Technical Design Document

Virtual Letter of Life (VLOL) Application

Version 2.0

Heather Barnes

Andrew Coleman

Panhavorn Hok

Michael Marcucci

Michael Shaw

Kimberly Van Allen

SWEN 670 Software Engineering Project

University of Maryland Global Campus

Date: October 31, 2020

**REVISION HISTORY**

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 09/29/2020 | 1.0 | Initial Technical Design Document Release |
| 10/31/2020 | 2.0 | Revised Technical Design Document Release |

Table of Contents

[1. Introduction 4](#_Toc51570760)

[1.1. Purpose 4](#_Toc51570761)

[1.2. Application Description and Document Scope 4](#_Toc51570762)

[1.3. Reference Material 4](#_Toc51570763)

[1.4. Definitions and Acronyms 4](#_Toc51570764)

[2. System Overview 5](#_Toc51570765)

[3. System Architecture 5](#_Toc51570766)

[3.1. Architectural Design 5](#_Toc51570767)

[3.2. Decomposition Description 6](#_Toc51570768)

[3.3. Error and Exception Handling 7](#_Toc51570769)

[3.4. Design Rationale 7](#_Toc51570770)

[4. Data Design 8](#_Toc51570771)

[4.1. Data Description 8](#_Toc51570772)

[4.2. Data Dictionary 8](#_Toc51570773)

[4.3. Data Sources 12](#_Toc51570774)

[5. Component Design 12](#_Toc51570775)

[6. Human Interface Design 13](#_Toc51570776)

# Introduction

## Purpose

This document describes the application’s design and architecture. It is intended to be used by the project team or any person who will be working with the application. By reading this document a user should be able to understand:

* + Technology that was used
  + The application’s architecture
  + The database’s design
  + The structure of the web interface

## Application Description and Document Scope

The VLOL application is intended to allow users to provide their doctors and emergency personnel with the patient’s medical information in the case that they cannot provide the information themselves. User and their providers can update the information, including medications, vaccines, conditions, allergies, and important patient-related information such as insurance. Patients can also delegate the updating of their profile to other users which may include family or advocates. Providers themselves have broader access to the application and can search for and update any patient’s record not just those who have granted specific access. The final role is the administrators who can access any part of the application, they can change add and remove users, change their roles, as well as update information in accounts.

## Reference Material

* + Assadullah, M. (2020, August 19). SWEN 670 9040 Software Engineering Project Syllabus. Retrieved from <https://learn.umgc.edu/d2l/le/content/515759/viewContent/18965508/View>
  + VLOL Team (2020, September 7). VLOL Software Requirement Specification V1.1.

## Definitions and Acronyms

* + VLOL – Virtual Letter of Life
  + Web interface – A website or web-based application that a user interacts with
  + Docker – A virtual machine capable of self-contained application deployment
  + SQL – Structured query language (A common relational database system)
  + API – Application programming interface
  + Router – A path used by a web client to request or send information
  + HTML – HyperText Markup Language, a type of programming language used by web clients to display a webpage
  + CSS – Cascading Styles Sheets, a type of language used to apply graphic stylings to HTML
  + Normalization – Allows a database to prevent duplicate data fields by having relationships between tables
  + Denormalization – The storing of duplicative data without relationships
  + MeSH – Medical Subject Headings
  + ICD – International Classification of Diseases
  + QR Code – Quick Response code

