Programmer’s Guide

Virtual Letter of Life (VLOL) Application

Version 2.0

Michael Shaw

Michael Marcucci

Kimberly Van Allen

Andrew Coleman

Heather Barnes

Panhavorn Hok

SWEN 670 Software Engineering Project

University of Maryland Global Campus

Date: 10/20/2020

Revision History

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 07/09/2020 | 1.0 | Initial Programmer’s Guide Release by Augustin Mwamba.  Reviewed by EMSPlus internal team. |
| 07/21/2020 | 1.1 | Reviewed and approved by EMSPlus internal team |
| 10/20/2020 | 2.0 | Reviewed and approved by Team VLOL |

Table of Contents

[1. Introduction 1](#_Toc54078455)

[1.1. Overview 1](#_Toc54078456)

[1.2. Scope 1](#_Toc54078457)

[1.3. References 1](#_Toc54078458)

[1.4. Acronyms 2](#_Toc54078459)

[2. Context 2](#_Toc54078460)

[2.1. Architecture Guiding Principles 3](#_Toc54078461)

[2.2. Architecture Constraints and Assumptions 5](#_Toc54078462)

[2.2.1. Constraints 5](#_Toc54078463)

[2.2.2. Assumptions 6](#_Toc54078464)

[2.2.3. Alignment with Priority Areas 6](#_Toc54078465)

[3. Data 7](#_Toc54078466)

[3.1. Storage 7](#_Toc54078467)

[3.2. Data Integrity 7](#_Toc54078468)

[3.3. Data Archiving 7](#_Toc54078469)

[4. Software 7](#_Toc54078470)

[4.1. Business Logic 9](#_Toc54078471)

[4.1.1. ORM vs SQL Schema 9](#_Toc54078472)

[4.1.2. API and API Keys 9](#_Toc54078473)

[4.1.3. User Authentication Workflow 9](#_Toc54078474)

[4.1.4. Provider Authentication Workflow 9](#_Toc54078475)

[4.1.5. Duplicate Unique Data 10](#_Toc54078476)

[4.1.6. Normalized and De-Normalized Database Schema 10](#_Toc54078477)

[4.1.7. Field Access to Data 10](#_Toc54078478)

[4.1.8. QR Codes 10](#_Toc54078479)

[4.1.9. Quiet zones for QR Codes 11](#_Toc54078480)

[4.1.10. Authentication for QR Codes 11](#_Toc54078481)

[4.1.11. Invalidation of QR Codes 11](#_Toc54078482)

[4.1.12. Document Uploads for Advanced Directives 11](#_Toc54078483)

[4.1.13. Delegate Access 11](#_Toc54078484)

[4.1.14. User Menu 11](#_Toc54078485)

[4.1.15. Condition Data 12](#_Toc54078486)

[4.1.16. Allergy Data 12](#_Toc54078487)

[4.1.17. Drug Data 12](#_Toc54078488)

[4.1.18. Vaccine Data 12](#_Toc54078489)

[4.1.19. Referenced Identifiers 12](#_Toc54078490)

[4.1.20. Access Logging 12](#_Toc54078491)

[4.1.21. Log Off Timeout 13](#_Toc54078492)

[4.2. Security Threats 13](#_Toc54078493)

[4.2.1. SQL Injection 13](#_Toc54078494)

[4.2.2. Cross-Site Scripting (XSS) 13](#_Toc54078495)

[4.2.3. User Impersonation 13](#_Toc54078496)

[4.2.4. Malware 13](#_Toc54078497)

[4.2.5. DNS Spoofing/Hijacking 14](#_Toc54078498)

[4.2.6. Packet Sniffing 14](#_Toc54078499)

[4.2.7. Denial of Service Attacks/Database Denial of Service Attacks 14](#_Toc54078500)

[4.2.8. Brute Force Attacks 14](#_Toc54078501)

[4.3. User Interface 14](#_Toc54078502)

[5. Technical Positions 15](#_Toc54078503)

[6. Coding Standards 15](#_Toc54078504)

Table of Figures

[Figure 1: VLOL system basic functionality 2](#_Toc54078347)

[Figure 2: VLOL system layered view 4](#_Toc54078348)

[Figure 3: VLOL system Deployment diagram 5](#_Toc54078349)

