Deployment and Operations Guide (Runbook)

Virtual Letter of Life (VLOL) Application

Version 1.0

Sefanit Urgessa

Mohammed Allibalogun

Robert Garcia

Meron Getachew Debela

Charles Baisie

Augustin Mwamba

Jamal Bourne

SWEN 670 Software Engineering Project

University of Maryland Global Campus

Date: 07/21/2020

# REVISION HISTORY

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Description** |
| 07/03/2020 | 1.0 | Initial Project Plan Release |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[REVISION HISTORY ii](#_Toc46003210)

[1. Introduction 1](#_Toc46003211)

[1.1 Overview 1](#_Toc46003212)

[1.2 Document Purpose 1](#_Toc46003213)

[1.3 System Overview and Description 1](#_Toc46003214)

[1.4 Acronyms 2](#_Toc46003215)

[2. Build 3](#_Toc46003216)

[2.1 File structure 3](#_Toc46003217)

[3. Deployment 4](#_Toc46003218)

[3.1 Deployment Steps 5](#_Toc46003219)

[4. Common Tasks 5](#_Toc46003220)

[4.1 File Management 5](#_Toc46003221)

[5. Error Messages 5](#_Toc46003222)

[6. Bugs: 9](#_Toc46003223)

[7. Disaster Recovery/Service Level Agreement 9](#_Toc46003225)

[8. Software Operation Instructions 9](#_Toc46003226)

[8.1. Installation 9](#_Toc46003227)

[8.2. Start, Stop, Re-Start 10](#_Toc46003228)

[9. FAQs 11](#_Toc46003230)

[10. License 12](#_Toc46003231)

[11. Reference 12](#_Toc46003232)

1. Introduction

## Overview

This document provides guidance for the Deployment and Operations of the Virtual Letter of Life (VLOL) system. The Deployment and Operations Run Book (DORB) outlines procedures and operations that the system administrator or operator carries out. The DORB consists of important information VLOL application administrators and operations staff require to perform day-to-day processes, to respond to performance success, and emergency situations or any event that effects the application. The DORB provides specific details on hardware and software components of the equipment running the application, contact information, procedural information and scripts, and Back-out procedures and Recovery guidelines.

## Document Purpose

The scope of this document falls within the deployment and maintenance phase of software development life cycle (SDLC). The main purpose of this document is to provide administrators guidance on how to deploy the application; conduct daily operations; and respond to incident. When key components and/or information change due to software upgrade, department (customer) requirements or any other reason; the DORB must be updated to reflect the new system, application or environment.

The main audience for this runbook is the System Administrators, VLOL Project Leads, and application administrators. To properly execute the tasks contained in this runbook for VLOL, the audience should be familiar with the following concept: Java Spring Boot, JavaScript, Linux/Windows, SQL, MySQL, HTML5, Git, Client-side development, Boot Strap, Maven and CSS.

## **1.3** System Overview and Description

Essentially, the VLOL application is a web-based version of the Letter of Life. The Letter of Life in its current form is a paper form that provides first responders with a list of important medical information regarding a participant in need. Currently, this form is completed by a participant and placed inside the residence at a location that is easily accessible by EMS personnel. However, when/if an emergency occurs outside of participants’ home and if the participant is unresponsive, it is difficult for EMS personnel to have access to the participant’s medical information. Thus, the availability of the Letter of Life form virtually (VLOL) will resolve the current business problem. VLOL allows authenticated users to add, search, view, update, and delete basic medical information about themselves or others, depending on their level of authorization. Below is the list of authorized users defined in the VLOL application and access definition:

* Program Participant - View, update, or delete their individual account information, including password, security question, or security question answer.
* Medical Services Provider (e.g., EMS, ED Staff, etc.) - Search and view the medical information of program participants (this does not include the participant's password, security question, or security question answer).
* Patient Agent (Primary Care Physician, Power of Attorney (POA) agent, etc.) - Search, view, and update the medical information of program participants (this does not include the participant's password, security question, or security question answer).
* System Administrator - Add, search, view, update, or delete all authenticated users, their roles, or their account information (this does not include the participant's password, but the system administrator may authorize a password reset).
  + System Administrators may also add, search, view, update, or delete information contained in ancillary tables, such as allergies, conditions, medications, and roles.

