2020 CSLabs Extension Project

Software Detailed Design Report

****

Submitted to:

Dr. Ronald B. Finkbine

Professor of Computer Science

Presented by:

Yiliang Lu, Junet Bello, and Copper Martin

December 6th, 2020

## Table of Contents

[1. Data design](#_Toc52873031) 1

* [Overview 2](#_Toc52873032)
* [Data structure 2](#_Toc52873032)

[2. architecture design](#_Toc52873033) 3

[3. interface design](#_Toc52873033) 4

* [Rundeck Api Interface](#_Toc52873032) 4
* [proxmox api Interface](#_Toc52873032) 5

[4. Procedural design](#_Toc52873033) 6

* [account Registration](#_Toc52873032) 6
* [account Login](#_Toc52873032) 8
* [Create Lab](#_Toc52873032) 10
* [Startup Lab](#_Toc52873032) 11
* [Shut Down Lab 12](#_Toc52873032)
* [Destroy Lab](#_Toc52873032) 13

[5. Key Personnel and Contribution Breakdown](#_Toc52873039) 15

6[. WORKS CITED](#_Toc52873039) 16

This page is intentionally left blank.

# **1. Data Design**

### 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).

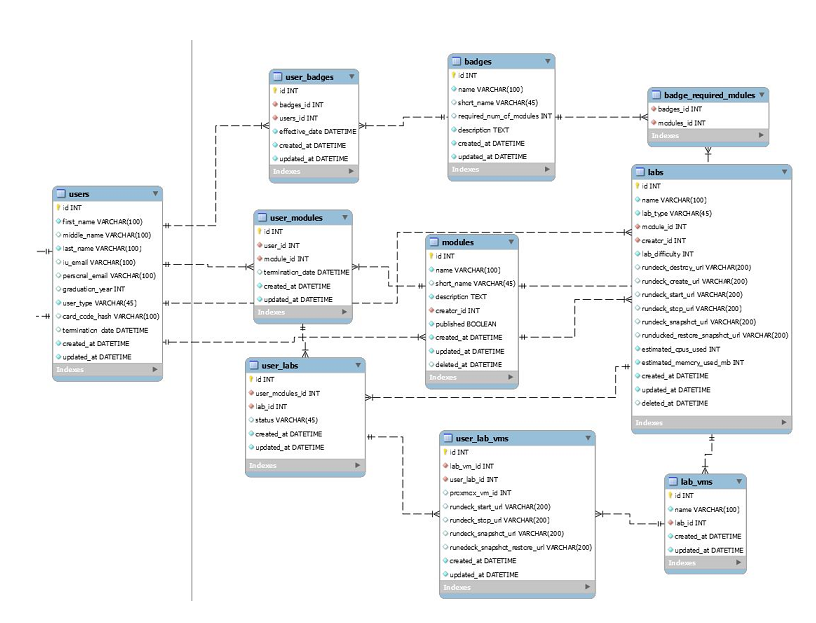
The CSLabs 2020 Capstone project is an extension of the previous year’s Capstone projects. The extension project does not aim to overhaul the existing system architecture. The project objectives are to add new functionalities, system quality enhancements, and user experience improvement to the CSLabs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

### Data Structure

Gallavin et al.’s team (2019) created the original Entity Relationship diagram below to describe the system data structure. The system creates multiple classes to represent the below referenced tables with the C# HTTP API. Typescript Interfaces in the React frontend are also created to match those tables. A List property exists on the C# class wherever there is a data design relationship.

According to Gallavin et al (2019), CSLabs uses modules as the main system entities. A module contains a list of labs that can be completed much like chapters in a book. Each lab contains a list of virtual machine (VM) definitions that should be used when the lab is started. Badges are awarded when certain modules and other requirements are met. A badge has an attribute called required\_num\_of\_modules that is set if the correct badge is awarded when a certain amount of modules are completed. The badge entity also has a relation to badge\_required\_modules that allows a badge to be awarded when a user completes a certain module(s).

The other concept used in CSLabs is the user entity. The user entity has many attributes shown in the diagram that are mostly used by the infrastructure team. Users relate to user\_badges which are awarded badges that a user owns. The user also has user\_modules, user\_labs, user\_lab\_vms, which are instantiations of the module. These instantiation tracks progression, and lab environment state. For data structures, dictionaries are used on the frontend application to quickly look up entities by Id. The backend uses Lists to store relations since all of the hard work is handled by the database.



**2. Architecture Design**

According to Gallavin et al (2019), the architecture of the CSLabs is consisted of several parts. The frontend is built using React in Typescript. It serves as an application running in the browser that connects the backend using AJAX requests. Data transport between the frontend and the backend utilizes the JSON format, and it is transmitted using the HTTP protocol.

The C# backend includes a Rundeck API interface that also uses JSON and HTTP to communicate. The Rundeck jobs provide the following actions for the system, Start a VM, Stop a VM, Clone a new VM from a template, and destroy a VM. When the user presses the button to start the lab, that action is translated into the instantiation of a lab using multiple Rundeck API calls. A small python command line interface is then called to invoke Proxmox API calls to carry out the actions. This design decision was made to create easy centralized tracking of tasks ran in the system.

**3. Interface Design**

According to Gallavin et al (2019), CSLabs utilizes the Rundeck API interface and the Proxmox API interface in the following manner.

**Rundeck API Interface**

Run Job

Runs a job via an ID. This is used to execute commands on the Proxmox API.

Run job and get output

Runs a job and gets the output. This is useful for commands that require a response. Commands like Create VM via template require a response of the ID of the VM so the database can track the VM.

Connect with Host, username, and password

The software interface requires the host, username, and password to connect to the given Rundeck host.