# System Overview

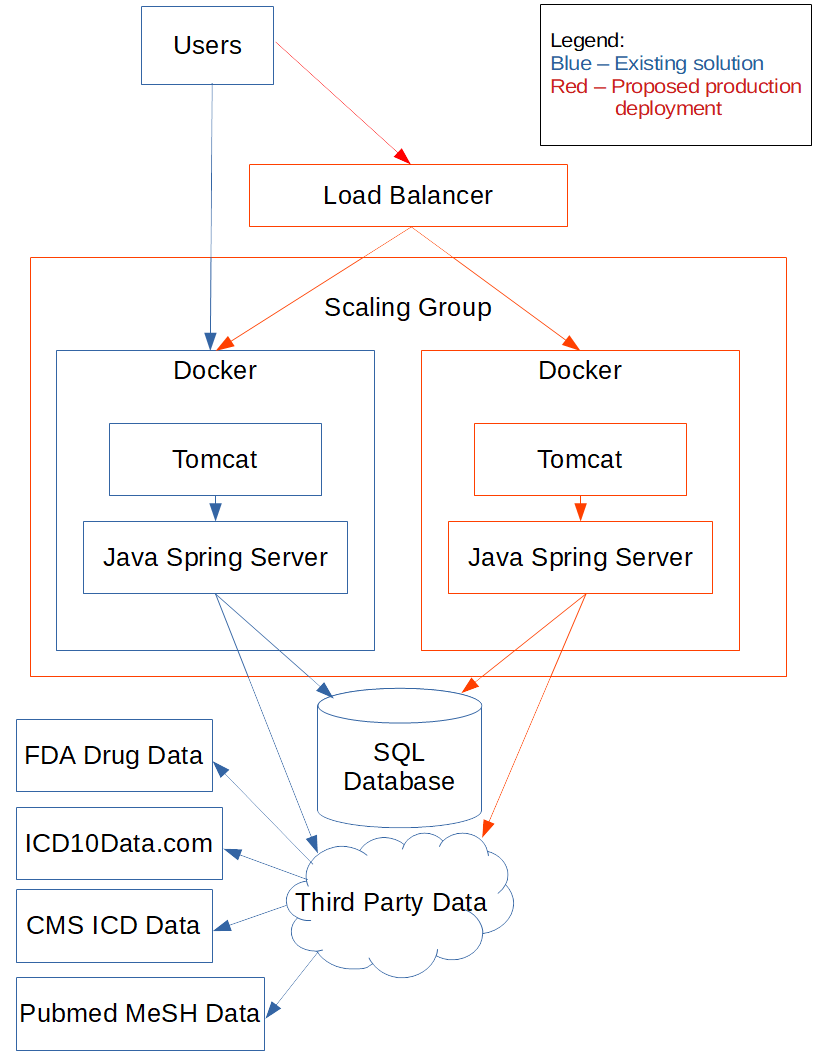
The VLOL application is a java-based web interface. The application serves web pages to the user who then interacts and may provide responses to the server. The server is the intermediary between the user and the database, it provides the logic to determine the flow of the application and the data saved to the database. The system will initialize some of its data via third-party integrations.

# System Architecture

## Architectural Design

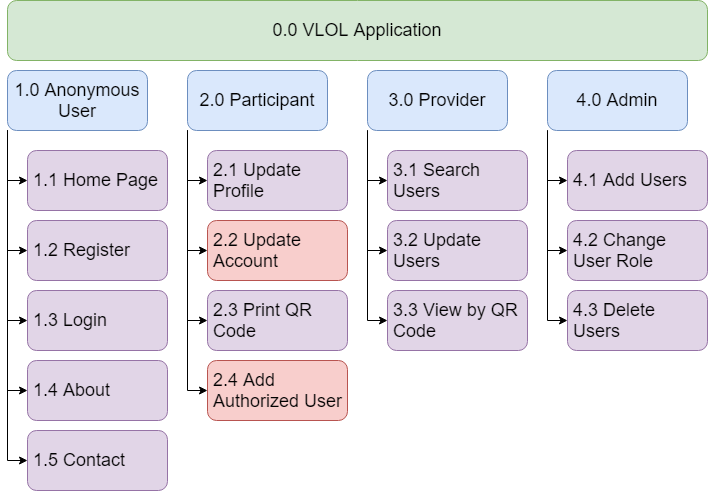
The architecture for this application uses a Java Spring to provide pages to a user and a SQL database to store and provide the data. The Java Spring application is served via an embedded tomcat instance. The system is currently developed using an embedded h2 SQL database however a production deployment would reach out to a separate database. The embedded database and the spring and tomcat instances are built into a docker image. The docker image allows the application to be easily deployed and scale. A production deployment would consist, ideally of, placing the docker image in a scaling group to allow multiple application servers to be deployed depending on the server load. The implementation of which is left up to the client, however, this recommendation is given based on deploying a performant web application. See figure 1, in blue, is the architecture used in development environments, and in red is the recommended deployment for a production environment.

*Figure 1. Architecture Diagram*



## Decomposition Description

*Figure 2. Decomposition Diagram*



The decomposition diagram is broken down by roles in the blue cards. Each role will be able to perform actions, in purple, by the lesser roles. However, actions in red are only able to be performed by the current role or admins.

## Error and Exception Handling

The Java server and the front-end JavaScript application both validate inputs. The front-end code is meant to provide the user with immediate feedback on what fields are required and the formats. The backend is used to provide a failsafe for front end validation, also, to prevent unacceptable inputs via API requests. Error handling of the application itself is twofold, firstly any errors generated by a user would bring the user to an error page, secondly, errors will be written to a log file on the server, this can be used by the application’s maintenance team to diagnose and fix errors.

