2020 CSLabs Extension Project

Software Architecture Specification Report

****

Submitted to:

Dr. Ronald B. Finkbine

Professor of Computer Science

Presented by:

Yiliang Lu, Junet Bello, and Copper Martin

November 8th, 2020

## Table of Contents

[1. Proposed software architecture](#_Toc52873031) 1

* [Overview 2](#_Toc52873032)
* [subsystem decomposition 2](#_Toc52873032)
* [hardware/software mapping 2](#_Toc52873032)
* [persistent data management 2](#_Toc52873032)
* [access control and security 2](#_Toc52873032)
* [global software control 2](#_Toc52873032)
* [boundary conditions 2](#_Toc52873032)

[2. Basic components 2](#_Toc52873033)

* [Use Cases 2](#_Toc52873032)
* [functions 2](#_Toc52873032)
* [Triggers 2](#_Toc52873032)
* [data stores 2](#_Toc52873032)
* [data flows 2](#_Toc52873032)
* [data elements 2](#_Toc52873032)
* [processors 2](#_Toc52873032)
* [data storage 2](#_Toc52873032)
* [data connections 2](#_Toc52873032)
* [external entities 2](#_Toc52873032)

[3. Key Personnel and Contribution Breakdown](#_Toc52873039) 5

4[. WORKS CITED](#_Toc52873039) 5

This page is intentionally left blank.

# **1. Proposed Software Architecture**

### 1.1 Overview

CSLabs is a virtual lab learning environment created and operated by the Indiana University Southeast (IUS) Computer Security Group (CSG). It is used by IUS faculty and students to practice computer security and learn other aspects of computer science using virtual machines (VM).

The CSLabs 2020 Capstone project is an extension of the previous year’s Capstone projects. The extension project does not aim to overhaul the existing system architecture. The project objectives are to add new functionalities, system quality enhancements, and user experience improvement to the CSLabs Webapp and backend application.

According to according to Clifton et al. (2019), the CSLabs system is comprised of several components: the frontend Webapp, the backend application server, the MariaDB database server, and the Proxmox virtual environment cluster.

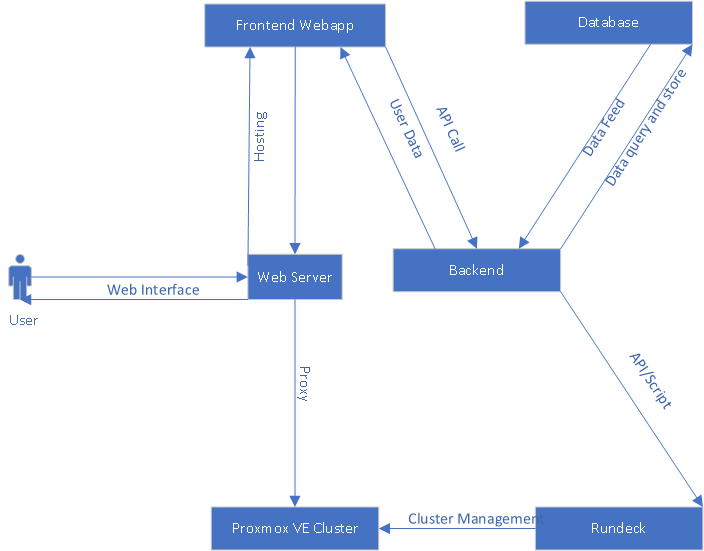
The frontend webapp is bootstrapped with Facebook’s React JavaScript library. The webapp uses Create-React-App with Typescript for building the user interfaces. According to Gallavin et al. (2019), this will provide maximum type safety in the project.

The backend application server is built with the DotNet Core framework in C#.

Using the DotNet Core framework allows the project to host the C# application on a Linux server. The backend application server receives API calls from the frontend Webapp for data and authentication. According to Gallavin et al. (2019), this setup provides maximum type safety for validating requests.

