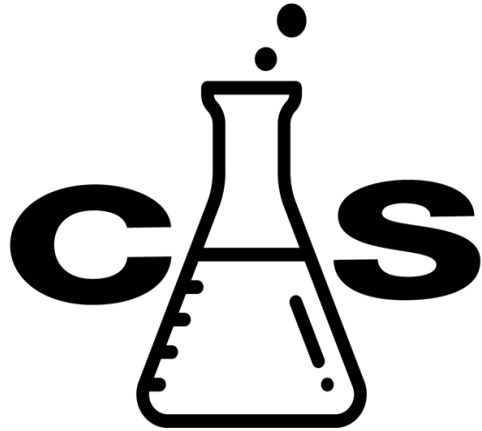


CSLabs Project Extension, Team 1 – 2021–2022

Software Requirements Specification Report



Project Team:

Jason Henry, Justin Hurst, Zaid Hussain

October 26th, 2021

Faculty Advisor:

Dr. Ronald B. Finkbine

Associate Professor of Computer Science

1. Data Design	4
2. Architecture Design	6
3. Interface Design	7
3.1 Rundeck API Interface	7
3.2. Front End Interface	9
4. Procedural Design	12
Create Lab	12
Shut Down Lab	13
Startup Lab	14
Destroy Lab	15
Register	16
Login	17

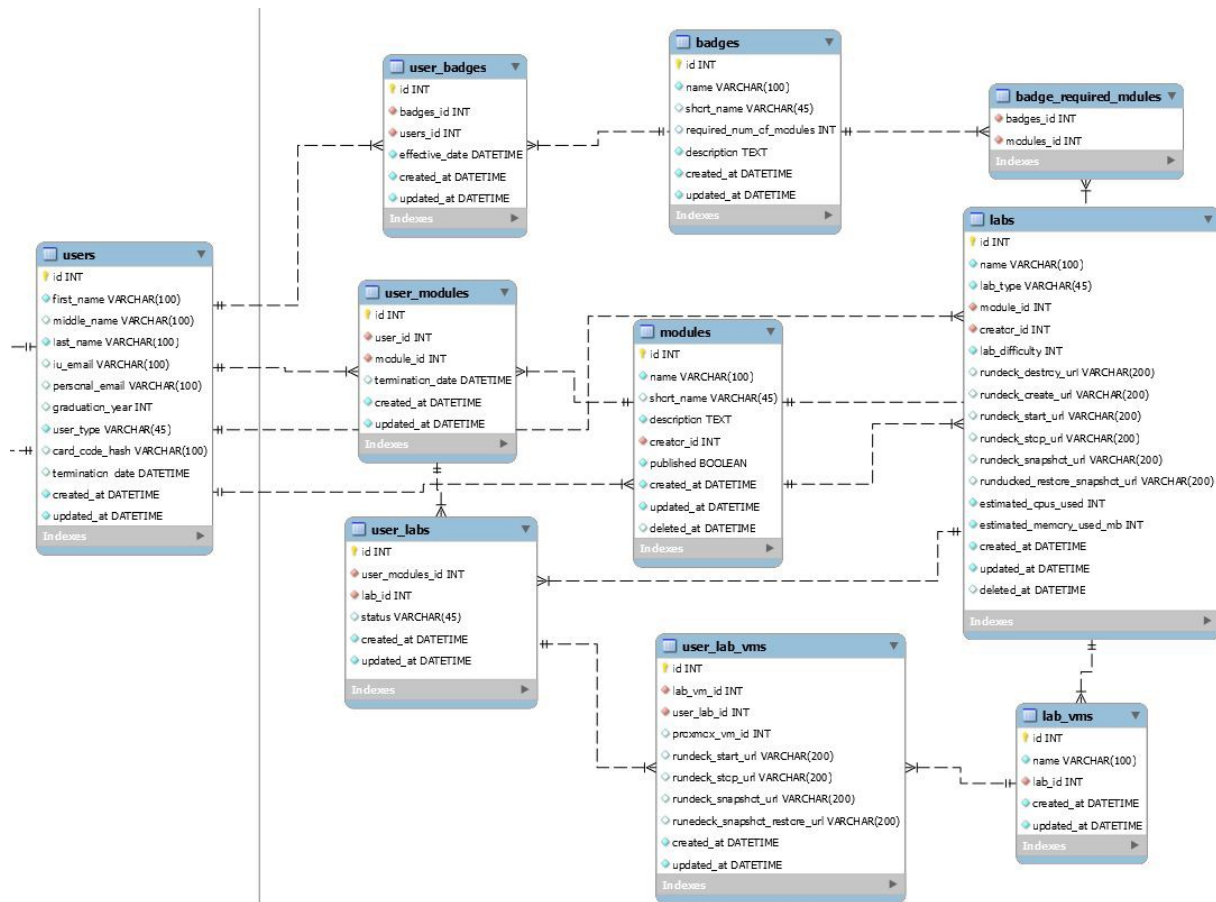
Key Personnel Information and Contribution Breakdown