Providers, agents, and administrators may search for program participants by entering their information in a search textbox or by scanning a Quick Response (QR) code that corresponds to the participant's identification number. Additionally, the VLOL application allows unauthenticated and unauthorized users (e.g., prospective participants, etc.) to learn about the application and register (i.e., add their account information).

## 1.4 Acronyms

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

* CSS – Cascading Style Sheet
* DDL - Data Definition Language
* DML - Data Manipulation Language
* DORB - Deployment and Operations Run Book
* EMS – Emergency Medical Service
* HIPAA – Health Insurance Portability and Accountability Act
* HTML - Hypertext Markup Language
* IVV - Integration, Validation, and Verification
* JSF - Java Server Faces
* MVC - Model-View-Controller
* NIST- National Institute of Standard and Technology
* ORM - Object Relational Model
* PII – Personally Identifiable Information
* RDBMS – Relational Database Management System
* RPO - Recovery Point Objective
* RTO - Recovery Time Objective.
* SA – System Administrator
* SDLC - Software Development Life Cycle
* VLOL - Virtual Letter of Life

1. Build:

The VLOL application was primarily written in Java and comprises of the following; Java 11 and Spring Boot (2.2.1), HTML5 and Thymeleaf (3.0.11), CSS3, JavaScript (ECMAScript 2018), Maven, and Tomcat or Glassfish Web Server.

* **Java 11 and Spring Boot (2.2.1)** - Spring Boot is an open source Java-based framework used to create a micro Service. Spring Boot automatically configures application based on the dependencies added by Maven.
* **Thymeleaf (3.0.11)** - Thymeleaf as a Java-based library/template uses syntax that maps server-side variables directly to common HTML-elements on the client-side.
* **HTML5** - HTML5 delivers almost everything user wants to do online without requiring additional software such as browser plugins. It is composed of HTML, CSS, and JavaScript.
  + HTML provides the structure of the web page display.
  + CSS provides the styling and presentation.
  + JavaScript provides responsiveness and interactivity to the web page.

For a successful built, the following must be in place;

* RDBMS
  + Built Database – following the ERD provided in SDD section 4.3
* Git repository
* Java 11 and Spring Boot
* HTML5 and Thymeleaf
* CSS3
* JavaScript

The build process thus includes;

* Downloading and installing the required services stated above.
* Fetching the code from the Git.
* Compile the code and check the dependencies.
* Run the automated unit tests.
* Link the library, files, and code accordingly.
* When everything successfully passes, it’s built an artifact and store.
* Achieve the build logs.
  1. File structure:

The current file structure for VLOL is critical for proper deployment and execution. The File Structure is a combination of representations of data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data. Figure 1 below is a detailed listing of the necessary folders and files for system operations.

Here is the description of the file structure captured in figure 1:

* *java.com.vlol* - Contains model and controller classes.
* *resources* - Contains view code, including HTML, CSS, JS, and images.
* *tests* - Contains unit tests and other Integration, Validation, and Verification (IVV) code.

├───nbproject├───src│ ├───main│ │ ├───java│ │ │ └───com│ │ │ └───vlol│ │ │ ├───config│ │ │ ├───constraint│ │ │ ├───controller│ │ │ ├───model│ │ │ ├───repository│ │ │ ├───service│ │ │ └───validator│ │ └───resources│ │ ├───static│ │ │ ├───css│ │ │ ├───images│ │ │ ├───img│ │ │ └───js│ │ └───templates│ │ └───admin

│ └───test

│ └───java

│ └───com

│ └───vlol

│ └───model

Figure 1. File Structure

1. Deployment:

The VLOL application can be deployed as a standalone or in a container such as Heroku or Docker. Deploying VLOL applications involves sending the application to Heroku using Git, which is a distributed version control system used to manage and version source code.

