Controller (MVC)

Model-View-Controller is an architectural pattern that divides software applications into three interconnected parts.

The controller is responsible for translating requests in to responses (Nadel, 2012). The controller 'inserts' the response back into the view, which could be html or it's equivalents.

The view translates the response, from the controller, into a visual format for the client (Nadel, 2012). The view also sends requests to the controller.

The model's task is to maintain state and give methods for changing this state. Typically, this layer can be decomposed to:

- Service layer - provides high-level logic for dependent parts of an application.
- Data access layer - services objects invoke this layer, which provides provides access to the persistence layer.
- Value object layer - provides data-oriented representations of terminal nodes in the model hierarchy.

## **5 Architectural Design**

### **5.1 Technologies**

#### **5.1.1 Node.js**

The system will be deployed in a Node.js environment. Node.js allows for queued inputs and since the system revolves around multiple inputs possibly going through at any particular time this provides a huge advantage over most other systems. Node.js also includes a wide range of modules through NPM (Node package manager). Some examples of these include ExpressJS, AngularJS, MongoDB, mongoose, and many more. Node.js processes programs asynchronously and thus is a no-interrupt driven language. This is an advantage because, as stated above, it allows for queued inputs. Node.js is written in JavaScript which means it is ubiquitous which is an obvious advantage.

Node.js relies heavily on callbacks to allow for synchronization which is an obvious hindrance for programmers that are not strong with recursion or callbacks. Although this con is outweighed by the advantages of the system.

#### **5.1.2 ExpressJS HTTP Server**

Express is a lightweight, high performance HTTP framework which supports URL configurable routing. This framework will lend to a more professional look of the system as well as enforce system maintainability and flexibility.

#### **5.1.3 AngularJS**

Angular is front-end development framework that enforces the MVC architecture pattern. Angular speeds up client-side templating, a feature that is essential for the PIMS System. In addition, the structure of AngularJS allows for dependency injection; thus allowing for simpler implementation of unit tests. The use of this framework will aid system usability, maintainability as well as performance.

#### **5.1.4 Broadway plug-in framework**

The PIMS is largely a modular system as such, this Dependency Injection (tactic: 5.3.10) framework will allow for a more flexible and maintainable system, as modules can be plugged in as needed. In addition, it will allow any future developers who may wish to extend the system to add plug-ins that they deem necessary or to add to the existing modules.

#### **5.1.5 Crypto**

Crypto is an npm module that allows for multiple forms of encryption. It is the default hashing algorithm for Node.js and is fast enough, so as to not affect performance. For the sake of password security (requirement: 3.2.5), the PIMS system

will make use of the Crypto's PBKDF2 encryption. This form of encryption uses HMAC-SHA1 to derive a key of some fixed length from the password, salt and iterations. It is slow and somewhat computationally expensive which is why it will only be used for password encryption.

#### **5.1.6 Express-Validator**

This npm package is an Express server middle-ware for the npm validator module. In the PIMS system, it is used for the purposes of form validation, particularly when making a post request to the server. This technology aids Usability, as it allows for the application to give appropriate feedback to a user concerning any input made.

#### **5.1.7 Node-mailer**

This npm package is for the notifications component of the system, allowing for the sending of email notifications to patients for the purpose of follow-up visits with a medical practitioner.

#### **5.1.8 MongoDB Database**

- MongoDB is a NoSQL document store database and is used particularly for applications that need to store large volumes of data, which is mandatory for the PIMS system.
- MongoDB ensures that scalability, a critical requirement, is enforced. This is due to its highly cache-able persistence environment which allows for better system performance. Also, because the system will require multiple database access tasks, the use of the MongoDB database is very helpful, as it allows for multiple read access and a single write operation.

#### **5.1.9 Mongoose Object Data Model**

- Mongoose is an object modelling environment for MongoDB and Node.js. It enforces service contracts and structure via validation, it also provides connection pooling, which improves system scalability. Automatic object to document mappings is also supported.

#### **5.1.10 Database Hosting Server**

- The PIMS MongoDB database will be hosted on MongoLab. It has the capacity to create multiple databases which would aid the system requirement of reliability and availability. It also offers database security using two-factor authentication to access the database.



#### **5.1.11 Unit.JS**

- Unit.JS supports dependency injection (tactic: 5.3.10), a valuable tactic the implementation of the PIMS system.

#### **5.1.12 Mocha**

Mocha was selected as our primary unit testing tool for Node.js and the browser, to assist in the simplification of asynchronous testing.

