Frank Albright

SE 577, Spring Quarter 2024

College of Computing and Informatics

Drexel University

Course Project

Deliverable 2

Just Frank Software Releases Its New Healthcare Management Tool

All-In-One Tool To Change How We See the Healthcare Industry

ARCHITECTURE DECISION RECORDS

1. Healthcare tool software architecture

Context and Problem Statement

There are multiple software architectures that could provide a solution for the JFS Healthcare tool.

Considered Options

Many architectures were narrowed down to two based on the proposed healthcare tool's premise of connecting data gathered from multiple doctors, pharmacists, insurance companies, and even caretakers. These interactions represent multiple independent services, ideal for a Microservices or Service-Oriented Architecture.

Decision

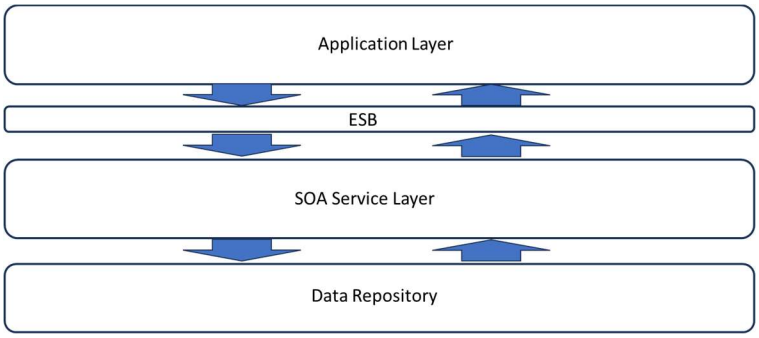
The proposal is use a Service-Oriented Architecture (SOA) after considering many tradeoffs. However, the main factor leading to this decision was the healthcare tool does not require data independence. In fact, the data will have to be accessed routinely by a single service.

Consequences

The choice for SOA means:

- the value of a data repository being accessed in a single layer is valued higher than data independence,
- inter-service messaging is accomplished through a centralized service bus (ESB), which increases complexity and reduces maintainability,
- incremental code changes will require more expenditure (time and materials),
- the ESB is a single point of failure.

Supporting Architecture Diagrams



2. Client access

Context and Problem Statement

Multiple clients need to access the JFS healthcare tool which will likely be using various styles of their own software tools.

Considered Options

The following options are being considered:

- Indirect client access through developed applications
- Direct access to external systems utilizing COTS channels

Many architectures were narrowed down to two based on the proposed healthcare tool's premise of connecting data gathered from multiple doctors, pharmacists, insurance companies, and even caretakers. These interactions represent multiple independent services, ideal for a Microservices or Service-Oriented Architecture.

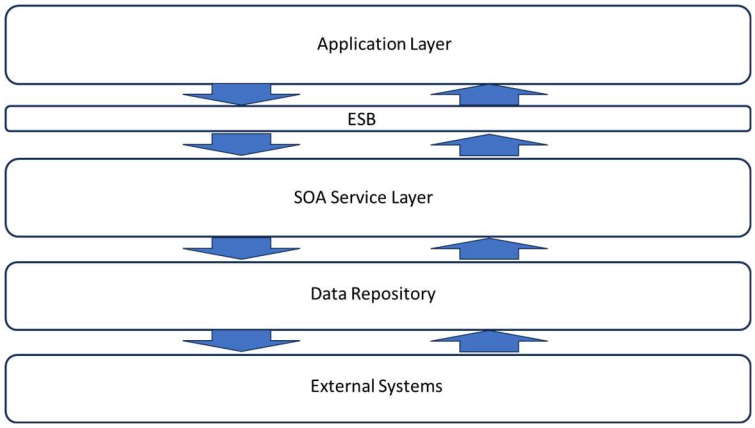
Decision

Both options were selected. Direct access of external systems is feasible as many COTS products have existing communication channels available and this provides the most seamless method to move the data. It is noteworthy that this method also provides intersystem communications in a more reliable fashion as no user input is necessary. However, while automated communications between COTS systems is ideal, it is not always feasible. For this reason, applications will be required to be developed and integrated independently.

Consequences

Incorporating multiple options means more complexity and more costs. In some circumstances, the automated process could be accomplished without significant development but will likely not always be the case. Additionally, integration of COTS tools could introduce licensing fees that will endure over the long term.

Supporting Architecture Diagrams


Application Layer
ESB
SOA Service Layer
Data Repository
External Systems

3. How to optimize interoperability with multiple external healthcare organizations

Context and Problem Statement

Direct interoperability means optimal integration and operation on many levels. This can be a labor intensive task if no optimal solution can be found.

Considered Options

The first approach is manual development an interface for each instance of desired interoperability. The second is to utilize existing COTS methods of inter- and intra-system communication among healthcare systems. Finally, an AI-based solution that can learn how to reach and obtain the pertinent data.

