Next Gen System Sustainability

Interim Project Report

Preliminary Requirements, Design, Implementation & Testing

NC Parks and Recreation

CSC 492 Team 4

Cole Goodnight, Kyle Manahan, Matt Murdoch, Josh Roddy,

Colin Szatkowski, and Justin Wald

North Carolina State University

Department of Computer Science

2/25/2022

# Executive Summary

*Author(s): Matt Murdoch*

*Reviewer(s)/Editor(s): Justin Wald*

Our sponsor is the North Carolina Department of Parks and Recreation (NCDPR), which manages state parks and recreational sites across North Carolina. To aid them in managing these resources, the NCDPR maintains a variety of public and private web applications. However, these applications were designed over twenty years ago on an ad-hoc basis and thus are disconnected and time consuming for the DPR IT team to support. The applications are written entirely in PHP5, which hasn’t been supported since 2019. They are served by an Apache webserver, and their database manager is MariaDB 5, which is also out of date. There is no testing environment to work in, and the codebase is extremely difficult to navigate and maintain due to the monolithic structure of the codebase. Our solution to this problem is twofold. Firstly, we created a new, containerized system for their applications that will have separate containers for the current stack, the database and routing. This will enable the creation of a testing environment and the use of version control. The containerized system will also allow them to pull out specific applications into their own containers and update or remake them without having to interfere with the rest of the system. Building off that, the second part of our solution is to begin the process of remaking their existing applications, with our focus being on a training calendar application called DPRCal. The NCDPR uses this application to schedule training courses for their employees, enroll people in those courses, and see an overview of all trainings in the system. Our sponsors chose this application because currently, they are required to manually input training courses, instructors, and enrollees into the calendar and manage all the details. Additionally, the UI is outdated and difficult to navigate, making the application less user friendly. We will re-make this application with a more user-friendly UI, and with extra functionality for instructors to create their own courses and for attendees to enroll themselves, which will relieve some of the burden on our sponsors.

In order to create our solution, we initially split the group into three teams. One team, composed of Justin and Matt, focused on creating the new system and migrating to it. They set up the container for the current applications, and then worked on getting the versions of outdated software up to date, including updating from PHP5 to PHP7, and MariaDB5 to MariaDB10. Another team, composed of Colin and Cole, focused on researching technologies and frameworks to use for the DPRCal remake. They determined that we should use React in the frontend, with Slim PHP in the backend. The third team, composed of Kyle and Josh, focused on generating requirements and low level designs for DPRCal, based on taking the existing calendar functionality of the current application and expanding it to allow instructors and attendees to manage courses and enrollment along with the admins. Once the designs and requirements were generated and our frameworks chosen, we changed our teams so that we could begin coding. Justin focused on the migration effort, finishing the update from PHP5 to PHP7 and creating different dev and prod environments. Matt and Cole handled the frontend of DPRCal, focusing on the creation of the basic admin functionality. Kyle, Colin and Josh developed the backend of DPRCal, focused on creating the REST APIs for getting course, section, user and instructor information. For the final iteration, Kyle and Josh moved to the frontend to assist Matt and Cole with instructor functionality and the creation of the Calendar. Colin focused on developing testing for the backend, while Justin handled authentication for DPRCal.

# Project Description

*Author(s): Matt Murdoch*

*Reviewer(s)/Editor(s): Cole Goodnight, Kyle Manahan*

**Sponsor Background**

North Carolina Department of Parks and Recreation

121 W. Jones St. 1615 Mail Service Center Raleigh, NC 27699-1615

919-707-9300

state.parks@ncparks.gov

Our Sponsor is the North Carolina Department of Parks and Recreation (NCDPR), which operates under the larger NC Department of Cultural and Natural Resources. The NCDPR manages state parks, natural areas, and recreation sites across all of North Carolina. Their goals are to conserve and protect North Carolina’s natural beauty and ecological features, provide safe recreational opportunities, and educate on stewardship of the state’s natural and cultural heritage. As part of this mission, the NCDPR maintains a variety of applications that allows for both the general public and private users to assist the NCDPR employees in their work. For example, the NCDPR maintains a calendar for keeping track of training, educational courses and certifications that their employees may be required to instruct or take.

**Problem Description**

The problem we are addressing is that the web applications the DPR maintains were developed ad-hoc and rely on heavily outdated software such as PHP5, MariaDB5, and Red Hat Enterprise Linux (RHEL) 7. The documentation for the applications is sparse, and they have no version control or testing environment to work in. As such, any development on the applications must be done in production. Another issue is that the current codebase was not developed using contemporary software engineering practices. The application stack is monolithic, with no frontend/backend separation and unnecessary interdependence between applications. Their database is large and disorganized, with duplicate information and extraneous rows. These issues make it very difficult to service the system and further expand functionality for applications. Additionally, some of their outdated software, like PHP5, is no longer supported, rendering their entire application stack unstable.