[Figure 4: VLOL system implementation view (Business Layer). 8](#_Toc54078350)

[Figure 5: VLOL system implementation view (Presentation Layer). 8](#_Toc54078351)

Table of Tables

[Table 1: Coding Standards 15](#_Toc54078364)

# Introduction

## Overview

This Programmer’s Guide outlines key principles and guidelines for developing and upgrading the Virtual Letter of Life (VLOL) software, which was developed by Team EMSPlus during Summer 2020 and Team Life Line during Fall 2020. The sections below describe and explain the rationale for various implementation decisions made by these teams during the development of the VLOL project.

## Scope

The purpose of this programmer’s manual is to help streamline software development for the VLOL system and to make future software modifications easier. By using this programmer’s reference manual, software developers working on the VLOL system will understand how the application is constructed, and how to make modifications or improvements to the existing code when necessary. Specifically, the manual provides information on the following:

* An overview of the architecture of the VLOL system and the rationale behind it.
* Programming guidelines and desired program features for the VLOL software.
* Software development utilities, and libraries available for developing the VLOL system.
* Guidelines and considerations for security and safe application development

## References

1. Bass, L., Clements, P., & Kazman, R. (2012). Software architecture in practice. Upper Saddle River, NJ: Addison-Wesley Professional.
2. Cervantes, H., & Kazman, R. (2016). *Designing Software Architectures: A Practical Approach*. Boston, MA: Addison-Wesley.
3. Lifeline team (2020). *Virtual Letter of Life (VLOL) Software Design Document*.
4. Lifeline team (2020). *Virtual Letter of Life (VLOL) Software Requirements Specification Document*.
5. Lifeline team (2020). *Virtual Letter of Life (VLOL) User Interface Design Document*.
6. IEEE Std 14764-2006 (2006). Software Engineering, Software Life Cycle Processes, Maintenance. Retrieved from <https://standards.ieee.org/standard/14764-2006.html>
7. Microsoft Patterns & Practices team (2009). *Microsoft Application Architecture Guide*. Microsoft Press.
8. Pressman, R. S. (2014). Software engineering: A Practitioner's Approach. New York, NY: McGraw-Hill Education.
9. Sommerville, I. (2016). Software engineering. Essex, England: Pearson Education Limited.
10. The Section 508 Standards. Retrieved from [The Section 508 Standards](file:///C:\Users\augustin\AppData\Roaming\Microsoft\Word\The%20Section%20508%20Standards)

## Acronyms

The following is a glossary of terms used in this document:

* **DB** – Database
* **HIPPA** - Health Insurance Portability and Accountability Act
* **JDBC** - Java Database Connectivity
* **MVC** - Model View Controller
* **ORM** - Object Relational Model
* **SDD** - Software Requirements Document
* **SRS** - Software Requirements Specification
* **VLOL** - Virtual Letter of Life
* **API -** Application Programming Interface
* **HIE** - Health Information Exchange

# Context

The VLOL system's main purpose is to provide users and first responders with a web-based version of the letter of life. At its core, the application allows authenticated users to retrieve User’s Letter of Life information from a data source (an external database) using a desktop browser or a mobile device, and to create, update and delete records based on roles and privileges they are granted within the system. Unauthenticated users (users without an account or credentials) can access the application’s contact page, information page, and registration page.

The context diagram shown in **Figure 1** illustrates the basic functionality of the VLOL software.