## Design Rationale

The application was developed in such a way that it would provide UMGC students with the best opportunity to apply what they learned in school to the development of this application. Java is the primary language used in UMGC courses and was therefore selected. Also, the requirements were to develop a web application, therefore, HTML and JavaScript are also used as they are requirements when developing web applications. There are three main libraries used in the Java application, Hibernate, Thymeleaf, and Spring Boot. Hibernate allows java to easily integrate with any database and is also a technology used in several courses. Thymeleaf allows java to integrate with the HTML code by allowing variables to be inserted into HTML templates, this is also for ease of use of development. Finally, Spring Boot provides a framework for authentication, server routing, and a host of other features, this provides developers for several features which gives them a head start on development.

The database itself is designed using SQL and a relational database model. There are several types of SQL databases, this allows the easy transition to the client’s system. Relational databases in general provide an easy to understand data model which matches the needs of the requirements. This is often a safe choice when starting any application and only in certain conditions would a different type of database be needed.

The actual infrastructure of the setup, including tomcat, docker, and the proposed scaling groups and load balancers are all used to provide scalability and easy deployment to the production application. These are not used as much for development but are meant to be leveraged for creating a scalable production deployment.

# Data Design

## Data Description

The database is divided into two sections, a relational data model and a denormalized set of tables used for information retrieval. The relational data model is used to store information about the user, be the user a patient, or an administrator. The denormalized table is used for autocompleting forms, these tables store information from third party databases and make the information available to a user as a way to promote data reuse and normalization for text entries. As this data is from third parties the consistency of ids and data cannot always be established, the denormalization allows this data to be separate from the user data to prevent third-party data changes from affecting the users.

## Data Dictionary

The modifiers in this section are primary key (PK), foreign key (FK), not null (NN), unique (UQ). For a visual please refer to the entity-relationship diagram, figure 3.

**4.2.1 Datasets**

| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| --- | --- | --- | --- |
| name | VARCHAR(64) | PK, NN, UQ | The unique ID for a dataset |
| last\_updated | DATE |  | The last updated date |

**4.2.2 Allergy**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| allergy\_id | BIGINT | PK, NN, UQ | The unique ID for an allergy |
| allergy\_name | VARCHAR(256) | NN, UQ | The allergy's name |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.3 Vaccine**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| vaccine \_id | BIGINT | PK, NN, UQ | The unique ID for a vaccine |
| vaccine\_name | VARCHAR(256) | NN, UQ | The allergy's name |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.4 Condition**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| condition\_id | BIGINT | PK, NN, UQ | The unique ID for a condition |
| condition\_name | VARCHAR(256) | NN, UQ | The allergy's name |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.5 Medication**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| medication\_id | BIGINT | PK, NN, UQ | The unique ID for a medication |
| brand\_name | VARCHAR(256) | NN, UQ | The drug’s brand name |
| generic\_name | VARCHAR(256) |  | The drug’s generic name |
| drug\_action | VARCHAR(1024) |  | The drug’s action on the body |
| controlled | BOOLEAN |  | Controlled substance flag |
| blood\_thinner | BOOLEAN |  | Blood thinner flag |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.6 App Role**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| role\_id | BIGINT | PK, NN, UQ | The user's role unique ID |
| role\_description | VARCHAR(512) |  | The description for this database user's role |
| role\_title | VARCHAR(32) | UQ, NN | The title for this database user's role |

**4.2.7 App User**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| user\_id | BIGINT | PK, NN, UQ | The unique ID for a user |
| role\_id | BIGINT | NN, FK | The role id of the user |
| admin\_comments | VARCHAR(300) |  | System administrator comments |
| date\_created | TIMESTAMP | NN | The creation date for this database user account |
| email | VARCHAR(320) | NN, UQ | The user's email address |
| first\_name | VARCHAR(32) | NN | The user's first name |
| is\_active | BOOLEAN |  | Is the user's account active? |
| is\_locked | BOOLEAN |  | Is the user's account locked? |
| is\_verified | BOOLEAN |  | Is the user's account verified? |
| last\_login\_date | TIMESTAMP | NN | The timestamp of the user last login |
| last\_name | VARCHAR(100) | NN | The user's last name |
| password | VARCHAR(72) | NN | The generated hash value of the user's password |

