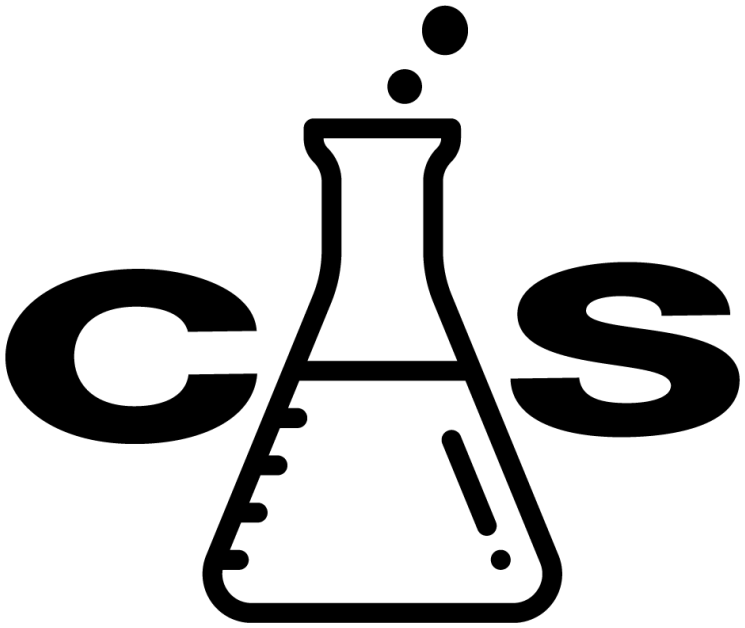


CS Labs – Web

Software Architecture Specification

Group Leader: Jason Gallavin

Members: Barel Musenga, Chase Benston

Oct 14, 2019

Contribution Breakdown and Key Personnel Information

Jason Gallavin – Project leader, contributes by creating reports, deciding architecture and larger project decisions.

Barel Musenga – Full Stack Developer, contributes by implementing the front end and back end, and parts of the lab automation.

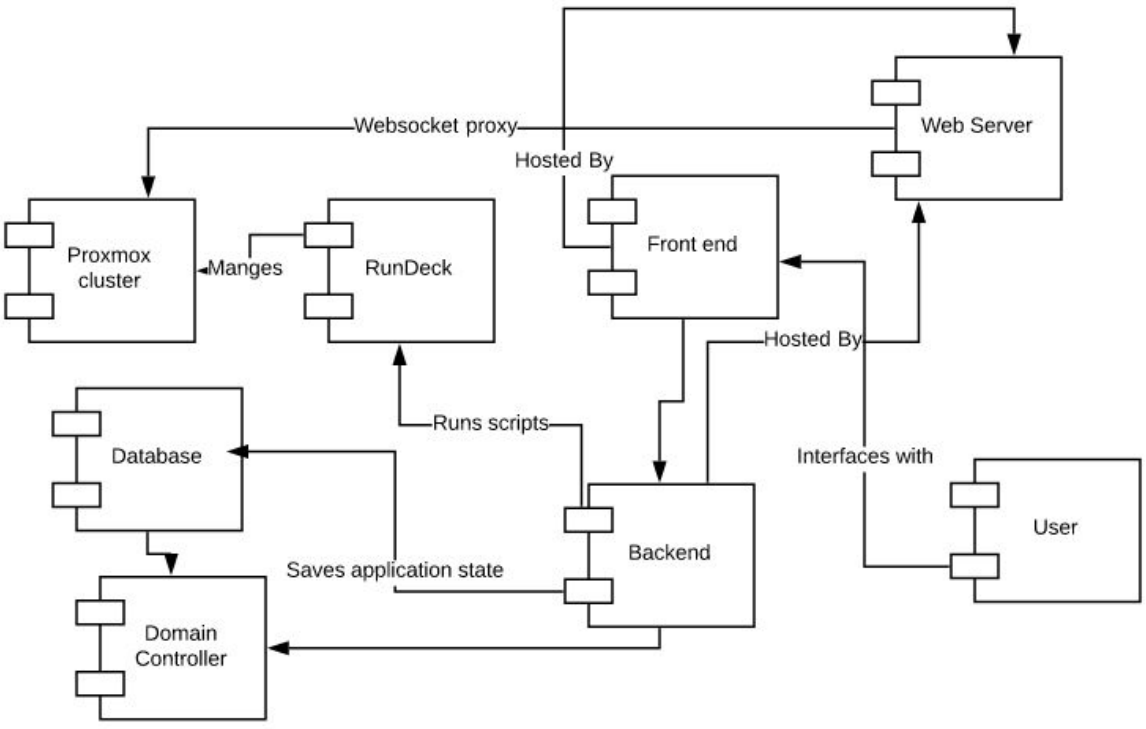
Chase Benston – Full Stack Developer, contributes by implementing the front end and back end, and parts of the lab automation.