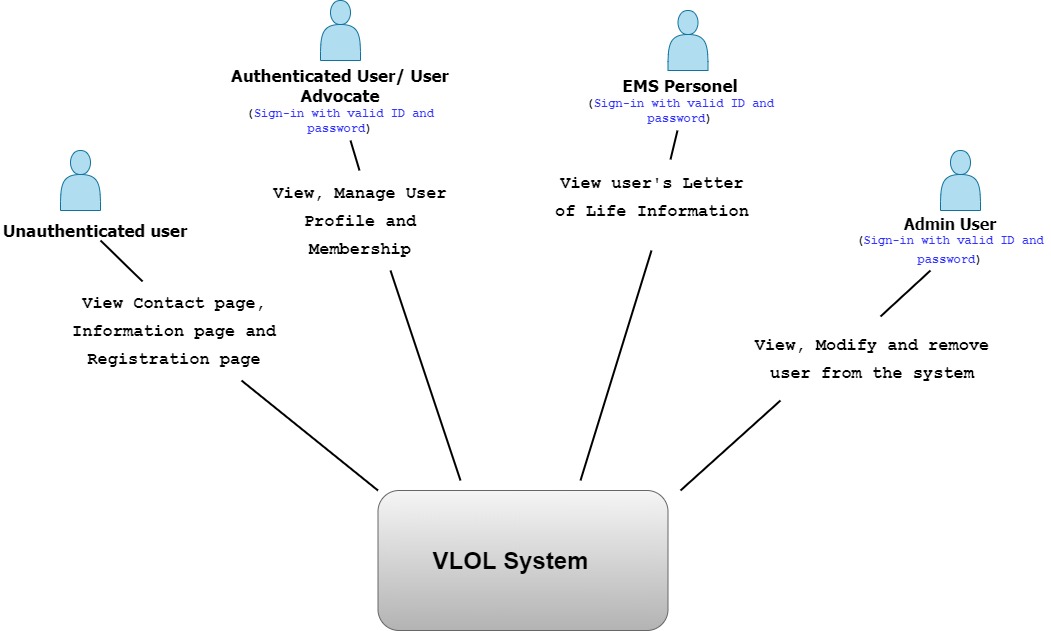


Figure : VLOL system basic functionality

The ideal solution that satisfies the system’s functional requirements, from a perspective of the four groups of users (Figure 1), is a software architecture that not only adheres to proven principles, but also minimizes costs and maintenance requirements, and promotes overall system usability and extendibility.

The remainder of this section describes the guiding principles and rationale behind the architectural choice made for the VLOL software, known constraints and assumptions for this architecture, and the architecture alignment with business priorities.

Future considerations to this architecture may include an API that exposes endpoints for State Healthcare Exchange platforms such as CRISP in Maryland to consume. The architecture of API would adhere to general best practices considered for REST.

## Architecture Guiding Principles

The VLOL system is based on a lightweight MVC (Model View Controller) design pattern consisting of a data model, a view (user interface), and controllers (processes that handle input).

The development has adopted the following design decisions to satisfy the system’s functional requirements and quality attributes:

* Logically structure the VLOL system as a web and mobile application, and use the Separation of Concerns principle to organize the system into software layers based on their functionality (presentation layer, business logic, and data access layer).

The rationale behindthis design decision is that the VLOL system is primarily intended to be accessed from web browsers and deployed on mobile devices such as smartphones, tablets, etc., therefore using a web application or mobile application architecture would meet the system’s functional requirements and quality attributes.

***Pros***: Web applications can be accessed from a variety of platforms using a standard web browser, they have fast page loading time and are simple to deploy. Mobile applications, for their part, support rich user interaction.

***Cons***: Web applications do not support rich user interaction, and mobile applications are less portable and have screen limitations.

***Discarded Alternative***: Rich Internet Application (RIA)

The RIA option was discarded because, although this type of application supports a rich user interface and can be easily deployed or upgraded, the development team believed that the plugins (runtime environment) for executing RIA were less available than the Java Virtual Machine (JVM).