**4.2.8 User Info**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| info\_id | BIGINT | PK, NN, UQ | The unique id for the info table |
| user\_id | BIGINT | NN, FK | The unique ID for a user |
| adv\_dir\_type | VARCHAR(50) |  | Advance directive type |
| adv\_directive | BOOLEAN |  | Does the user have an advance directive? |
| city | VARCHAR(50) | NN | The user's city of residence |
| dob | DATE | NN | The user's date of birth |
| doctor\_name | VARCHAR(100) |  | The user's primary care physician |
| doctor\_phone | VARCHAR(10) |  | The primary care physician's phone number |
| email | VARCHAR(320) | NN, UQ | The user's email address |
| first\_name | VARCHAR(32) | NN | The user's first name |
| ins\_company | VARCHAR(50) |  | The user's medical insurance company name |
| ins\_policy\_no | VARCHAR(32) |  | The user's medical insurance policy number |
| phone | VARCHAR(10) | NN | The user's phone number |
| poc\_name | VARCHAR(100) | NN | A point of contact for the user |
| poc\_phone | VARCHAR(10) | NN | The phone number for the user's point of contact |
| ssn | VARCHAR(9) | NN, UQ | The user's social security number |
| street\_address | VARCHAR(100) | NN | The user's street address |
| us\_state | VARCHAR(2) | NN | The user's state of residence |
| user\_comments | VARCHAR(300) |  | User additional comments |
| zipcode | VARCHAR(5) | NN | The user's zip code number |

**4.2.9 Advance Directive**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| advance\_directive\_id | BIGINT | PK, NN, UQ | The unique ID for an allergy |
| user\_id | BIGINT | PK, NN, FK | The foreign key to the user table |
| advance\_directive\_file | BLOB |  | The file in a binary form |
| advance\_directive\_content\_type | VARCHAR(128) |  | The MIME type to serve |
| advance\_directive\_filename | VARCHAR(256) |  | The name of the stored file |
| advance\_directive\_type | VARCHAR(64) | NN | The type of advanced directive |

**4.2.10 Authorized User**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| authorized\_user\_id | BIGINT | PK, NN, UQ | The unique ID for an authorization |
| user\_id | BIGINT | PK, NN, FK | The foreign key to the user table |
| authorized\_email | VARCHAR(320) | NN | The email authorized |
| authorized\_name | VARCHAR(128) | NN | The name of the authorized user |

**4.2.11 User Allergy**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| allergy\_id | BIGINT | PK, NN, UQ | The unique ID for an allergy |
| user\_id | BIGINT | PK, NN, FK | The foreign key to the user table |
| allergy\_name | VARCHAR(256) | NN, UQ | The allergy's name |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.12 User Vaccine**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| vaccine \_id | BIGINT | PK, NN, UQ | The unique ID for a vaccine |
| user\_id | BIGINT | PK, NN, FK | The foreign key to the user table |
| vaccine\_name | VARCHAR(256) | NN, UQ | The allergy's name |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.13 User Condition**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| condition\_id | BIGINT | PK, NN, UQ | The unique ID for a condition |
| user\_id | BIGINT | PK, NN, FK | The foreign key to the user table |
| condition\_name | VARCHAR(256) | NN, UQ | The allergy's name |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

**4.2.14 User Medication**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| medication\_id | BIGINT | PK, NN, UQ | The unique ID for a medication |
| user\_id | BIGINT | PK, NN, FK | The foreign key to the user table |
| brand\_name | VARCHAR(256) | NN, UQ | The drug’s brand name |
| generic\_name | VARCHAR(256) |  | The drug’s generic name |
| drug\_action | VARCHAR(1024) |  | The drug’s action on the body |
| controlled | BOOLEAN |  | Controlled substance flag |
| blood\_thinner | BOOLEAN |  | Blood thinner flag |
| ref\_id | VARCHAR(64) |  | The original dataset’s id |

