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

July 30, 2015

Contents

1 Critical Architectural Requirements

1.1 Security

1.1.1 Description

This is the most critical requirement for the patient information management system. 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.

1.1.2 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.

1.1.3 Mechanism

1. Strategy:

    - Encryption: Patient data is confidential and should be encrypted. Not every patient has to be encrypted, but the data should not imply to which patient it belongs to. The patient's personal information could be encrypted instead of all the attributes.
    - Authorization: A user's access level and identity needs to be determined before they can access the system.
    - Store log information: Although this applies to audibility, it helps to know the nature of a security threat. If new threats are noted, more security features can be implemented.

1.2 Usability

1.2.1 Description

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

1.2.2 Justification

Patient information management system is user-oriented. How the users interact with the system is a 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.

1.2.3 Mechanism

1. Strategy:

    - 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.

2. Architectural Pattern(s):

    - 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.

1.3 Scalability

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.

1.4 Reliability and Availability

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

1. Strategy:

    - 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.

=======

Description:

The Buzz system should be integrated with ease on any host site.

Justification:

The Buzz system is primarily designed for the CS website, but if in future a need to host it on another site arise,it needs to be easily integrated to the the site without any extra cost and the need to change the implementation of the system.

Mechanism:

1. Strategy:
   The contracts based decoupling strategy can be used to implement integrability and achieve extensibility on the system by providing an interface between the system and the host site which will improves decoupling on the host site and the system.

2. Architectural Pattern:
   Messaging can be used to enhance decoupling and increase flexibility, the Buzz system or host site can change without affecting the other party, both parties will act as independent component that use an interface between them to request services and share resources.

2 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.

3 Tactics and Strategies

3.1 Scalability and Performance

- Optimize repeated processes.
- Reuse resources and results.
- Reduce contention by replicating frequently used resources.
- Clustering.
- Efficient use of storage.
- Load Balancing.
- Caching.
- Use REST (Representational State Transfer) to make BuzzSpace into a scalable web service.

3.2 Security

- Authenticate users by requesting username and password when interaction with the system begins.
- Authorize users checking if a user has the rights to access and modify either data or services.
- Encryption to maintain confidentiality of data.
- Input Validation to detect malicious attacks.
- Auditing and logging for identifying and recovering from attacks.

3.3 Usability

- Usability is enhanced by giving the user feedback as to what the system is doing.
- Descriptive Error messages must be provide along with the necessary steps to address the errors.
- System must respond to actions performed by the user.
- Separate the user interface from the rest of the application using Model-View-Controller.

3.4 Integrability

- Use modular programming to modularize the system.
- Use REST (Representational State Transfer) to decouple BuzzSpace from other software that may need its services.

3.5 Maintainability

- Use modular programming to modularize the system.
- Use object-oriented programming to sub divide the sub-system features.

3.6 Monitor-ability

- Monitor-ability can be enhanced by using fault detection tactics:
    1. Ping/echo: One component issues a ping and expects to receive back an echo, within a predefined time, from the component under scrutiny.
    2. Heartbeat: One component emits a message periodically and another component listens for it. If the heartbeat fails, the originating component is assumed to have failed and a fault correction component is notified.
    3. Exceptions: These are raised when an anomaly in a component occurs, encounter an exception when a fault is detected.

3.7 Reliability

- Reliability can be enhanced by using fault recovery and preventions tactics:

    1. Active redundancy: All redundant components respond to events concurrently. All redundant components will have the same state. When a fault occurs in the responding component, the system will switch to the next redundant component, minimizing downtime.
    2. Checkpoint/rollback: the states of the components will be recorded periodically or in response to certain events. When a fault occurs, the system should be restored to the previously consistent state or checkpoint.
    3. Transactions: The system should bundle several sequential steps in such a way that the entire bundle can be undone at once.

3.8 Testability

- Enhanced testability by recording the information that enters the system and using it as input into the test harness, and recording the output of the system components.
- Separating the interface from the implementation to enable substitution of implementations for various testing purposes.
- Creating a specialized testing interfaces to capture variable values to a system component and also seeing the output of the component in order to detect faults.
- The components can maintain useful information regarding its execution internally and then be viewed in the testing interface.

3.9 Platform

3.9.1 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 Express, AngularJS, mongo, 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 obvous hinderance for programmers that are not strong with recursion or callbacks. Although this con is outweighed by the advantages of the system.

3.10 Frameworks

- Express
- AngularJS

3.11 Operating System

- Linux

3.12 Databases

3.12.1 Database(s)

- MongoDB - is a NoSQL document store. MongoDB is used especially for applications that need store large volumes of data, which is mandatory for PIMS.
  MongoDB ensures that scalability, a critical requirement, is enforced. This is due to the highly cache-able persistence environment. Concurrency is also handled via the locking mechanisms. Multiple read access and a single write operation is allowed.

3.13 Object Data Model

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

3.14 Languages

3.14.1 Programming Languages

- JavaScript
- Java

3.14.2 Mark-up Languages

- HTML

/subsubsectionTemplate Engine

- Jade

3.15 Application Servers

- GlassFish Server
- Tomcat

3.16 Dependency Management

- Apache Maven

3.17 Web Services

- SOAP-based

3.18 APIs

- Java Persistence API
- JAX-RS RESTful web services
- JAX-WS web service endpoints
- Java Persistence API entities
- The Java Database Connectivity API (JDBC)
- The Java Persistence API
- The Java EE Connector Architecture
- The Java Transaction API (JTA)

3.19 Others

- AJAX
- Servlets
- Java Server Pages
- JavaServer Faces
- JavaServer Faces Facelets
- Enterprise JavaBeans (enterprise bean) components
- Java EE managed beans

4 Recommended Technologies

4.1 Databases

4.1.1 Object Relational Databases

- Postgresql

4.1.2 NoSQL Databases

- Neo4j
- MongoDB

4.2 Object Data Mappers

To cater for the use of NoSQL Databases

- Hibernate OGM