- Mocha tests run serially, allowing for flexible and accurate reporting, whilst mapping uncaught exceptions to the correct test cases.
- Mocha offers browser support which proves useful for our project, as we are testing our application across different browsers.
- Mocha offers a JavaScript API for running tests which assists in identifying errors easily and makes the framework much easier to understand and use.
- Mocha is also very popular as a unit testing tool; there is an online community of users offering assistance and suggestions.
- It is also very configurable and assists in describing test suites.

#### **5.1.13 Chai**

- We selected Chai as it is an assertion and expecting library for Node.js and the browser and given that we are creating a web application and using JavaScript as our testing framework; Chai pairs them both very well.
- Chai also works well with Mocha and other unit testing frameworks, as we had decided to use more than one to achieve readability for the output of our unit testing program.
- Chai assists in performing various assertions against our JavaScript code.

#### **5.1.14 Should JS**

- Should is another assertion library that makes use of functions that are natural language-based and are thus simple and intuitive to use and implement. Its expressive nature helps us keep our unit test code clean and it adds semantics to error messages.

#### **5.1.15 Super test**

- Super test is a unit testing tool that will enable the testing of the ExpressJS HTTP server, particularly the url endpoints. It will enable the testing of web pages and the type of pages they should render given some action.

#### **5.1.16 Jade Templating Engine**

The template engine Jade is simple to use and has a readable layout. It allows for minimal code and speeds up the entire process of writing web pages. Jade's readable layout helps improve system maintainability.

#### **5.1.17 Heroku Dev Center Deployment Server**

- Heroku is a cloud based application platform and it offers an easy to use deployment service and it easily integrates with our version control system Github. As a result, when system implementation is pushed to the master branch it is automatically updated on the Heroku Server.

#### **5.1.18 Operating Systems**

- Windows
- Possibly Android OS

#### **5.1.19 Dependency Management**

- **NPM**
  - \* This will be used to build the project and ensure that any system component is not IDE-specific
  - \* The details pertaining to each of the packages are within a package.json file
- **Bower**
  - \* This will help to maintain client-side dependencies, particularly the components that will be used for the front-end
  - \* The details pertaining to each of the packages are within a bower.json file

#### **5.1.20 PassportJS**

This is a widely used Node.js authentication module and it offers a wide range of authentication strategies. The PIMS makes use of the PassportJS Local Strategy for user authentication. This strategy allows for one to use their own authentication measures, making it flexible to use. PassportJS also provides session management which is a very useful feature for the PIMS; wherein multiple users will be accessing various system resources at the same time.

#### **5.1.21 MeldJS AOP**

Meld is a widely used AOP library for JavaScript. It allows for one to add to or change the behaviour of functions in a non-invasive manner. It will address the Aspect-Oriented Programming tactic by supporting the following functionality:

- Logging (strategy: 5.3.6)
- System traceability

## **5.2 Package Managers**

### **5.2.1 NPM**

We use npm for our server-side dependencies

### **5.2.2 Bower**

We use bower to manage our client-side dependencies, in particular Angularjs, as this package manager helps to make it easier to obtain and manage the many Angularjs components needed for our Neural Network front-end.

### **5.2.3 Winston logging**

Winston is a widely used multi-transport async logging library for Node.js which will allow for all user operations, server requests as well as errors to be logged to a file stored on the server. The purpose of logging (strategy: 5.3.6) is to ensure system auditability (strategy: 3.2.7). The logging functionality will be injected through aspects so as to not have the logging functions inter-woven through the code; thereby decoupling the logging functionality from the core system functionality. The use of aspects will make the logging functionality flexible and maintainable.

### **5.2.4 Synaptic Neural Network**

Synaptic is a technology neutral, JavaScript Neural Network library that offers services for different types of neural networks with different training algorithms. The services offered are simple to apply and non-invasive. A user will be allowed to see the results of a neural network in the front-end, as this library works in the browser.

## 5.3 Tactics and Strategies Addressing Quality Requirements

### 5.3.1 Aspect Oriented programming

- AOP is an interception mechanism that will allow for the addition of new services to the system. This tactic will aid the following system functionalities:
  - \* Logging (tactic: 5.3.6)
  - \* System traceability
- It will also address the following system requirements:
  - \* **Auditability** (requirement: 3.2.7) - logging functionality can be seamlessly injected across system services
  - \* **Maintainability** (requirement: 3.2.6) - additional services can be added to the system
  - \* **Integrability** (requirement: 3.2.4) - AOP can make integration of other system components easier, particularly if those components would have to be inter-woven through the code.