Zaid Hussain – Project Leader, contributes to the analysis reports, the frontend and backend by implementing additional, essential functionalities

Justin Hurst – Full Stack Developer, contributes to the frontend and backend by implementing additional, essential functionalities

Jason Henry – Full Stack Developer, contributes to the frontend and backend by implementing additional, essential functionalities

1. Data Design ^[1]



The Entity Relationship Diagram above describes the structure of CSLabs. Classes will be created that represent these tables in the C# HTTP API. Typescript Interfaces in the React frontend will be created to also match these tables. Where there is a relationship, a List property exists on the C# class.

The main entity of this system is a module. A module contains a list of labs that can be completed like chapters in a book. Each lab contains a list of virtual machine definitions that should be used when the lab is started. Badges can be awarded when certain modules and other

requirements are met. A Badge has an attribute called `required_num_of_modules` that is set if the badge is awarded when a certain number of modules are completed. The Badge entity also has a relation to `badge_required_modules` that allows a badge to specify which module(s) a user has to complete in order to be awarded.

The next important entity in the system is the user. The user 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 instantiations track progression of the user, 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 ^[1]

The architecture of the system consists of several parts. The front end built using React in Typescript will serve an application running in the browser that connects the backend using AJAX requests. JSON is used as the transport format. Both the frontend and backend translate to and from JSON and data objects. Using HTTP as the protocol to transmit the JSON, the frontend and backend can communicate allowing the data to display.

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 multiplerundeck 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 ^[1]

3.1. Rundeck API Interface

- Run a job
 - Runs a job via an ID. This is used to Execute commands on the Proxmox API
- Connect with Host, username, and password.
 - The software interface requires the host, username, and password to connect to the given rundeck host.
- Runs a job and produces 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.

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 templatevm 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-up 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.

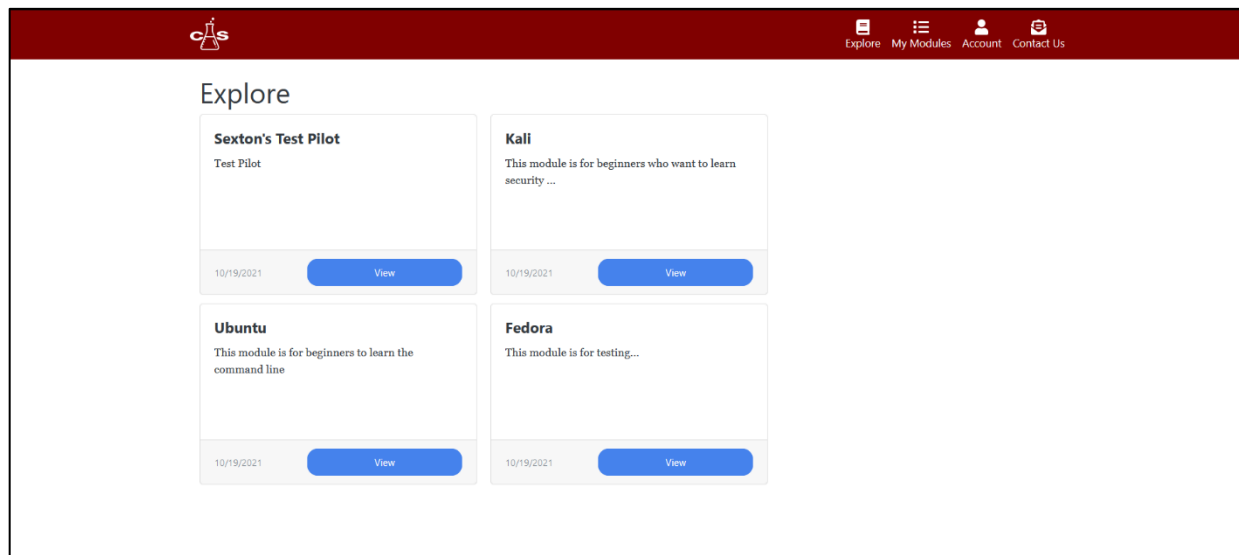
- Destroy VM by ID
 - When a lab should be removed. This command is issued to remove the VM. This will serve to save server space when the user is finished.