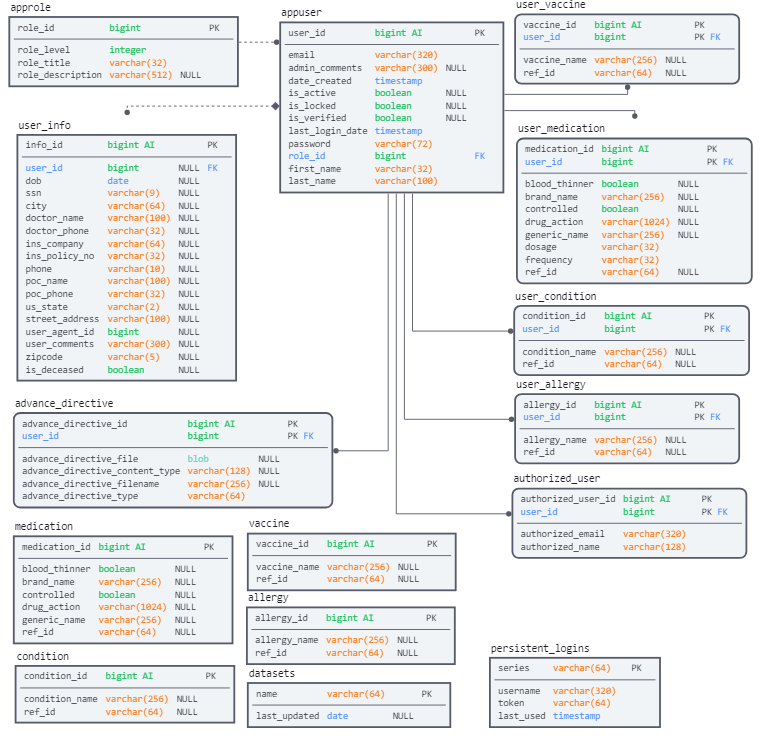
**4.2.15 Persistent Logins**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| username | VARCHAR(320) | PK, NN, UQ | The user name that is logged in |
| series | VARCHAR(64) | NN | Internal spring boot field |
| token | VARCHAR(64) | NN | The stored token |
| last\_used | TIMESTAMP | NN | Last used date of the token |

**4.2.16 Document**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field Name** | **Data Type** | **Modifiers** | **Description** |
| document\_id | BIGINT | PK, NN, UQ | Document’s unique ID |
| user\_id | BIGINT | PK, NN, FK | Internal spring boot field |
| document\_file | BLOB | NN | Binary data file |
| document\_content\_type | VARCHAR(168) | NN | File’s MIME type |
| document\_file\_type | VARCHAR(256) | NN | Document’s file extension |
| document\_type | VARCHAR(64) | NN | Document type |

*Figure 3. Entity Relationship Diagram*



## Data Sources

There are several external data sources used by the application to populate the denormalized tables. All of the external sources provide their information free of charge to the public.

* + Pubmed is a medical information repository by the National Institute of Health and National Library of Medicine, they provide a SPARQL endpoint for their data which is used by the VLOL application for allergies, this data comes from the MeSH dataset.
  + The Food and Drug Administration offers a download of their approved drugs dataset. This is used by VLOL for medications and allergies data.
  + Conditions data is supplied by the ICD dataset from the Centers for Medicare & Medicaid Services.
  + Vaccinations data is also supplied by the ICD dataset but for ease of access is retrieved from the ICD10data.com website.

# Component Design

The application is broken up into several packages:

* The base package includes the application start and any utility classes.
* The config package includes configuration for Spring Boot routes and authentication
* The controller package includes routes for the client to send a retrieve information
* The data package includes classes which get data from third party resources to insert into the database
* The model package includes the database schema used by hibernate to translate the database fields into Java friendly fields. Each field may also include additional constraints to provide server-side validation.
* The repository package includes classes to query the database, each class is built on top of a spring boot class provide an initial set of queries
* The service package includes classes that interface with the repository and may provide additional logic for database updates.
* Additionally, HTML templates as well as JavaScript and CSS are used in the resources package, these are made available via the Spring framework and are served via the routes in the controller package.

A typical route is formed by creating a route in a controller class. These routes are automatically made available by the Spring framework to send or receive data. After a client establishes a connection, either a resource is returned, or data is retrieved from a corresponding class in the service package. The service class will then interface with a corresponding repository class to get or insert data in the database. The repository class will use a corresponding class in the model package as an accepted input for updating the database or will return the data in the form of the model for data retrieval.

# Human Interface Design

The user interface is developed using HTML in combination with Thymeleaf which provides a structure to add data to the HTML. Frontend validation is also used to mimic validation done by the server to provide the user with a better experience and helpful error messages. Frontend validation is performed using the JavaScript Parsley validation library.

The layout of the interface starts with the homepage and several static pages for about and contact pages. To initiate the user workflow a user must either login or register to reach their data. The initial user workflow begins at the user menu. This menu contains links to data additional pages, such as the medication page or user account page which are used to add, modify, or remove data from a user’s profile. There are several additional pages such as the authorized user page and the QR code page which, respectively, can be used to add other users, who can update the current account, and print a QR code to view the account. User’s with authorization can be transferred from their menu to another user’s menu to update and view the authorizing user’s information.

Administrators and providers have a slightly different workflow after login. Like the user, they start at a menu that provides links to some capabilities such as adding accounts or scanning a QR code. However, their main workflow involves first searching the database for a user and accessing that user’s menu where they are then able to update or view the user’s information.