1. Proposed software architecture

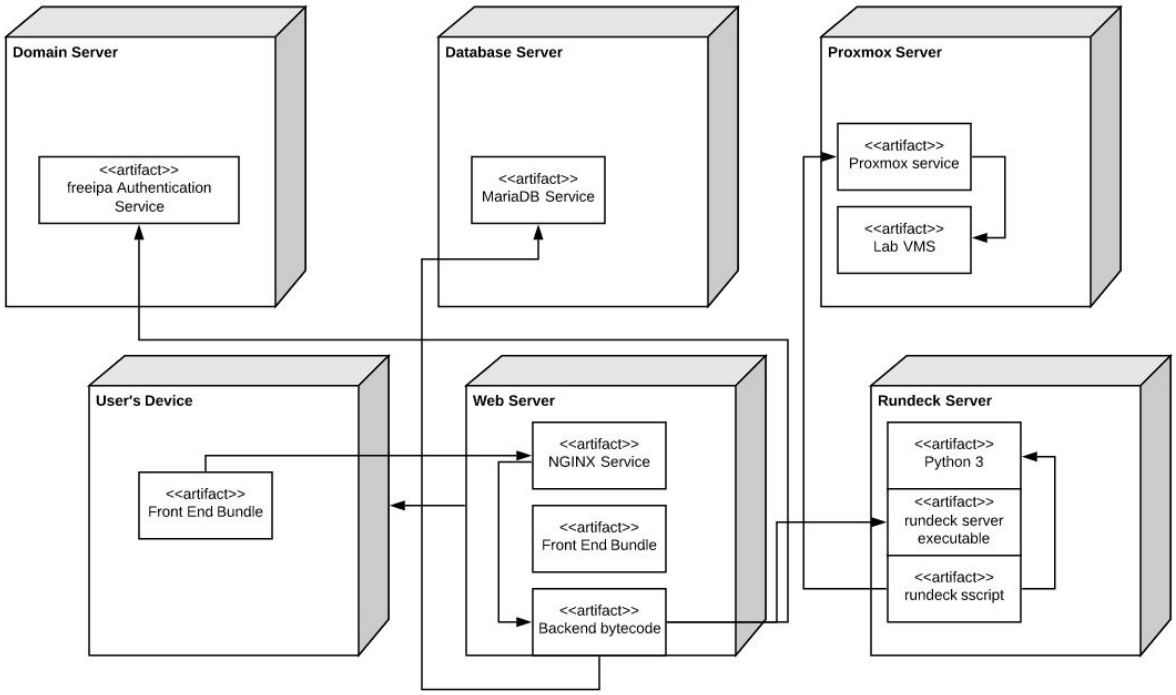
1.1. Overview

The system is divided into the following parts: the frontend, backend api, domain controller, rundeck, database, and the proxmox cluster. First the frontend is going to be built in react with typescript as the language. This will provide maximum type safety with js during the development of this project. The frontend will make api calls to the backend for data and authentication. The backend will be built with ASP .Net Core Web Api in C#. This also provides maximum type safety for validating requests. .Net Core allows us to host the C# application on a linux server. This brings our licensing costs to 0. The backend uses the domain controller for authenticating users. The backend also uses rundeck to issue scripted commands to proxmox. Proxmox will be running, and housing VMs that are controlled by these rundeck scripts.