- 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

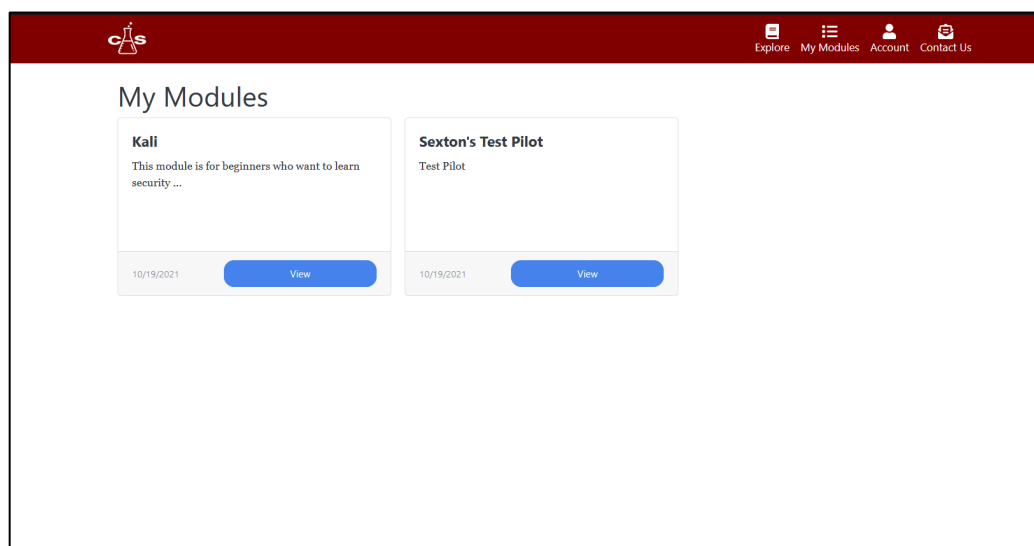
3.2. Front End Interface

The front end supports the following web browser IE 11, Edge, Firefox, Chrome, Safari. Below is the current version of the frontend.



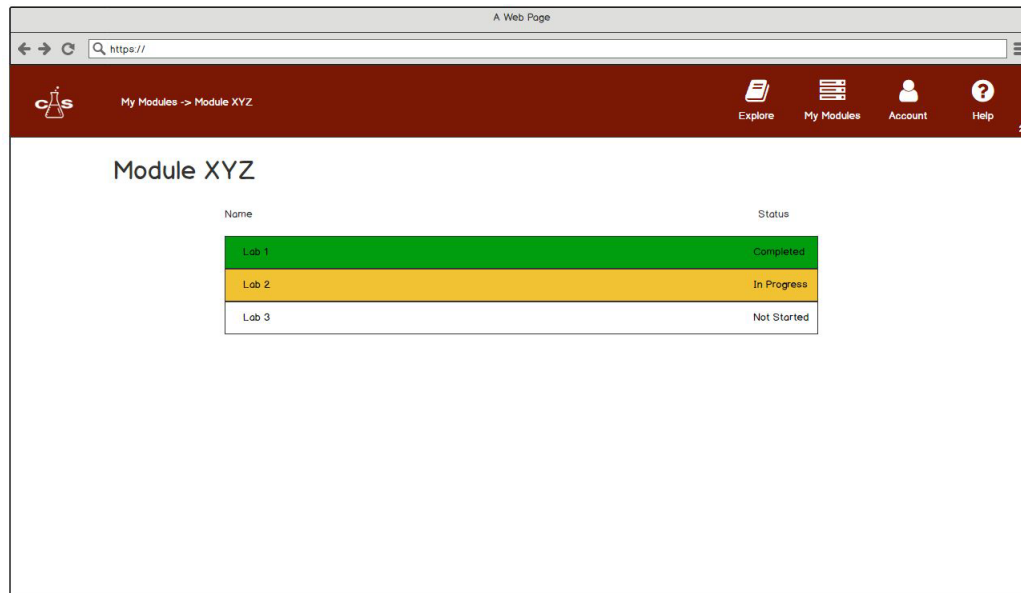
Here is where the user will select a module which they are interested in.

My Modules Page



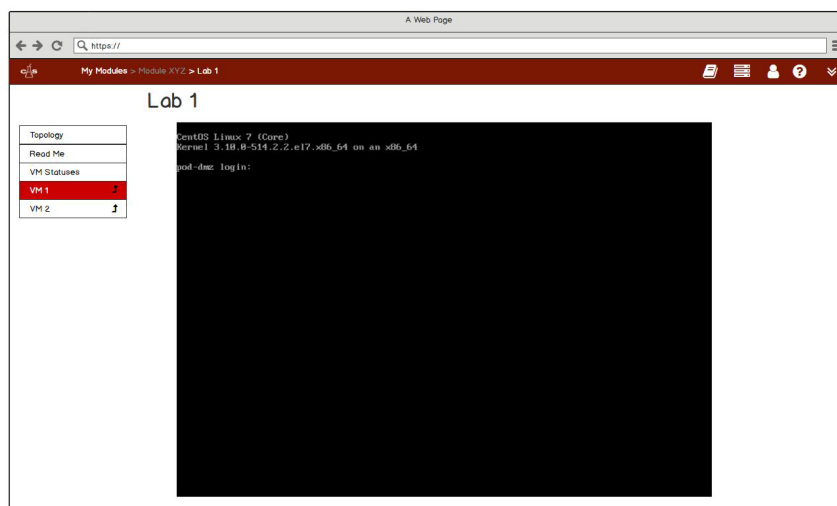
Here the user can browser modules they have started.