**Proposed Solution & Project Goals/Benefits**

Our solution to this problem is twofold. First, we will migrate their existing application stack to a new environment, which will allow us to implement version control, provide a testing environment, and update their outdated software versions. We will accomplish this by moving their current application stack to a modern, containerized environment. This will make it easy for us to separate the production and testing environments, making servicing the application easier. Eventually, this system will enable the NCDPR to separate out all of their applications from their combined monolithic stack and into their own containers, which will make scaling the applications easier in the future and enable greater flexibility with regards to technologies for individual applications. The second part of this project is remaking an existing application. Given the difficulty in working off the current codebase, it will be easier to remake new applications entirely rather than simply update the existing ones. Remaking the applications will allow us further expand the functionality and employing contemporary software engineering practices will make servicing them in the future easier. The application we will remake is the training calendar used to schedule required training courses for NCDPR employees, called DPRCal. Our remake will turn the application into a single page application with REST APIs that allow us to separate the frontend from the backend. The new application will use modern frameworks like React and Slim PHP, and will feature a modernized UI that is easier for the user to navigate. The long term goal for this project is to modernize the entire application stack, which requires remaking each individual application. Our project will provide them with the framework to accomplish this, and remaking one application will provide them with a template for remaking the rest of the applications in the future.

# Resources Needed

*Author(s): Cole Goodnight*

*Reviewer(s)/Editor(s): Justin Wald, Matt Murdoch*

| Resource Name | Purpose for Resource | Status | Version | Licensing Information |
| --- | --- | --- | --- | --- |
| DPRCal existing files | The existing software is needed for analysis of legacy structure, requirements for the rewrite, and refactoring to work with modern software stack. | Obtained | n/a | proprietary |
| DB schema/data | The legacy database is needed to determine how data was stored, how we should access it, and what information is relevant to the calendar software. | Obtained | n/a | proprietary |
| RHEL | The legacy operating system(version 7) will be changed to the new version 8 RHEL. This will be the operating system that the server will run on after updating. | Obtained | 8 | UBI EULA |
| Apache | Apache will be used as the runtime environment for the containers. | Obtained | 2.4.52 | Apache License |
| Docker Engine | This software will be used to containerize the system. These containers will be useful for migrating individual parts of the DPR system without affecting others. | Obtained | 20.10.12 | Docker Personal License |
| Docker Compose | Used for defining and running multiple containers that interact with each other, such as the legacy and new containers. | Obtained | 2.2.3 | Docker Personal License |
| Slim | This is built on PHP and will be used to build the backend REST API. | Obtained | 4 | MIT |
| MariaDB | This is used as the database for the entire web app | Obtained | 10 | GPLv2 |
| PHP | Scripting language used for the backend of the web app | Obtained | 7 | Open Source PHP License |
| React | Frontend framework used for the UI of dprCal | Obtained | 17.0.2 | MIT |
| NGINX | Used as a reverse proxy for routing API calls and for designation to the legacy app or the new dprcal | Obtained | 1.20.2 | BSD |
| Jest | Used for testing the UI of the new application | Obtained | 28.0 | MIT |
| PHPUnit | Used for testing the php backend of the new application | Obtained | 9 | PHPUnit open source |
| Postman | Used for testing the API calls during development | Obtained | 9.16.0 | Free postman license |
| MaterialUI | Used for styling the front end | Obtained | 5.6.3 | MIT |
| FullCalendar | Used for the calendar in the new dpr cal | Obtained | 5 | MIT |

# Risks & Risk Mitigation

*Author(s):* [Cole Goodnight](mailto:nbgoodni@ncsu.edu)