**Figure 2** shows a logical or layered architectural view of the VLOL system.

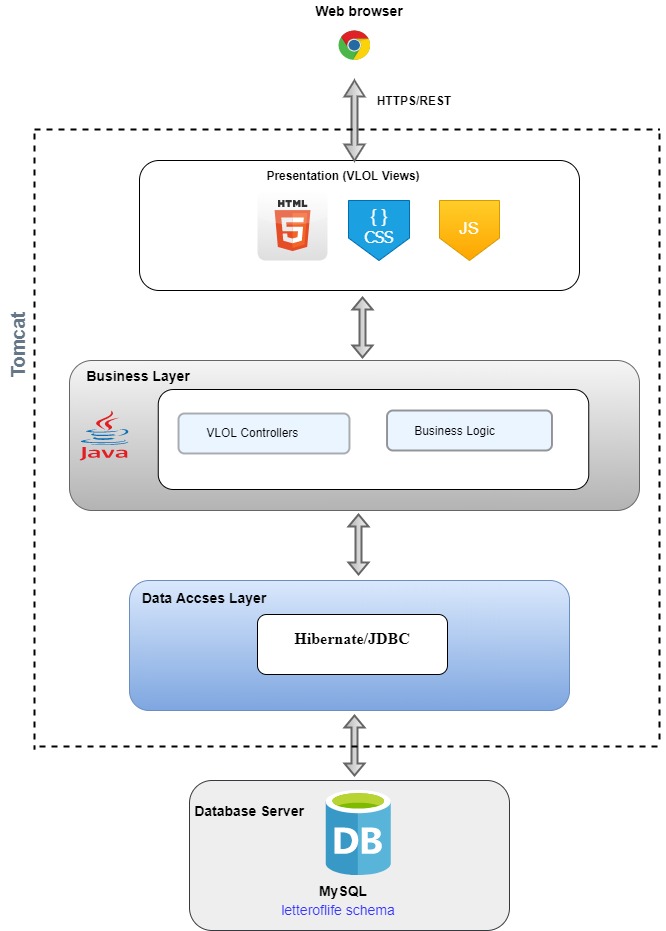


Figure 2: VLOL system layered view

* Physically structure the application using a three-tier deployment pattern with Client, Business, and Data tiers.

***Rationale***: The system must be accessed from a web browser and a mobile device, and an existing database (Relational database) must be used.

The benefit of this approach is that there is less performance cost compared to a 2-tier deployment scheme, for example.

The following diagram shows the deployment pattern of the VLOL system

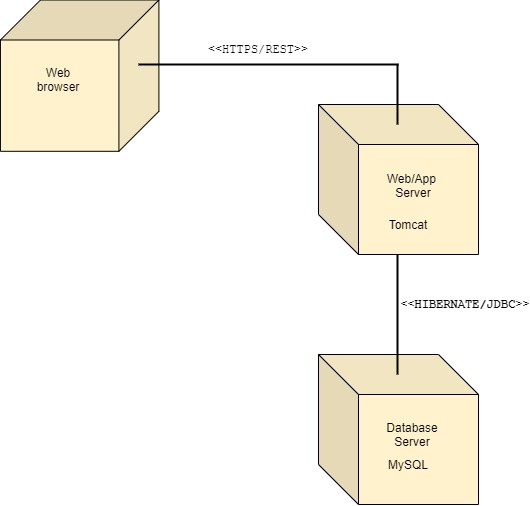


Figure : VLOL system Deployment diagram

* Select the Spring Security framework to manage authentication and authorization: The needs for this application also include managing user authentication and authorization. All these features are available in the Spring Security framework which can easily integrate with other frameworks such as Java.

In summary, the development team believes that the functional decomposition of the system and service components and their distributed deployment will provide improved scalability, availability, manageability, and resource utilization for the newly created VLOL system.

After new analysis of the architecture, Team Lifeline decided that the core design did not warrant any major changes and time would be more well-spent building off the existing design and making iterative enhancements.

## Architecture Constraints and Assumptions

Development of the VLOL software will follow the design roadmap used for greenfield systems for mature domains since it’s a new system that is being developed from scratch and there is an already established infrastructure of tools and technologies and associated knowledge base for this type of systems.

The following constraints and assumptions are identified to have the potential to properly constrain and bound the development of the VLOL system:

### Constraints

**Application Constraints**: The VLOL application and database designs must be flexible, and extensible to satisfy the future need for better performance, accessibility, scalability, and availability. Therefore, existing data must never be negatively impacted by new changes or implementations.

**Development Constraints**: The VLOL system development must comply with industry standards including existing software patterns, coding standards, and best practices. The development process should include a formal validation process to assure that the enhancements are reliable, functional, and scalable and meet business requirements.

**Technology Constraints**: The system shall be composed primarily of open source technologies (for cost reasons). For those components where the value/cost of using proprietary technology is much higher, proprietary technology may be used.

**Architectural Constraints**: The system must be accessed through a Web browser on different platforms (Windows, OSX, Linux, etc.) and via Mobile devices.

### Assumptions

**System Scalability**: It must be possible to scale the deployment of the VLOL system to a large number of users geographically dispersed across the US without a significant increase in effort or cost.