The predetermine requirement to run the application is for the SAs to have Git to pull the code, Tomcats or Glassfish web server to run the VLOL application, relational database (RDBMS) for persistent storage. All other productivity such WinSCP, VM or putty are outside the scope of this document.

## 3.1 Deployment Steps

To be able to run this application, the following prerequisites must be in place or followed:

* Download and create Git account
* Download JDK 11
* Download JRE 11
* Download/install Maven plugin
* Download/install Tomcat or Glassfish Web Server
* Have RDBMS system
* Pull from Git repository
* Checking in the code to the repository.
* Download the source code from the repository.
* Compile and build the application and generate jar/war from the same.
* Locate the generated war/jar in a common shared network location.
* Download the jar/war file and deploy to the target server
* Run unit, integration, system, and acceptance testing and then run the application.

1. Common Tasks:

The VLOL application structure is typically a client/server web application. The application does not require any administrative tasks to be performed while the application is running. The application does require System Admins or Developers to execute some tasks as needed to ensure the system is operational as specified by requirements. The following tasks will be performed:

* Daily review of log files to ensure the system is healthy and functioning as expected
* Weekly audit for pattern of activity
* Incident respond will occur as needed for any incident
* Patch management and update: Install all the latest patches and fixes for the application and upgrade application to the latest version if/when necessary. Patches and service packs should be installed during the specified scheduled maintenance windows.

4.1 File Management**:**

As provided in Section 2.1, VLOL application utilizes different files to discover and decompose the necessary tasks to display on the target viewport.

1. Error Messages**:**

The following are VLOL system responses and probable causes that a System Admin should be aware of:

**HTTP status errors**

* **400 Bad Request:**  
  Your request is invalid and/or not formed properly. You need to reformulate your request.
* **401 Not Authorized:**  
  Either you need to provide authentication credentials, or the credentials provided aren't valid.
* **403 Forbidden:**  
  We understand your request but are refusing to fulfill it. An accompanying error message should explain why.
* **404 Not Found:**  
  Either you're requesting an invalid URI or the resource in question doesn't exist (ex: no such user).
* **500 Internal Server Error:**  
  We did something wrong. We'll be notified and we'll look into it.
* **502 Bad Gateway:**  
  Returned if Intervals is down or being upgraded, or if the system is overloaded and API requests are being throttled.
* **503 Service Unavailable:**  
  Usually as a result of suspension, we are refusing to process this request. You \*may\* try again later.

**Summary java.lang.error**

* **Sr.No. 1: AbstractMethodError**  
  This is Thrown when an application tries to call an abstract method
* **Sr.No. 2: AssertionError:**

This is thrown to indicate that an assertion has failed.

* **Sr.No. 3: ClassCircularityError**

This is thrown when circularity has been detected while initializing a class.

* **Sr.No. 4: ClassFormatError**

This is thrown when the Java Virtual Machine attempts to read a class file and determines that the file is malformed or otherwise cannot be interpreted as a class file.

* **Sr.No. 5: Error**

This is an Error is a subclass of Throwable that indicates serious problems that a reasonable application should not try to catch.

* **Sr.No. 6: ExceptionInInitializerError**

These are the Signals that an unexpected exception has occurred in a static initializer.

* **Sr.No. 7: IllegalAccessError**

This is thrown if an application attempts to access or modify a field, or to call a method that it cannot access.

* **Sr.No. 8: IncompatibleClassChangeError**

This is thrown when an incompatible class change has occurred to some class definition.

* **Sr.No. 9: InstantiationError**

This is thrown when an application tries to use the Java new construct to instantiate an abstract class or an interface.

* **Sr.No. 10: InternalError**

This is thrown to indicate some unexpected internal error has occurred in the Java Virtual Machine.

* **Sr.No. 11: LinkageError**

The Subclasses of LinkageError indicate that a class has some dependency on another class.

* **Sr.No. 12: NoClassDefFoundError**

This is thrown if the Java Virtual Machine or a ClassLoader instance tries to load in the definition of a class and no definition of the class could be found.