### 5.3.2 Contracts-driven Development

Service contracts were specified prior to the commencement of system implementation. The different pre and post-conditions pertaining to each functional requirement as well as the data structure constraints are enforced across the system services. The use of contracts addresses the following quality requirements:

- **Testability** (requirement: 3.2.8) - Having specified the pre and post-conditions for each use case, any violation of a pre-condition will give a failing test and will throw an exception. Also, any failure to fulfill a post-condition will show a failure in a service of the system.
- **Maintainability/Flexibility** (requirement: 3.2.6) - any system component that is replaced by another must be able to satisfy the same contract as the previous component

### 5.3.3 Indexing

Indexing of database objects will allow for faster retrieval of these objects; thus improving system performance as well as scalability.

### 5.3.4 Templating

Node.js offers many templating engines and these allow for:

- A consistent user interface which improves system maintainability (requirement: 3.2.6) and usability (requirement: 3.2.1).

### **5.3.5 Database Connection pools**

Connection pooling will be used to achieve better system scalability (requirement: 3.2.2) and performance (requirement: 3.2.9)

### **5.3.6 Logging**

System logging will capture audit data and will help achieve system auditability (requirement: 3.2.7), security (requirement: 3.2.5) as well as maintainability (requirement: 3.2.6). The use of aspects will lend to the achievement of these requirements, as it will make the logging functionality flexible and maintainable.

### **5.3.7 Client-side Rendering**

By rendering pages on the client-side (in the browser), load is taken off the server to render pages and this helps to improve system performance (requirement: 3.2.9) and scalability (requirement: 3.2.2), as well as usability given that web pages will load quicker.

### **5.3.8 Asynchronous processing**

Asynchronous processing will be achieved through delaying tasks which may involve lengthy operations or tasks that have to wait for resources into the background to process. This will achieve better system scalability (requirement: 3.2.2), performance (requirement: 3.2.9) and responsiveness, as other operations will not be blocked or held up.

### **5.3.9 Framework for UI elements**

By applying a dynamic Javascript UI component library, for all the user interface elements, the following will be achieved:

- A dynamic user interface; thereby achieving system usability (requirement :3.2.1)
- System maintainability (requirement : 3.2.6), as components will be re-used
- System scalability 3.2.2 and performance 3.2.9, as these libraries will off-load system rendering to the client-side.

### **5.3.10 Dependency Injection**

Dependency Injection will help in achieving the following:

- Maintainability/flexibility (requirement :3.2.6) - different system components can be replaced easily
- Testability (requirement :3.2.8) - dependency injection aids unit testing, as mock objects can be injected allowing for a system component to be tested as a stand-alone module.

- System components can be seamlessly applied in different environments.

## 5.4 Development Architecture

### 5.4.1 Version control

The version control system that will be used is git. The final and reviewed implementation of the system will reside in the master branch, whilst the combined, prototype of the implementation will be in the Develop branch, where the system will be tested before being placed in the master branch. Any new feature, testing or bug fix is developed in a new branch where it is tested before being merged into the trunk. These new branches are created from the Develop branch.

### 5.4.2 IDE

The choice of an IDE is not limited to a specific one. All the project builds are IDE independent and are driven solely by npm.

### 5.4.3 Builds

NPM is used to build the project components. The package.json file specifies all the server-side dependency properties to build the project. The *sub modules* of the project are kept in private git repositories and can be obtained through the Pentec Github Organisation.

### 5.4.4 Unit Testing

Unit testing will be done using Unit.JS.

### 5.4.5 Documentation

All code is annotated with JSDoc documentation metadata which generates and hosts the API documentation centrally. This quickens system development, as the documentation generation is automatic and it improves system maintainability (requirement: 3.2.6).

### 5.4.6 Issues and Bugs

Any issues found with the system implementation can be raised using Github's issue tracker.

## 6 TESTING INTRODUCTION

### 6.1 Objectives

The following document contains the detailed outline of the PIMS testing process and results. Below are the main modules that were tested.

- User Login
- PIMS Artificial Intelligence
- PIMS Statistics
- PIMS Notifications
- PIMS Space

Testing was done on the system mainly for the following 5 reasons:

- To ensure that the system meets both functional and non functional requirements according to the given specifications
- System stress, that is to make sure that the system does not fail with multiple users or any other factors because it is expensive to resolve and fix at a later stage.
- To handle and resolve System failures and bugs appropriately and in good time.
- To point out the defects and errors that were made during and after the development phases.
- To ensure that the final product is of top, professional, software engineering standards.