1.2. Subsystem decomposition

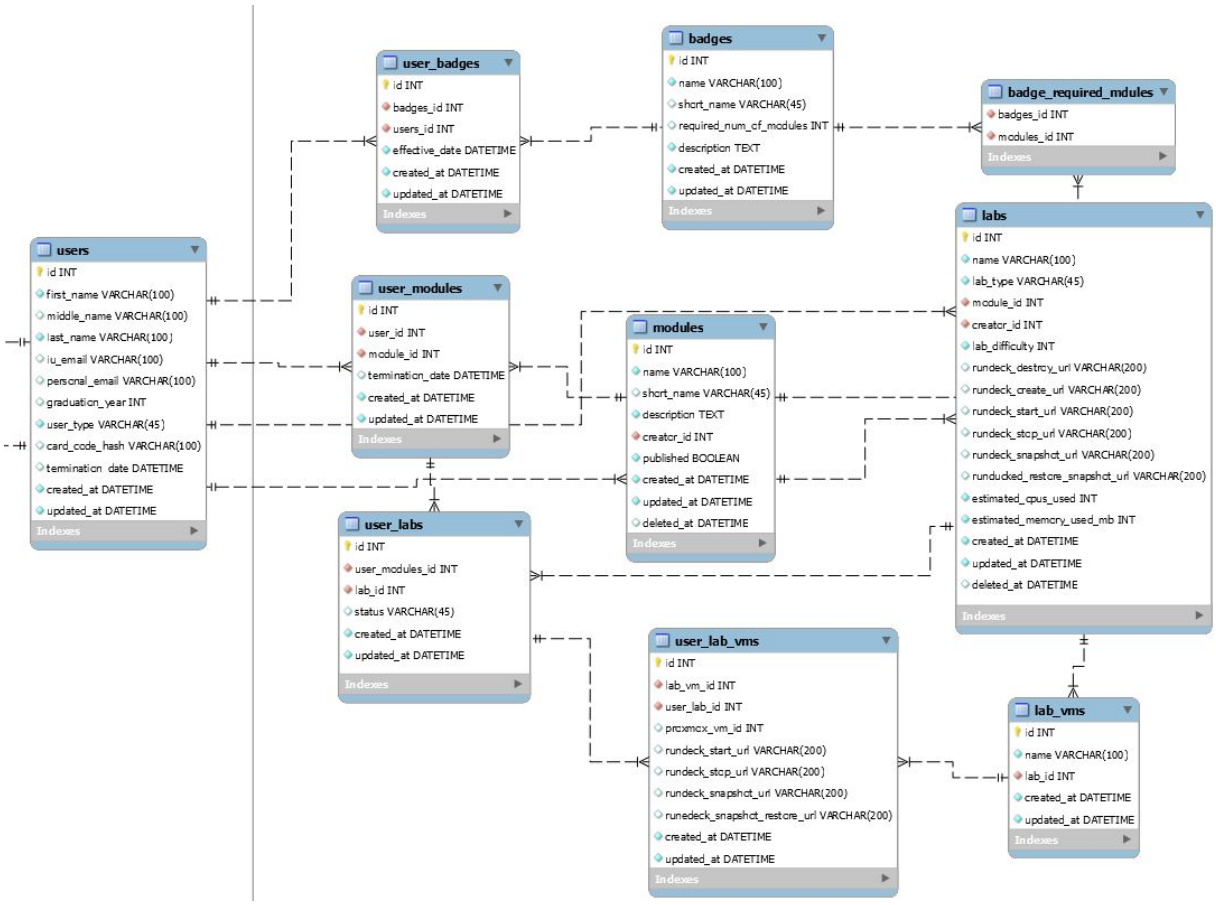


1.3. Hardware/software mapping



1.4. Persistent data management

MariaDB will be used as your persistent database. This database provides a SQL interface to store and retrieve application state. The database will reside on a separate server where the web server can access it with low latency. The application's entities are shown below in the database diagram. 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

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.

Admin

- Can do everything staff and User can do.
- Can create public modules
- Can promote users to staff or admin

Staff

- Can create private modules

- Can edit their private modules
- Can set a start and end date for the private modules they own.
- Can share their module using a share link

User

- Can start a module
- Can change their email address
- Can change their password
- Can start up a vm in their own lab
- Can shut down a vm in their own lab
- Can create a snapshot of a vm in their own lab
- Can restore a snapshot of a vm in their own lab

1.6. Global software control

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 js is single threaded, 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

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 are 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 You will just 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.

Nine Basic Component Types

Use Cases

Teachers need a way for their students to easily have access to a working environment. A student may not have a 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.

Functions

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.

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.

Front end components

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

Triggers

For the 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.

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.

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.

Data Elements

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

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.

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.

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.

Actors/External Entities

The system must interface with namecheap 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.

.