

Pharmacy Error Tracker

Architecture Notebook

There is guidance within this template that appears in a style named InfoBlue. This style has a hidden font attribute that allows you to toggle whether it is visible or hidden in this template. Use the Microsoft® Word® menu **Tools > Options > View > Hidden Text** check box to toggle this setting. There is also an option for printing: **Tools > Options > Print**.

# Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

# Architectural goals and philosophy

The goal of the architecture is to provide a default, easy to use system that will enable a Hospital Pharmacy to track errors that occur in the dispensing of medicines, such as directions, dosage/strength, form (e.g. intravenous vs per oral), batch number, expiry date, medication (i.e. wrong medication), wrong patient, wrong quantity, etc. The system shall be customizable by the user to suit their exact reporting needs. Currently, errors are recorded on a piece of paper and later transferred to an Excel Spreadsheet on a single machine. The current process offers limited access and reportability to the user.

In terms of deployment concerns, due to the possibility of being unable to deploy software on a client’s hardware directly, the system will need to be based on the cloud, and access would be provided by a webpage. This has added benefits in that the system will not need to adapt to legacy systems. The architecture needs to be robust and able to safely store data without significant loss. The architecture needs to be flexible to allow the user to customize it to meet their needs. And the system will need to be responsive while operating from a cloud server.

# Assumptions and dependencies

The most critical area is the ability to make a web form and link the output to a backend database. This will require dependencies on a few key software resources, those being the Vue.js HTML/Javascript framework, the MySQL database software, and the Metabase data visualization software. All of these software resources are free to use for the purposes of the project.

The skill and experience of the team has been determined to be lacking in a few key areas. This is a risk that is being mitigated by the use and sharing of learning resources among team members.

# Architecturally significant requirements

[Link to System Wide Functional Requirements](https://bitbucket.org/itc303teampharmacon/pharmacy_app/src/5c9c58f2234bc16fd91c14597c59e356ebe677af/documents/System-wide%20Function%20Requirements.docx?at=%2318Docs)

[Link to Critical Core Use Cases and Candidate Architecture](https://bitbucket.org/itc303teampharmacon/pharmacy_app/src/5c9c58f2234bc16fd91c14597c59e356ebe677af/documents/Critical%20Core%20Use%20Case%20%26%20Architecture%20Candidate%20Memo.docx?at=%2318Docs)

# Decisions, constraints, and justifications

* Decided to develop a webpage app rather than a dedicated app – This approach better fits the skillsets of the team members and also better fits the project’s requirements.
* Constrained to using a cloud server for delivery – Certain clients, such as those working at hospital pharmacies may not be able to install programs on their hardware for security reasons making running the service locally impossible.
* Decided to allow the client to disable fields in the web form. Different clients may have different requirements regarding what detail level of data to collect from their users, and will likely have better knowledge of these requirements than our team members. By allowing the client to hide fields that they do not need or want data from, they can reduce the length of the form and make it easier and faster for their users to input errors.

# Architectural Mechanisms

## Architectural Mechanism 1 - Persistence

Services to handle the reading and writing of stored data. This is data about the users of the system, as well as data pertaining to the subject matter (i.e. pharmacy error records) This data will be stored in a database run by MySQL.  
Attributes:

* **Granularity:** Unknown, not more than a few kilobytes.
* **Volume:** No upper bound, not more than 10000 objects initially
* **Duration:** Indefinitely
* **Retrieval mechanism:** User data will be retrieved by matching a name and password hash. Error data will not be uniquely retrieved, only retrieved in bulk using SQL queries via Metabase.
* **Update frequency:** Objects may be updated by the user.
* **Reliability**: The objects need to survive a system crash.

## Architectural Mechanism 2 – Flexibility

Provides services to enable the user with the correct authority to hide fields in the form to improve ease of use. The customization will require the database to be able to accept null values in those fields.

## Architectural Mechanism 3 – Security

Provides services to protect access to certain resources or information. The database will need to have security mechanisms to protect user data, especially login passwords.

## Architectural Mechanism 4 – Availability

The percentage of time that the system must be available for use, including planned outages.

## Architectural Mechanism 5 – Communication

A mechanism for handling inter-process communication. The web page system will need to communicate with the database system, and vice versa.

* **Latency:** Ideally within a few seconds
* **Synchronicity:** Can be asynchronous.
* **Size of message:** Less than 1mb at most
* **Protocol:** Javascript and SQL.

## Architectural Mechanism 6 – Graphics

Supports user interface services, in this case rendering graphs and visualizations of data. We will use Metabase to handle this.

## Architectural Mechanism 7 – Online Help

Provides online help capability. Just an online manual will be required.

# Architectural views

We will be using the following views to describe the architecture:

* **Logical:** Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.
* **Use case:** A list or diagram of the use cases that contain architecturally significant requirements.