## 6.2 Testing Strategy

## 6.3 Scope

## 6.4 Reference Material

## 6.5 Definitions and Acronyms

# 7 TEST ITEMS

## 7.1 Program Modules

## 7.2 Job Control Procedures

## 7.3 User Procedures

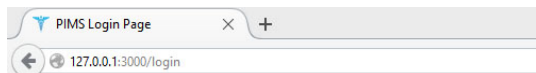
## 7.4 Operator Procedures

# 8 FEATURES TO BE TESTED (Functional Requirements)

## 8.1 PIMS Login

It is a priority that a user logs into the system for security reasons to be able to interact with it. When testing the login use case we tested for the user's authentication and rights. We tested whether or not he has admin or normal user rights.

The local login page



Successful login unit tests

```
login user
✓ authenticate should login user
✓ authenticate should retrieve username
✓ authenticate should retrieve password
✓ authenticate should fail with empty username
✓ authenticate should fail with empty password
✓ authenticate should fail with empty username and empty password
✓ authenticate should return a boolean
✓ checkAdmin should return a boolean
```

## Conditions

The following pre and post conditions are defined for adding a new user.

### Pre conditions

- User must not be logged in
- User must have valid credentials, a valid username and password.
- User credentials must be found in the Mongo database.

## Post conditions

- User successfully logged in according to his user rights

Code snippets for user login use case, with test data ?a? . Testing that a valid username is entered

```
it("authenticate should retrieve username", function(done){
  login.authenticate("a", "g", function(err){
    User.findOne({username: "a", password: "g"}, function(found){
      found.username.should.equal("a");
    });
  });
  done();
});
```

Code snippets for user login use case, with test data ?g? . Testing that a valid password is entered

```
it("authenticate should retrieve password", function(done){
  login.authenticate("a", "g", function(err){
    User.findOne({username: "a", password: "g"}, function(found){
      found.password.should.equal("g");
    });
  });
  done();
});
```

Code snippets for user login use case. Testing that the user is logged in after valid authentication

```
describe("login user", function(){
  it("authenticate should login user", function(done){
    login.authenticate("a", "g", function(err){
      User.findOne({username: "a", password: "g"}, function(found){
        found.username.should.equal("a");
        found.password.should.equal("g");
      });
    });
  });
  done();
});
```

## Remark

Login ensures security and access control. Testing regarding the logging in all pass, thus the PIMS system is secure.

## 8.2 PIMS Notifications

Send notification is a feature that is not in the requirement documentation but was asked to be added in by our client. It allows him to send notifications to remind patience of their next follow up. The send notification use case unit tests failed because the schema

could not be found, see figure bellow.

```
send notification to patient
1> findPatient should retrieve email address
2> findPatient should return email address
✓ should pass
3> should pass
```

Test failure causes

```
1> send notification to patient findPatient should retrieve email address:
MissingSchemaError: Schema hasn't been registered for model "patients".
    at Context.<anonymous> (C:\Users\Waliko\Documents\GitHub\Pentec_PIMS\test\
est.login.js:113:21)

2> send notification to patient findPatient should return email address:
MissingSchemaError: Schema hasn't been registered for model "patients".
    at Context.<anonymous> (C:\Users\Waliko\Documents\GitHub\Pentec_PIMS\test\
est.login.js:124:21)

3> send notification to patient should pass:
Error: the string "don't pass" was thrown, throw an Error :>
```

The following pre and post conditions for the send notification must hold true.

## Conditions

The following pre and post conditions are defined for adding a new user.

### Pre conditions

- User must be logged in as administrator.
- Patient must exist in the system database.
- Patient must have an active email account.

### Post conditions

- A notification informing patient about their next appointment is sent to the users email account.

Code snippets for send notification use case that looks for a patient email address in the database to send a notification to.

```
it("findPatient should return email address", function(done){
  notification.findPatient("sue", function(found){
    found.should.equal("nodemailingtest@gmail.com");
  });
  done();
});
```

## Remark

Since Send notification fails unit testing, the module needs to be revised and tested again.

## 8.3 PIMS Edit Profile

### 8.3.1 Pims Login

Pims edit user profile should be able to allow the admin user to update his profile and edit his information accordingly.