The previous project developers have advised that the backend application server also uses Rundeck open source automation service to issue scripted commands to the Proxmox virtual environment cluster. Proxmox will be running, and housing VMs that are controlled by the Rundeck scripts

**1.2 Subsystem Decomposition**

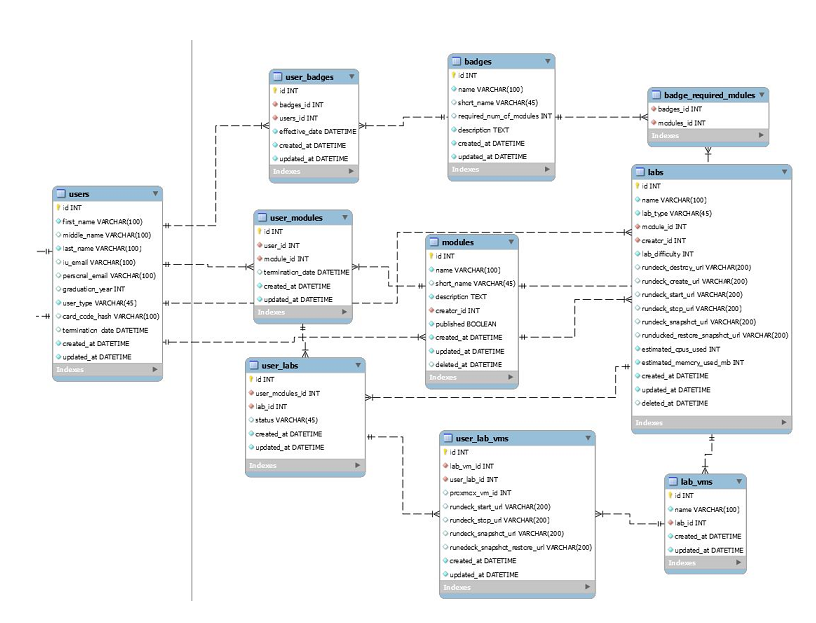


**1.3 Hardware / Software Mapping**

### 

### 1.4 Persistent Data Management

According to Gallavin et al. (2019), MariaDB is used as the persistent database. MariaDB provides a SQL interface to store and retrieve the application state. The database will reside on a dedicated server where the web server can access it with low latency. The application’s entities are shown below in the database diagram created by Gallavin et al. (2019). Most of the entities follow standard CRUD operations. The user\_modules, user\_labs, and user\_vm’s show instances of each entity which are created when a user starts the module.



### 1.5 Access Control and Security

According to Gallavin et al. (2019), each user is authenticated using a username and password. The default user group is user. Only an admin can promote users to higher permission groups.

|  |
| --- |
| ***User*** |
| Can start a module |
| Can change account email address |
| Can change account password |
| Can start up a VM in an owned lab |
| Can shut down a VM in an owned lab |
| Can create a snapshot of a VM in an own lab |
| Can restore a snapshot of a VM in an own lab |

|  |
| --- |
| ***Staff*** |
| Can create private modules |
| Can edit owned private modules |
| Can set a start and end date for owned private modules they own. |
| Can share owned modules using links |

|  |
| --- |
| ***Admin*** |
| Can do everything that staff and user can |
| Can create public modules |
| Can promote users to staff or admin |

### 1.6 Global Software Control

According to Gallavin et al. (2019), the control flow of the backend is asynchronous. It uses a thread pool to run tasks in the background and start on the next line of code after the asynchronous call. This allows for efficient handling of API requests without blocking new connections.

The front end uses asynchronous and event-driven control flows. Asynchronous code flows as described above are used for network requests. However, JavaScript/Typescript is single threaded, and there is no threadpool. Asynchronous methods are crucial to keeping the UI smooth. Button clicks and input changes are tracked using event based code flow. When the login button is pressed, it fires an event that initiates the form checking process.

### 1.7 Boundary Conditions

According to Gallavin et al. (2019), the Backend system can be started up using the Dotnet CLI runtime. This is built into a systemd service that can be started and stopped. Running the command dotnet ef database update will update the database’s schema to match the version deployed. This will run automatically when a new version of the system is deployed. Shutting down the backend can be triggered with the standard systemd stop command. The backend application will utilize logstash to log all unhandled errors into a database for logs. From there we can analyze the errors from a graphical UI enabling enhanced debugging. The front end will handle errors the same way. Any expected error like unauthorized or bad request is handled gracefully in the frontend to tell the user what they did wrong.

As with any software, it should be frequently updated or security threats might go unfixed. The frontend and backend dependencies should be updated at least every 6 months. When replacing servers, CSG will need to export the MariaDB database using MySQL workbench and import it into the new database server. The backend will store profile pic uploads on disk at /var/www/cslabs-backend/assets/img/profile/, A new version of the application must be built to the /var/www/cslabs-backend directory. The frontend is stored at /var/www/cslabs-webapp/. The NGINX configuration should also be copied over at /etc/nginx/sites-enabled/default.