**System Usability**: The application user interface must be intuitive to use, must provide a good overall user experience, must be localized and globalized, and must be accessible to disabled users following the **508** standards.

**System Testability**: The VLOL system must be designed in a modular fashion to support testing. It must also provide instrumentation or implement probes for testing, enable mechanisms to debug output, and ways to specify inputs easily. Also, the system must design components that have high cohesion and low coupling to allow the testability of components in isolation from the rest of the system. Proper testing tools should be used for unit testing and the execution of unit tests should be automated.

**System Availability**: The system must run 24 x 7 x 365, with an overall availability of 99.9% and downtime of 0.1% (Except for planned outages or scheduled maintenance).

**System Security**: All communications must be authenticated and encrypted using strong and secured certificates. Additionally, secure authentication methods used include JWT tokens. Server-side and front-end validation are used to ensure maximum prevention and defense from bad actor threats.

**Reusability**: Modularization of components and functionality shall be used throughout the development of this software.

### Alignment with Priority Areas

The goal initial purpose of this software architecture is to help achieve development goals. Therefore, the development team believes that having a design will help drive development work, guide system construction, satisfy system requirements, and ultimately prepare for an eventual software release.

# Data

This section of the Programmer’s Guide describes application data storage, security, and integrity checking schemes. Additional information on the VLOL application data architecture can be found in the application Software Design Document (SDD).

## Storage

The VLOL system will store its data in a MySQL database server to minimize costs and save time. Access to the database server must be accomplished via the Hibernate Framework. Database connections must be standard MySQL connections over TCP/IP. HIPPA security and standards data access security measures must be observed at all times. The VLOL application data will be backed up, when necessary, using appropriate backup devices.

## Data Integrity

The VLOL system must ensure that application data is protected against attacks that try to steal or corrupt data and protect the mechanisms used to gain access to the data source. The VLOL system will include sanitizing error and exception handling mechanisms so that data source information is not revealed, and using least privilege accounts to restrict privileges to only those needed to perform the operations required by the application.

Database access should be done through parameterized queries to prevent SQL injection attacks from succeeding. The system will prevent the use of string concatenation to build dynamic queries from user input data.

Database integrity checking and recovery software integrity checking should run as a

background task checking for signs of database corruption.

Provide definitions for all terms, acronyms, and abbreviations required to properly interpret how they are used in this document. Provide a reference to the project’s glossary document if necessary.

## Data Archiving

When necessary, application data will be archived following the best practices regarding data arching, and archived data will be stored to tape or disk.

# Software

The VLOL software modules (source code, data files, executables, documentation, etc.) are organized in terms of packaging, layering, and configuration management (ownership, release strategy, etc.).

Figures 4 and 5 show the application layers, packages, and package hierarchy. The package hierarchy starts with a standard Java namespace compliant structure and then is divided into sub-packages for the different layers including *vlol.src.main.java.com.vlol* for the Business layer (controller, model, repository, and service) and *vlol.src.main.resources.templates*, *vlol.src.main.reources.templates.admin* for the Presentation layer. The business layer includes the files contained in the Controller, Model, Service, and Repository folders. See below for a screenshot reference from the project.

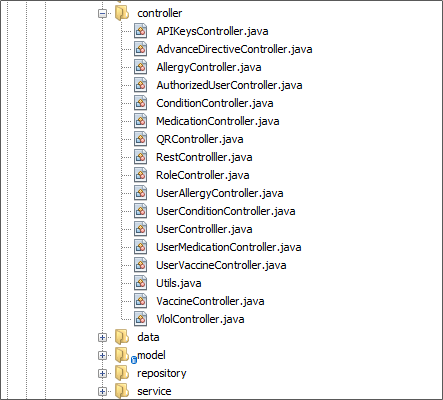


Figure : VLOL system implementation view (Business Layer).

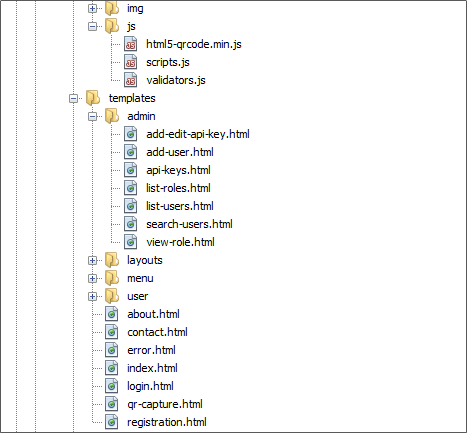


Figure : VLOL system implementation view (Presentation Layer).