The Code bellow demonstrates the update of the user name after retrieving it.

```
describe("update profile", function() {
  it("should retrieve username", function(done) {
    User.findOne({username: "Leon"}, function(err, contact) {
      should.not.exist(err);
      contact.username.should.equal("Leon");
    });
    done();
  });

  it("should modify profile details [surname]", function(done) {
    User.findOne({username: "Leon"}, function(err, contact) {
      contact.surname = "Snymanss"
      should.not.exist(err);
      contact.surname.should.equal("Snymanss");
    });
  });
});
```

Edit profile was tested for the following conditions

- Retrieve data
- Modify profile details

The figure bellow depicts the successful testing of the UpdateAuthentication and check-Admin functions.

```
update profile
✓ should retrieve username
✓ should modify profile details [surname]
✓ should modify profile details [password]
✓ should modify profile details [email]
✓ should modify profile details [user_rights]
```

## 8.4 PIMS Add User

Adding a user, is a feature only available for the admin user and is a crucial use-case needed for saving a new user's to the system.

The add user case passed unit testing successfully as seen in the figure bellow

```
add new user
✓ should save user details in database
```

## Conditions

The following pre and post conditions are defined for adding a new user.

### Pre conditions

- User must be logged in as admin.
- User to be added must not already exist in the database.
- User must be a medical personnel.

### Post conditions

- New user is added and can interact with the system.

The code bellow shows the testing for adding a new user with sample data

```
describe("login user", function(){
  it("authenticate should login user", function(done){
    login.authenticate("a", "g", function(err){
      User.findOne({username: "a", password: "g"}, function(found){
        found.username.should.equal("a");
        found.password.should.equal("g");
      });
    });
  });
  done();
});
```

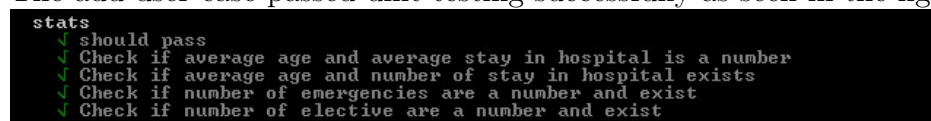
### Remark

Add User unit test successfully passes.

## 8.5 PIMS Statistics

PIMS statistics is another feature for admin user only. For the statistics use case we tested for four different cases. All passed unit test and succeeded..

The add user case passed unit testing successfully as seen in the figure bellow



```
stats
✓ should pass
✓ Check if average age and average stay in hospital is a number
✓ Check if average age and number of stay in hospital exists
✓ Check if number of emergencies are a number and exist
✓ Check if number of elective are a number and exist
```

## Conditions

The following conditions had to be met for the statistics tests to pass.

### Pre conditions

- User must be logged in as admin.
- Object type has to be a digit.

## Post conditions

- Statistical results are returned.

Unit test code to validate the number of elective procedures is indeed a number.

```
it("Check if number of elective are a number and exist", function(done){
  GS.aggregate(
    [{ $match : { "typeOfProcedure.Elective": true, "ProcedureDate": { '$gte': new Date("2014-01-02"), '$lte': new Date("2014-01-2") } } }], function(err, myResult)
    {
      should.exist(myResult);
      var num = myResult.length;
      for (var i = 0; i < num; i++) {
        var newElement = {};
        newElement['date'] = new Date(myResult[i].ourDate).toString('dd-MM-yyyy');
        newElement['close'] = myResult[i].count;
        arr.push(newElement);
      }
      var resBody = { myStatsArr: arr; };
      console.log(resBody);
      should.exist(resBody);
      res.json(resBody);
      console.log("POST response sent.");
    }
  );
  done();
});
```

Unit test codes to validate the average age and stay in hospitals are indeed numbers.

```
- it("Check if average age and number of stay in hospital exists", function(done){
-   AD.aggregate(
-     { $group: {
-       "_id": 1,
-       avgAge : { $avg: "$Age" } }
-     }, function(err, avg)
-     { if (err){ throw err;
-       res.redirect('stats');
-       }
-       else{
-         AD.aggregate(
-           { $group: {
-             "_id": 1,
-             avgStay : { $avg: "$TotalNumberOfDaysHospital" }
-           } }, function(err, avgStay)
-           { if (err){
-             throw err;
-             res.redirect('stats');
-             }
-             else{
-               var average = JSON.stringify(avg[0].avgAge);
-               var averageStay = JSON.stringify(avgStay[0].avgStay);
-               should.exist(average);
-               should.exist(averageStay);
-             }
-           }
-         );
-       }
-     }
-   );
-   done();
- });
```

## Remark

Statistics is the heaviest module the PIMS. More unit testing will be done to ensure accuracy, reliability and currency. For now all tests succeeded and passed.