* **Sr.No. 13: NoSuchFieldError**

This is thrown if an application tries to access or modify a specified field of an object, and that object no longer has that field.

* **Sr.No. 14: NoSuchMethodError**

This is thrown if an application tries to call a specified method of a class (either static or instance), and that class no longer has a definition of that method.

* **Sr.No. 15: OutOfMemoryError**

This is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

The **java.lang.Exceptions** - provided for different exceptions thrown under java lang package

* **Sr.No. 1: ArithmeticException**  
  This is thrown when an exceptional arithmetic condition has occurred.
* **Sr.No. 2: ArrayIndexOutOfBoundsException**  
  This is thrown to indicate that an array has been accessed with an illegal index.
* **Sr.No. 3: ArrayStoreException**  
   This is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.
* **Sr.No. 4: ClassCastException**  
   This is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
* **Sr.No. 5: ClassNotFoundException**  
   This is thrown when an application tries to load in a class through its string name using: The forName method in class Class.
* **Sr.No. 6: CloneNotSupportedException**  
   This is thrown to indicate that the clone method in class Object has been called to clone an object, but that the object's class does not implement the Cloneable interface.
* **Sr.No. 7: EnumConstantNotPresentException**  
   This is thrown when an application tries to access an enum constant by name and the enum type contains no constant with the specified name.
* **Sr.No. 8: Exception**  
   This is the class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.
* **Sr.No. 9: IllegalAccessException**  
   This is an IllegalAccessException is thrown when an application tries to reflectively create an instance (other than an array), set or get a field, or invoke a method, but the currently executing method does not have access to the definition of the specified class, field, method or constructor.
* **Sr.No. 10: IllegalArgumentException**  
   This is thrown to indicate that a method has been passed an illegal or inappropriate argument.
* **Sr.No. 11: IllegalMonitorStateException**  
   This is thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.
* **Sr.No. 12: IllegalStateException**  
   The Signals that a method has been invoked at an illegal or inappropriate time.
* **Sr.No. 13: IllegalThreadStateException**  
   This is thrown to indicate that a thread is not in an appropriate state for the requested operation.
* **Sr.No. 14: IndexOutOfBoundsException**  
   This is thrown to indicate that an index of some sort (such as to an array, to a string, or to a vector) is out of range.

1. Bugs:

Currently, the application has no outstanding bugs, errors, or exceptions that have been bypassed, and it is passing 100% of its unit tests. However, it is important to note that the application is in a very early stage of development.

1. Disaster Recovery/Service Level Agreement**:**

Disaster Recovery strategies encompasses business continuity factors such as recovery point objective (RPO) and recovery time objective (RTO). The developers of the VLOL utilized the free tier versions of tools and services for development, therefore, does not require any agreement of assurance or commitment from the service providers. This application is developed as a prototype-level application or free tier level application that can be adopted by any Fire Department/EMT.

Production-level infrastructure and access controls are out-of-scope for this development. Any deployed version of the application is maintained in the pre-built umgc Git repository provided by the summer 2020 DevOps team. There is no assurance for responsibility, availability, accessibility or scalability of the application beyond those offered by the respective vendors.

# Software Operation Instructions

The sections below in 8.1 and 8.2 will provide step-by-step instructions on how to install the VLOL software, how to start, how to stop, and how to re-start software if it is not properly shutdown.

## Installation

The VLOL application will run on any Linux or Windows-based operating system, real or virtual, provided the following requirements are met:

x86-64 processor

* 2 GB for Windows platforms, 1 GB for non-Windows platforms
* Java SE Runtime Environment (JRE) 11.x
* Java SE Development Kit (JDK) 11.x
* Maven 3.6.x
* MySQL 8.x
* HTTP Server: The VLOL application uses an integrated Tomcat Embedded Jasper web server, so there is no need for a separate standalone Tomcat instance.