## Business Logic

Business logic is defined as any application logic that is concerned with the retrieval, processing, transformation, and management of application data; application of business rules and policies; and ensuring data consistency and validity. The business logic components shown in Figure 6 in the application Business Layer, contain behavior and application logic that is specific to the use cases (or user stories) described in the VLOL Software Requirements Specification Document (SRS).

### ORM vs SQL Schema

When choosing the ORM to generate the schema or to use a predefined schema file we chose the schema file as it allows for a clearer definition of the database definition.

### API and API Keys

Several integrations were proposed, and as there is little documentation about the integrations themselves or which integrations will move forward, a simple schematic for integration was chosen. The API Key would be passed via a header with any request, although there are many different options for authentication this method is hopefully the easiest to implement for a potential third-party.

JSON was chosen as the format for the API as it is well supported in most languages and is becoming a web standard for APIs.

### User Authentication Workflow

The implementation of the user registration and verification involves a user registering and verifying their email address. The reset password functionality is also completed via email verification. Other options were considered such as no verification, phone verification, and security questions. No verification was quickly thrown out as we have HIPPA related data and it behooves us to implement easy security wins. Phone verification is a little more challenging and, in most cases requires a secondary service, as such the choice was made to not prioritize this feature. Security questions were previously used however were removed as they were deemed a security risk as they would be considered an alternative to email verification which in most cases is more secure.

### Provider Authentication Workflow

The provider has enhanced access to all users of the application and hence the registration and verification is a balance between administrator action and user verification. There are two established workflows for the addition of a provider. A provider may sign up and verify their email, however before they receive access the admin will receive a notification to verify the account and grant them access. This would be an ideal workflow for external providers, however, for EMS personnel which may be in the organization which owns the application, this would require a lot of admin verification. A second option provided is for the third party to integrate an external database with the application database or for the database admin to batch load verified users into the application’s database. This workflow removes the need for admin verification but requires each user to reset their password for their initial access. External login services such as Single Sign-On were considered however, as little information is known about the client systems at this time, we chose to provide the application login service and any enhancements can be done after.

### Duplicate Unique Data

Some data such as social security numbers are already unique identifiers. However, as we have no way to verify the correctness of this data or if an entry was mistakenly entered, the decision was made to make these fields non-unique. This prevents other users from failing to register information due to other users’ errors and allows for duplicate account creation. It is hoped that a maintenance team can periodically check these fields to identify any errors, misuse, or duplication. It is also possible that a third-party integration may allow for verification in which case this may be revisited.

### Normalized and De-Normalized Database Schema

Some tables fit in a normalized database schema, such as the user tables and user related tables. However, several tables are setup in a denormalized fashion, these tables may potentially contain duplicate data with the user table. These tables are not normalized to allow for the third-party data to change without affecting the users’ data. For example, if the FDA were to drop a drug the user data should not be affected.

### Field Access to Data

The idea of having a physical portal for the Virtual of Letter of Life is one of the main goals of the project. To do so the QR code was chosen as a way for people to access the user’s VLOL account. The QR code was chosen due to its simplicity and wide support. A specific app does not need to scan a QR code and it is small enough to be on a bracelet if needed. Another option would be to provide a multi-page print version of the VLOL data however, this cannot be updated easily and for people without a good place to store this such as the homeless, a compact version is better. Having a code to access the application also allows for it to be invalidated in case it is lost or stolen.

### QR Codes

The decision to use QR codes was made because the general public is already used to thinking of QR codes as something that needs to be scanned to be opened. By using a portable device like a mobile phone to scan the QR code, the information is stored in a more durable form than could a sheet of paper. The speed of scanning the QR code is also preferable to other methods such as printing a pdf. The print size was chosen to ensure easy readability and scanning

### Quiet zones for QR Codes

The padding between the QR codes and its surroundings to help the reader identify the code at just a glance and to make reading the code easier to scan by older devices.