**Proxmox API Interface**

Get VM status

Retrieves the power status of the VM. This is used to display the power indicator on the front end.

Create VM via template

When a user starts a lab, this command is executed to create a new vm from a template VM and return the ID so the VM can be deleted later on.

Shut down VM

This command is executed to shut down a VM given it’s ID. This will be called if the user closes the browser.

Start VM

When a user starts a lab, this is called after creating the VM. The user can also manually start the VM from the frontend.

Destroy VM by ID

When a lab should be removed, this command is issued to remove the VM. This will serve to save server space when the user is finished.

Get Ticket

For the frontend to be able to display a VM in a webpage, noVNC is used with a

websocket. To authenticate that websocket, a ticket is required which can be retrieved.

Connect Websocket

After the ticket is retrieved a websocket connection is requested from the frontend which is then proxied to Proxmox. The websocket can then be used securely to interact with the VM.

**4. Procedural Design**

According to Gallavin et al (2019), CSLabs utilizes that following procedural design to facilitate the system functionality.

**Account Registration**

For a user to gain access to the system, they will need to register for an account.

Input

* First Name
* Last Name
* School Email
* Personal Email
* Graduation Year
* Phone Number
* Password
* Confirm Password

Output

A base64 encoded JWT token to be used on all API requests.

Component

A user is created in the database with the hashed password and the attributes assigned. A JWT token is then generated and sent to the client.

Constrains

This process should be very quick, less than 100ms response time. Either the School email or personal email is required. Either can be used to login to the account. The remaining required fields are First Name, Last Name, Password, and Confirm Password.

Process

*if(request.Password != request.ConfirmPassword)*

*Return “Passwords do not match”*

*if(!request.FirstName)*

*Return “First Name is required”*

*if(!request.LastName)*

*Return “Last Name is required”*

*if(!request.SchoolEmail && !request.PersonalEmail)*

*Return “School email or Personal Email is required”*

*user = register(request.all())*

*Return user.makeToken()*

**Account Login**

For a user to gain access to the system once they are registered, they will need to log into their account

Input

* Email
* Password

Output

A base64 encoded JWT token to be used on all API requests. The system will display validation message if error occurs.

Component

The password is hashed and then searched in the database along with the email. If a user object is found, a JWT token will be generated to be served as an authentication token.

Constrains

This process should be very quick, less than 100ms response time. Either a school email or a personal email is required.

Process

*if(!request.Password)*

*Return “Password is required” if(!request.Email)*

*Return “Email is required”*

*user = DB*

*.Where(u => u.PersonalEmail == request.Email || u.SchoolEmail == request.Email )*

*.Where(u => u.Password == hashed).First()*

### *if(!user)*

### *Return “Incorrect credentials”*

### *Return user.makeToken()*

**Create Lab**

When the user starts a lab, the create lab procedure will start. This procedure will clone the template VMs for that lab and start the VMs.

Input

The template VM IDs has a list of arguments.

Output

The list of instance ids is used for the cloned VMs.

Component

Converts input to output by looping through the IDs and calling the clone template command on Proxmox. Each call to Proxmox will return the instance id.

Constraints

### The design is constrained by the Proxmox API, a lab must start in less than a minute. For optimal performance, multithreading might be needed to make parallel calls.

Process

*templateIds= getTemplateIdsFromArguments()*

*instanceIds = [ ]*

*foreach(templateIds as templateId):*

*instanceIds[ ] = cloneTemplate(templateId)*

*print(JSON.dump(instanceIds))*

**Startup Lab**

After a lab has been shut down, the user may need to start again where they left off. This procedure resumes a lab.

Input

The VM IDs are given as arguments. These will be the VMs to start.

Output

There will be a string response of “success”.

Component

Converts input to output by looping through the VM IDs and calling the Proxmox start API command for each VM ID.

Constraints

The design is constrained by the Proxmox API; a lab must start in less than a minute. For optimal performance, multithreading might be needed to make parallel calls.

Process

*instanceIds= getInstanceIdsFromArguments()*

*foreach(instanceIds as instanceId):*

*startVm(instanceId)*

*print(“sucess”))*

**Shut Down Lab**

When a user strops a lab without completing the lab objective, the lab module will shut down to save memory and CPU resources.

Input

The input is the lab id given by the frontend when the user presses the shutdown button. This will send a list of VM IDs to Proxmox.

Output

There will be a string response of “success”.

Component

Converts input to output by looping through the VM IDs and calling the Proxmox shutdown API command for each VM ID.

Constrains

A lab should shutdown in approximately 30 seconds. Parallel programming can be used to speed up this process.

Process

*instanceIds= getInstanceIdsFromArguments()*

*foreach(instanceIds as instanceId):*

*shutdownVm(instanceId)*

*print(“sucess”)*

**Destroy Lab**

When a user completes with a lab, the system should destroy the lab object to save memory and CPU resources.

Input

The input is the lab id given by the frontend when the user presses the destroy button or if the user has left the lab unattended for a week. This will send a list of VM IDs to Proxmox to destroy.

Output

There will be a string response of “success”.

Component

Converts input to output by looping through the VM ids and calling the Proxmox delete API command for each VM ID.

Constrains

A lab should be able to be destroyed in around 30 seconds. Parallel programming can be used to speed this up.

Process

*instanceIds= getInstanceIdsFromArguments()*

*foreach(instanceIds as instanceId): destroy(instanceId)*

*print(“sucess”)*

**5. 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 Details."  29 Oct. 2019, <https://github.com/ius-csg/CSLabs-Capstone-Documentation/tree/master/cslabs-Infra-2019-2020/REPORTS>.  27 Nov. 2020

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