To install the application, ensure the JAVA\_HOME, JDK\_HOME, JRE\_HOME, and MAVEN\_HOME environment variables are populated with the correct path to the JDK and Maven application directories. For example, in Windows:

* JAVA\_HOME: C:\Program Files\Java\jdk-11.x.x
* JDK\_HOME: %JAVA\_HOME%
* JRE\_HOME: %JAVA\_HOME%
* MAVEN\_HOME: C:\Program Files\apache-maven-3.6.x\bin

Afterwards, clone the following repository within the system's local repository folder:

git clone <https://github.com/garciart/SWEN670.git>

## Start, Stop, Re-Start

**Start VLOL**

Before starting, get the designated IP address and server port from the system administrator (e.g., localhost:5000, etc.) .

To start the application, open the application.properties file in the SWEN670\VLOL\src\main\resources folder and ensure the server port is set to "server.port=5000". This will allow the application to run on Heroku, as well as Debian, RHEL, and Windows.

Return to the VLOL application root folder (SWEN670\VLOL) and execute the following commands:

* mvn compile
* mvn spring-boot:run

Verify that the application is running by either entering the application IP address and server port in a web browser (e.g., <https://warm-hamlet-22515.herokuapp.com/>, etc.) or checking for a HTTP 200 OK response using a Client URL (cURL) command (e.g., curl [http://localhost:5000](http://localhost:5000/), etc.).

**Stop VLOL**

To properly shutdown the application, use a cURL POST with administrator privileges to activate the shutdown endpoint:

* curl.exe -u admin@vlol.gov:P@ssW0rd -v -X POST <http://localhost:5000/actuator/shutdown>

In the event of an emergency, such as an attack, look up the application's process ID number (PID) and shutdown using the taskkill command:

* netstat -ano | findstr 5000
* taskkill /PID <PID>

If necessary, terminate the application by adding the force flag ("/F") to the taskkill command:

* taskkill /F /PID <PID>

**Re-Start VLOL - if not Properly Shut Down**

If the application crashes or is shutdown forcefully, check the embedded Tomcat server log to determine the cause. These files are currently located at **C:\temp\logs\logs**.

After correcting any problems, restart the application using the steps described under "Instructions on How to Start the Software".

# FAQs

1. Do I need account to read about this application?

*No. Anyone who wants to read about the application can do so in the about page.*

1. Can I nominate a relative as an agent?

*Yes. Be aware that anyone you nominate will have access to your personal identifiable Information (PII).*

1. How do you secure my information?

*The application development followed the NIST standard. Roles and access are assigned based least privilege and need-to-know principles. Additionally, the application uses two factor authentication and the applications runs on HTTPS protocol.*

1. Can I decide not to include my Social Security Number during account creation?

*The SSN is needed to identify participant/users.*

1. Does the application follow HIPAA requirement?

*Yes.*

1. Will EMS team be able to access my information if I don’t have/create an account? *No.*
2. Can I give my username and password to anyone and still be secured?

*No, you or a trusted agent should be the only one aware of your username and password.*

1. Is there a way to ask for support?

*The current development team will provide the subsequent team with a Programmer's Guide, which will cover operations, maintenance, and retirement. The point of contact for this team is the Software Engineering (SWEN) department of the University of Maryland Global Campus.*

# License

The application code is currently covered by the Massachusetts Institute of Technology (MIT) License. This license is a permissive "copy right" license; the code may be reused without royalty or permission, but the developers are not liable for any claim, damages or other issues arising from its use. However, the developers hold the right to change this license in the future, and adopters of the application may restrict access to their code modifications through other licensing.

# Reference

How Heroku Works. (June 09, 2020). Heroku. <https://devcenter.heroku.com/articles/how-heroku-works#:~:text=When%20you%20create%20an%20application%20on%20Heroku%2C%20it,heroku%20remote%20instead%3A%20%24%20git%20push%20heroku%20master>

Java.lang.Exceptions.(n.d). Tutorialspoint. Retrieved on July 16, 2020 from <https://www.tutorialspoint.com/java/lang/java_lang_exceptions.htm>

Java.lang.Errors.(n.d). Tutorialspoint. Retrieved on July 16, 2020 from <https://www.tutorialspoint.com/java/lang/java_lang_errors.htm>