### Authentication for QR Codes

To allow other people to access a user’s data via a QR code security measures needed to accompany the code so that not just anyone with a URL can access the data. JWT authentication was chosen due to its stateless nature and URL friendly design. It also allows for a higher level of security than just pairing the URL with a secret key.

### Invalidation of QR Codes

In the case of a lost QR code or if there is an unauthorized usage, the decision was made to implement a way for the QR code to be invalidated and a new one generated. The design chosen was for simplicity for the user, with a button to invalidate and generate a new code. The other option was to allow for multiple QR codes to be generated and invalidated individually. However, the use case for this may not exist for our user base, and for simplicity of design, the single code option was used.

### Document Uploads for Advanced Directives

Advanced directives are a type of form that tells a provider what to do in various situations if you are not capable of responding. It was identified that these forms may be paired with a Letter of Life. The decision was made to allow these forms to be uploaded to the server and to be retrievable via the interface. There are various forms therefore a feature to upload and store multiple documents were added to the interface.

### Delegate Access

The original application allowed for the ability for one user to be made a delegate for another user. The decision was made to enhance this design and to provide users to designate other users as delegates of their accounts. The designation would be done via an email address which we believe would be the simplest option to specify an account.

### User Menu

The original application consisted of a large form where information would be entered on registration. We believe that the primary user would be on the elderly patient and therefore large forms can be daunting. We decided to split out the interfaces with a simple registration page as well as separate sections for the data points, where it made sense. We hope that this would provide a better interface for users to enter and update information.

### Condition Data

For condition data, the ICD data was chosen as the source for the autocompletes. Our other option was MeSH data however ICD was chosen as it offers identifiers that can be used by providers to look up and access additional information. We hope that ICD data would also be used by many of the third-party integrations we are looking at. ICD is used often in the medical sector whereas MeSH is often used for the tagging of articles.

### Allergy Data

Allergy data does not have a great ICD equivalent. For this source MeSH data was used to populate a common list of allergies. Some massaging of the data was also done to remove duplication.

### Drug Data

For condition data, the FDA data was chosen as the source for the autocompletes. FDA drug identifiers are used in the clinical sector and the FDA provides an up-to-date registration of drugs. Data massaging also takes place to remove some of the other drugs the FDA tracks like “hand sanitizer” which is not as helpful for the application.

### Vaccine Data

Vaccines are another set of data that is not broken out specifically in either ICD or MeSH data, however, to get some form of a data source ICD data provides some vaccine names as synonyms to a blanket vaccine heading. To easily get this data we pull it from a website that lists out the synonyms and massages the data to remove duplication and make the data more user-friendly.

### Referenced Identifiers

For all third-party data sources, they are used in an autocomplete to help a user fill out their forms. In doing so the application can tag the data with helpful identifiers that can be used by providers. However, choosing a value from an autocomplete is optional, which means that data will not contain referenced identifiers in that case. We hope that the users will choose data from the autocomplete and try to guide them to do so, but given that the data may not be up-to-date or their data may not be listed the choice was made to allow for write-in responses.

### Access Logging

For HIPPA compliance an access log needs to be written so that an analysis can be done on the logs. This was set up as a file in the docker container with a rolling log.

### Log Off Timeout

Due to HIPPA, there is a requirement to log off the user after an amount of time to prevent other users of a shared computer from accessing the application. Based on that we have set a 30-minute timeout after a user is inactive.

## Security Threats

Several different application security threats were considered throughout the project.

### SQL Injection

Whenever a form can receive data and enter it into a SQL database this is a threat. Hibernate the ORM which integrates with SQL should be using prepared queries to prevent direct SQL injection into a query.

### Cross-Site Scripting (XSS)

As a user’s data can be viewed by an admin or provider there is a potential for a user to add to their profile code which when viewed by another user could run as a script. To prevent this all strings a checked for RFC 3986 compliance which blocks characters such as < and >. This on its own is sufficient to block 99% of XSS attacks.

### User Impersonation

One of the goals for the application is to integrate with Health Information Exchanges (HIE). These services provide medical information on a user by matching the application's user information with their own. A simple attack vector that could be used to download this data would be to create fake accounts with a series of social security numbers. The HIE would then update the application with that socials information which could then be downloaded by the user. Currently, the application has no means to prevent this however we hope that by integrating with the HIE they will be able to provide information to verify the user.