Module Page



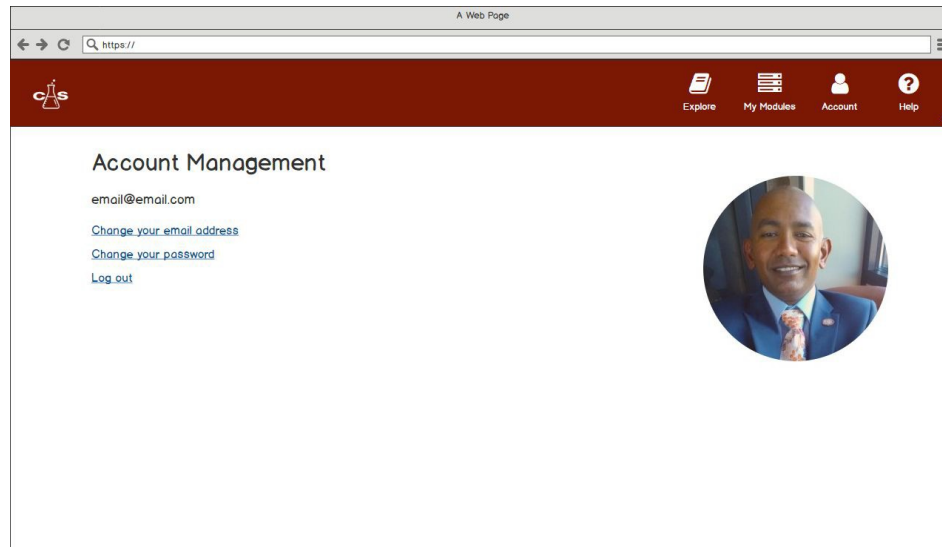
The user will select which lab they want to start. This page also shows your progression through the module.

Virtual Machine Page



Here the user can interact with virtual machines and complete the lab.

Account Management Page



Here the user can change their email address, and password. When you press either link, the respective content shows in place. The user can upload a new photo to replace the current one.

Account Management

Current Password

New Password

Retype New Password

Passwords don't match

Back

Change Password

Account Management

New Email

Retype New Email

The emails don't match

Back

Change email

4. Procedural Design ^[1]

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 as a list of arguments.

Output

The list of instance ids for the cloned VMs.

Component

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

Constraints

Must be able to start a lab in less than a minute. This design is constrained to the proxmox's API so 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))
```

Shut Down Lab

When a user is finished with a lab or is finished for the day. The lab should 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 be able to shut down in around 30 seconds. Parallel programming can be used to speed this up.

Process

```
instanceIds= getInstanceIdsFromArguments()  
foreach(instanceIds as instanceId):  
    shutdownVm(instanceId)  
print(“sucess”)
```

Startup Lab

After a lab has shut down, the user might need to start again where they left off. This procedure turns a lab back on.

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

Must be able to start a lab in less than a minute. This design is constrained to the proxmox’s API so for optimal performance, multithreading might be needed to make parallel calls.

Process

```
instanceIds= getInstanceIdsFromArguments()  
foreach(instanceIds as instanceId):  
    startVm(instanceId)  
print(“sucess”))
```

Destroy Lab

When a user is permanently finished with a lab, it should be destroyed 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.

Constraints

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”)
```

Register

For a user to gain access to the system and create and interact with their personal modules, 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 are 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()
```

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. Validation message on error.

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.

Constraints

This process should be very quick, less than 100ms response time. Either the School email or personal email are 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()
```

References

1. Gallavin, Jason et al. "CS Labs – Web Software Requirements Specification." 1 Oct. 2019, <https://github.com/ius-csg/CSLabs-Capstone-Documentation/tree/master/cslabs-web-2019-2020/DesignDocs>. 27 Sept. 2021
2. Clifton, Zac et al. "CS labs Infrastructure Hardware Requirements Specification for CS labs Operations and Application." 14 Oct. 2019, <https://github.com/ius-csg/CSLabs-Capstone-Documentation/tree/master/cslabs-Infra-2019-2020/REPORTS>. 27 Sept. 2021