Decision

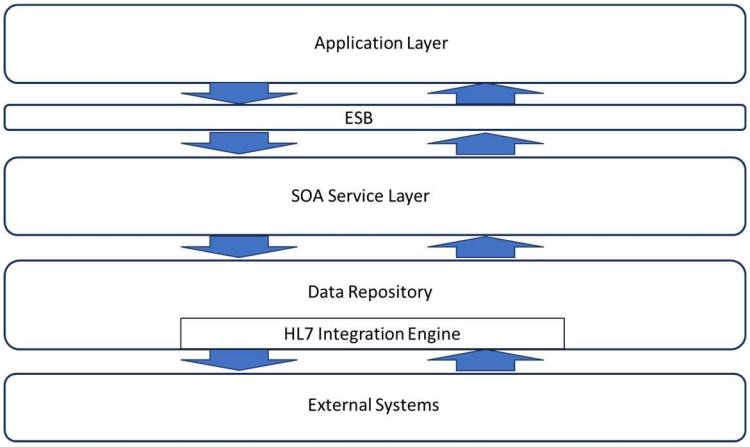
The decision was made to utilize both manual developed applications to interact with external systems and the Health Level 7 (HL7) protocol. The HL7 is an international standard establishing messaging rules between systems that is prevalent in the industry and in continued development by third party developers. Although an automated, standardized approach is desired in all instances, it will not meet all requirements. For instance, patients will not be communicating via this protocol at all as patients in general will not have a healthcare solution at their disposal. Also, standardized protocols are always in development, so changes could require a backup solution be utilized.

Consequences

Integrating multiple methods for clients to incorporate appropriate data into the JFS data repository introduces increased complexity and costs, it is desired to utilize the established solution whenever possible (less development). However, even if only as a backup solution, custom developed client applications will need to be incorporated and planned for. The use of a standardized protocol solution means others will be maintaining and developing it, JFS will be required to evaluate and possible incorporate

updates to the proposed healthcare tool as new releases become available and frequently used. Direct inter-system connections means stringent security requirements must be put in place and adhered to strictly as well.

Supporting Architecture Diagrams



4. Selection of secure hosting platform

Context and Problem Statement

There are multiple platforms that could host the JFS Healthcare tool. One must be selected that can keep costs low and help maintain HIPAA compliance.

Considered Options

There are many options available. AWS, Atlantic, Microsoft Azure, and Rackspace were mainly evaluated in depth. Each overs tiered pricing plans that can be changed as needed. All offer HIPAA compliance with certifications and security frameworks in place that are appropriate for a tool that will store healthcare records.

Decision

The decision to AWS was made based upon the immense track record and highest value for the options given. With pricing based on usage with hardware exclusively for developing HIPAA compliant services, AWS is the most prepared to host HIPAA compliant services, and the most experienced at doing so. Other options offered more structured pricing options (like Azure seemingly ala carte pricing) that would likely not meet JFS's needs ideally, resulting in features and options that are paid for being left unutilized. Other options, like Rackspace, offer public cloud based solutions that could introduce additional security issues and indicate inexperience in healthcare industry solutions.

Consequences

The many features available within the AWS hosting solution introduce higher costs. Further, training would likely be required for JFS developers to ensure optimal use can be made of the platform and available resources, while minimizing required support from Amazon developers. Also, a PC Mag editor rating of only 'Good' will require additional efforts to ensure patient data privacy in a TBD fashion.

5. Programming languages

Context and Problem Statement

There are many programming languages, or combinations of programming languages, that could be chosen to implement this application. Front-end, back-end, training, future development, feature availability, speed and support – all characteristics of programming languages that make this decision one of the most significant.

Considered Options

C++, Javascript, Java, Python, Ruby, Rust, Typescript were all considered options for reasons including JFS has experience using these languages or JFS developers think new languages could present a viable and beneficial opportunity for our tool.

Decision

While JFS developers have some experience with C++, it is dated and its features can be performed by other languages that have additional benefits.

Java: is a very good language with a lot of industry experience and capabilities but industry appears to be moving on (first it was onto python, now several others as well).

JFS developers have plenty of experience with Javascript, as does the industry. It is a very common language that is sure to be more than capable but is losing industry support.

Python has been gaining popularity, especially in administrative applications. However for unknown reasons, it has been abandoned by some larger systems.

JFS developers have limited Ruby experience but have reported positives in terms ongoing support. However, it is also reported that Ruby can be slow but it is not clear how significant of a downside that is for this application.

JFS developers have no experience with Rust but continue to propose it as the language of the future. Reported benefits are performance comparable to C++, while maintaining firm memory control and interaction with a developer-loved packager, Cargo. Rust also is easily suitable for multiple platforms as it compiles to native machine code.

Typescript is another new language that JFS developers have no experience with but continue to report as a language of the future. Essentially, it is explained as a Javascript replacement but with better features. Additionally, there appears to be support for porting directly from Javascript to Typescript. A link is apparent in that Typescript seems to be gaining popularity in an inversely proportional rate that Javascript is losing popularity.

The decision has been made to prioritize longevity without the need to retool to a new language and runtime over compile time. Therefore, the JFS healthcare tool will utilize Typescript for a front-end language and Rust on the back-end of the architecture.

Consequences

Using two new languages will mean significant learning curves for the JFS developers, as well as training expenses. However, this investment is an up-front, one-time cost. The slow compiles times are acceptable when providing better runtime performance. A fast customer experience is prioritized over the JFS developer experience during project integration. Support for both languages is an unknown during initial product stages. However, this is projected to improve over time as the language becomes more seasoned and communities expand. Initial development is expected to greatly assisted by the direct porting of Javascript to Typescript. If necessary, JFS developers could even code in Javascript and utilize this coding function.

Contact Info:

Frank Albright

Just Frank Software™, JK

president@justfranksoftware.com

+1.856.889.0558