### Malware

As the users can upload files via the Advanced Directive feature a rogue user could upload an infected file which can cause another user to be infected when opened. Although the application does check for a basic file type this is nothing that cannot be spoofed and pdfs and documents can contain malware. Therefore, a malware scanner that runs on upload would be preferred to prevent this attack vector. However, this is not currently in scope due to the attack’s limited range of effectiveness.

### DNS Spoofing/Hijacking

The application does pull from third-party resources therefore it is possible that the third-party websites may have their domain hijacked or a bad actor could attempt to redirect the application to use their data. This attack is limited in effectiveness as the database cannot be SQL injected or used with cross-site scripting. The damage may be limited to downtime for the auto-complete feature. As such no action was taken to prevent this attack, which in itself would be a very unlikely attack vector.

### Packet Sniffing

An insecure transfer protocol such as HTTP can easily be intercepted and interpreted, as such HTTPS would be the preferred method of communicating with the users. This is currently out of scope as the application will be handed off for integration into the client’s systems. However, we offer this recommendation for their installation.

### Denial of Service Attacks/Database Denial of Service Attacks

All websites risk denial of service attacks, however, it is not usually a problem until it is a problem. If it does become a problem several services can be placed in front of the application to help prevent it. As such it is outside of the scope of the application.

### Brute Force Attacks

Brute force password attempts on a user’s profile although limited in scope to a single user are still preventable. Firstly, the application ensures that there is a minimum password complexity. Secondly, the application will lock any account if there are several sequential incorrect login attempts. We also considered blocking IP addresses for several attempts across accounts. However, due to the shared nature of IPs in settings such as hospitals and shared devices combined with the ease for an attacker to switch IPs the risk-reward for blocking a bad actor versus blocking a shared IP address was too low. There was also a consideration for MAC addresses to be linked to a specific account and only be used for that account, however, due to the shared nature of devices in the healthcare community and public machines which may be used to access the risk of hindering real users was too great.

## User Interface

The VLOL system user interface shall conform to usability guidelines for user interface design, particularly the Jakob Nielsen's 10 general principles for interaction design.

Additionally, to provide rich feedback to the user, especially for errors and exceptions, and the system unresponsiveness, the VLOL system must also implement technologies and techniques that provide maximum user interactivity, such as Asynchronous JavaScript and XML (AJAX) in web pages and client-side input validation.

The reader is encouraged to consult the VLOL User Interface Design Document for additional information about the system user interface.

# Technical Positions

The VLOL system uses the following framework technologies to build its application and to interact with the underlying MySQL database:

* **Java SE 11.0.7**. Java is used as the core development technology**.** The system will be programmed in Java and Java-related technologies to leverage the expertise of the development team.
* **Spring Boot Framework**: Spring Boot will be used to provide a complete and modular framework for developing the VLOL application. It is a widely used framework in support of enterprise application development.

An alternative that was considered for developing the VLOL system is **JEE**. Spring was eventually selected because it is considered more “lightweight” and the development team is already familiar with it, which can lead to greater and earlier productivity.

* **Hibernate Framework**: Hibernate is an object to relational mapping (ORM) framework that integrates well with Spring. It offers to the VLOL application an ORM Framework specialized in data retrieval and persistence.

Other ORM frameworks were not considered, as the development team was already familiar with, and happy with the performance of Hibernate.

# Coding Standards

The development of the VLOL system will adopt the following industry naming and coding standards:

Table 1: Coding Standards

|  |  |
| --- | --- |
| **Standard** | **URL** |
| CERT Secure Coding Standards – Java Guidelines | <https://www.oracle.com/java/technologies/javase/seccodeguide.html> |
| Code Conventions for the Java TM Programming Language | <https://www.oracle.com/java/technologies/javase/codeconventions-contents.html> |
| Guidelines, Patterns, and code for end-to-end Java applications | <https://www.oracle.com/technetwork/java/namingconventions-139351.html> |
| Secure Coding Guidelines for Java SE | <https://www.oracle.com/java/technologies/javase/seccodeguide.html> |