Gynaecological Patient Information management System:

Architectural Requirements

Team Pentec:

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

Final Version

September 24, 2015

Contents

1 Architectural requirements

1.1 Access and integration requirements

1.1.1 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.

1.1.2 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.

1.1.3 Integration channels

The PIMS will need to be integrated with:

- The NHLS website in order to retrieve diagnostic codes form 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

1.2 Quality requirements

1.2.1 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.

1.2.2 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.

1.2.3 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 too 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.

1.2.4 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 it's 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 id primarily

1.2.5 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 1.2.7, it helps to know the nature of a security threat. If new system threats are noted, more security features can be implemented.

1.2.6 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:** 3.2.2 Strict adherence to contract specifications will aid system extensibility.

1.2.7 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.

1.2.8 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.

1.2.9 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 1.2.1. 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 increses in database requests.

1.3 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

1.4 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.

1.4.1 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.

1.4.2 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.

2 Architectural Patterns and Styles

With Node.js being a dynamic, prototype-based language, the GoF Patterns do work as well with it compared to object oriented languages. Also, Node.js is non-blocking and as such the following patterns are either Javascript specific or work well with prototype-based languages. Some of the patterns discussed here do have some aspects of the GoF patterns.

2.1 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.

2.2 Module Pattern

The Module Pattern pulls from some major principles of object-oriented programming, namely encapsulation and abstraction [3]. It enforces data encapsulation by constraining a module to one file and it provides the client of a module with only the public API methods of the module. This avoids JavaScript's public/private variable scoping. For abstraction, the module pattern provides a layer of abstraction for the client. As a result, whichever client that uses a module does not have to concern themselves with the process of instantiating the module. An interface is then defined between the client and the module and all the client has to be concerned with is knowing the parameters of the module constructor as well as the api of the returned instance.

3 Architectural Design

3.1 Technologies

3.1.1 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 as 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.

3.1.2 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.

3.1.3 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.

3.1.4 Broadway plug-in framework

The PIMS is largely a modular system as such, this Dependency Injection (tactic: 3.2.10) 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.

3.1.5 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: 1.2.5), 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 form the password, salt and iterations. It is slow and somewhat computationally expensive which is why it will only be used for password encryption.

3.1.6 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.

3.1.7 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.

3.1.8 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.

3.1.9 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.

3.1.10 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.

3.1.11 Unit.JS

- Unit.JS supports dependency injection (tactic: 3.2.10), a valuable tactic the implementation of the PIMS system.

3.1.12 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.

3.1.13 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.

3.1.14 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.

3.1.15 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.

3.1.16 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 improves system maintainability.

3.1.17 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.

3.1.18 Operating Systems

- Windows
- Possibly Android OS

3.1.19 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

3.1.20 PassportJS

This is a widely used Node.js authentication module and it offers a wide range of authentication strategies. The PIMS makes use of 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.

3.1.21 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: 3.2.6)
- System traceability

3.1.22 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: 3.2.6) is to ensure system auditability (strategy: 1.2.7). 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 form the core system functionality. The use of aspects will make the logging functionality flexible and maintainable.

3.1.23 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.

3.2 Tactics and Strategies Addressing Quality Requirements

3.2.1 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: 3.2.6)
  * System traceability

- It will also address the following system requirements:
  * **Auditability** (requirement: 1.2.7) - logging functionality can be seamlessly injected across system services
  * **Maintainability** (requirement: 1.2.6) - additional services can be added to the system
  * **Integrability** (requirement: 1.2.4) - AOP can make integration of other system components easier, particularly if those components would have to be inter-woven through the code.

3.2.2 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: 1.2.8) - 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 a the system.
- **Maintainability/Flexibility** (requirement: 1.2.6) - any system component that is replaced by another must be able to satisfy the same contract as the previous component

3.2.3 Indexing

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

3.2.4 Templating

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

- A consistent user interface which improves system maintainability (requirement: 1.2.6) and usability (requirement: 1.2.1).

3.2.5 Database Connection pools

Connection pooling will be used to achieve better system scalability (requirement: 1.2.2) and performance (requirement: 1.2.9)

3.2.6 Logging

System logging will capture audit data and will help achieve system auditability (requirement: 1.2.7), security (requirement: 1.2.5) as well as maintainability (requirement: 1.2.6). The use of aspects will lend to the achievement of these requirements, as it will make the logging functionality flexible and maintainable.

3.2.7 Client-side Rendering

By rendering pages on the client-side (in the browser), load is taken off the server to render pages and this is helps to improve system performance (requirement: 1.2.9) and scalability (requirement: 1.2.2), as well as usability given that web pages will load quicker.

3.2.8 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: 1.2.2), performance(requirement: 1.2.9) and responsiveness, as other operations will not be blocked or held up.

3.2.9 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 :1.2.1)
- System maintainability (requirement : 1.2.6), as components will be re-used
- System scalability 1.2.2 and performance 1.2.9, as these libraries will off-load system rendering to the client-side.

3.2.10 Dependency Injection

Dependency Injection will help in achieving the following:

- Maintainability/flexibility (requirement :1.2.6) - different system components can be replaced easily
- Testability (requirement :1.2.8) - 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.

3.3 Development Architecture

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 Unit Testing

Unit testing will be done using Unit.JS.

3.3.5 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: 1.2.6).

3.3.6 Issues and Bugs

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

References

[1] Ben Cherry. Javascript module pattern: In-depth, 2010.

[2] Phani Pasupula. Mongodb/mongoose connect best practices, 2014.

[3] David Sevcik. Design patterns in node.js:, 2014.