# **2. Basic components Project Description**

The basic component sections below were originally written by Gallavin et al. (2019), TCA has made small updates and corrections to the original writing.

### 2.1 Use Cases

Teachers need a way for their students to have easy access to a working environment. A student may not have a capable machine that can run a full-fledged VM. The student can use our lab environment to work on their assignment regardless of the computer they are using.

Public users will come to this site to learn. There will be a few free public modules that users can learn from. This will gain the product popularity to pave the way for future paid modules.

### 2.2 Functions

**a. Log In**

The login request is a stateless function that returns your authentication token if valid credentials are given or access denied if the credentials are not valid.

**b. Navigating to public pages**

Navigation to any public page only takes the URL as input and returns the page without any contextual knowledge. Given the public page URL, you will always get the correct page.

**c. Front end components**

A lot of the frontend components are just pure functions that accept arguments and return UI elements. That gives the same output given the same input. Other components have state, mostly pages that are not functions.

### 2.3 Triggers

For teachers, we would say it would be a time trigger. Students take too long to set up the software locally which puts time pressure on the assignment. With this product, students can quickly jump into the concepts without muddling with configuration.

### 2.4 Data Stores

The MariaDB service is used to store the data. At rest the data will sit there until it is deleted or modified by the backend. This will serve as a central store for the backend. The Proxmox server will also hold data about the VM’s statuses. This will sit on the Proxmox server’s storage.

### 2.5 Data Flows

There is a data flow between the webserver and the database server. The web server will need to retrieve entities from the database. Once retrieved from the database, the data flows over http in the form of a response to the frontend where the UI is then updated. Data also flows from the backend to the Rundeck server and then to the Proxmox server when managing VMs.

### 2.6 Data Elements

The following definitions were originally defined in the previous SRS report by Gallavin et al. (2019). The terms have been updated by TCA, and the list is sorted alphabetically

|  |  |
| --- | --- |
| Badge | A reward given to a user for completing certain requirements |
| BadgeRequiredModule | Defines a required module for a badge |
| Lab | A specific lab in a learning module that has one or more VMs |
| LabVm | A VM template for a lab |
| Module | A learning module that houses a collection of labs |
| User | A user of CSLabs system |
| UserBadge | A user’s instance of a badge when they have earned it. |
| UserLab | A user’s instance of a lab |
| UserLabVm | A user’s instance of a VM that can be run |
| UserModule | A user’s instance of a learning module |

### 2.7 Processors

There are several processors in this system. The browser on the user’s computer processes the frontend code. The backend server is processing the requests using .Net Core. The users are going to be processing step by step guides and performing actions on the VMs.

### 2.8 Data Storage

The database storage will be on a VM with daily snapshots. The disk is stored on the infrastructure's Proxmox instance which uses RAID to increase redundancy. Hard drives are used as the physical storage device.

### 2.9 Data Connections

There will be a data connection between the user’s device and the web server using HTTP as the transport protocol. The backend then has a data connection with the database during database calls using TCP with MariaDB protocol. Communication between the backend and the Rundeck server will use HTTP. The Rundeck server also uses HTTP to call the Proxmox API. The application will utilize mailgun to send emails via SMTP.

### 2.10 Actors/External Entities

The system must interface with namespace for access to the domain name. The system will have to interface with teachers to create modules and labs for their students. The software will also interface with students / public users to present labs.

**3. Key Personnel and Contribution Breakdown**

|  |  |  |
| --- | --- | --- |
| Position | Name | Contribution |
| Project leader | Lu, Yiliang | Conduct primary and secondary research; draft and edit reports; coordinate events and meetings; establish a liaison with external parties and advisors. |
| Full Stack Developer | Bello, Junet | Create and manage backlogs; project feasibility consulting; primary developer for the CSLabs backend; alternate project POC |
| Full Stack Developer | Martin, Cooper | Primary developer for the CSLabs web-app frontend; alternate event coordinator; unit testing |

Works Cited

Clifton, Zac et al. "CS labs Infrastructure Hardware Requirements Specication for CSlabs Operations and Application."  15 Oct. 2019, <https://github.com/ius-csg/CSLabs-Capstone-Documentation/tree/master/cslabs-Infra-2019-2020/REPORTS>.  27 Oct. 2020

Gallavin, Jason et al. " CS Labs – Web Software Architecture Specification." 14 Oct. 2019, <https://github.com/ius-csg/CSLabs-Capstone-Documentation/tree/master/cslabs-web-2019-2020/DesignDocs>.  27 Oct. 2020