*Reviewer(s)/Editor(s): Matt Murdoch, Colin Szatkowski*

* Time
  + This project involves a huge scope with a complex existing system. This requires a lot of time in determining the requirements of the legacy system, updating the software versions without breaking functionality, discovering technologies to use for the new system, rewriting the calendar system, and creating sufficient documentation, tests, and configuration files. To make development quicker without losing quality, we put a lot of time in requirements and design to mitigate reworking the design. Implementation should be done back to front so that during each step, the software is working properly and can be tested fully. If tasks are taking longer than expected, then we reassign team members to get through difficulties.Our subteams and iteration planning also helped with time management as we could complete many tasks at the same time and knew what to work on.
* Communication
  + We will need to keep communication with our sponsors to make sure that the technologies meet the needs of the NC DPR and verify that the new calendar meets their requirements. Without proper communication, the system might not be able to meet their requirements or the software used may not be preferred by the NC DPR. To mitigate this risk, we have weekly meetings and slack communication with our sponsors.
* Configuration
  + Configuration of the new system must be done so that it is replicable and without any issues. It must be done so that the software can be run on the existing DPR servers as well as on individual computers for development purposes. The process of starting the system, updating the system, and replicating the system must be documented so that it is replicable in different environments. Proper documentation and testing will mitigate this risk as the steps for configuration will be clear and the config files will be tested on relevant environments.
* Function
  + The function of the overall system must not be harmed by the upgrading of existing technologies and introduction of new technologies.The rewritten calendar application must encompass the legacy functionality at a minimum and be easily understood by users of the legacy system. To ensure functionality is not lost between the existing system and the new system, we will write tests, record functionality, and run black box tests.

# Development Methodology

*Author(s):* [Cole Goodnight](mailto:nbgoodni@ncsu.edu)

*Reviewer(s)/Editor(s): Matt Murdoch, Colin Szatkowski*

We are currently using feature driven development in order to split the work into small, achievable, working software. This is beneficial in order to show our clients (sponsors) the current progress of the project and give them concrete deliverables with due dates. The iterations of this development process involve defining requirements, building the feature list, creating design documents, implementing the features, and then testing them. These iterations generally are about three weeks long, but are flexible. The end criteria for an iteration is full functionality as set in the requirements and testing complete 100 percent method coverage for backend, 70 percent statement coverage for the frontend.

The requirements are made into a text document listing all the relevant requirements for the project and the requirements for the development iteration are made into a feature based requirements list. This list is created as a team to ensure the scope is within one iteration and the tasks for the iteration are distributed well. Design documents are created such as wireframes and uml in order to make sure the iteration follows a set design approved by the team. Feature implementation is split into sub groups based on front end and back end. Testing is done during implementation and afterwards if needed to increase code coverage.

# System Requirements

*Author(s): Kyle Manahan*

*Reviewer(s)/Editor(s): Matt Murdoch,* [Joshua Roddy](mailto:jbroddy@ncsu.edu)

## **Overall View**

## Within our system, we will be constructing a calendar application that will allow users to search for and enroll in classes, check and gain certifications, and view the calendar of classes and public events. There will also be functionality for instructors and admin users. Instructors will be able to create new classes and courses, along with being able to assign themselves to specific courses and upload their certifications. On the admin level, everything attendees and instructors are able to do will be available, as will the ability to create, edit, and delete classes/courses. All of these tasks will help our sponsors keep track of their own courses but will primarily be used as a template on how their other applications will look when updated to modern frameworks and APIs.

**Glossary:**

* **User:** Any general user of the system, can be an admin, instructor, or attendee.
* **Course:** The classes available to take for the training calendar.

## **Functional Requirements**

* **RQ1 - Course Search\***
  + **RQ1.1:** The DPR Calendar system shall provide all users the ability to search for courses based on year, time period, title, keyword, activity, and district.
  + **RQ1.2:** The year will be able to search from the current year and the next year.
  + **RQ1.3:** The time period will be able to search either the entire year or from January to a specified month.
  + **RQ1.4:** Both title and keyword search will be able to search any word or phrase present in the title or training description respectively.
* **RQ2 - Course List**
  + **RQ2.1:** The DPR calendar will have a result list that displays all classes that are available and show all available sections based on search queries. All levels of users will be able to navigate the class list.
  + **RQ2.2:** The class results will be subdivided by month and will be organized chronologically by date and further organized alphabetically if multiple classes are available on the same date.
* **RQ3 - Course Sign Up**
  + **RQ3.1:** The system shall allow a user to view a course.
  + **RQ3.2:** The system shall allow a user to view the class list of a course.
  + **RQ3.3:** The system shall allow a user to view the instructor of a course.
  + **RQ3.4:** The system will have the current class list be available to view by the admins and instructors.
  + **RQ3.5:** The system will allow attendees to be able to see how many current seats have been taken.
  + **RQ3.6:** The system will show instructor info that contains contact details, any extra comments, and any relevant certifications for the course.
* **RQ4: Course Editing and Creation**
  + **RQ4.1:** The system will let instructors and admins add certification to courses.
  + **RQ4.2:** The system will allow instructors and admins to edit classes.
  + **RQ4.3:** The system will allow instructors and admins to add classes.
  + **RQ4.4:** The system will allow for a search functionality based on an attendees name that is already registered.
  + **RQ4.5:** The system will allow for a search functionality based on any existing class title.
  + **RQ4.6:** The system will allow for search functionality based on any class category.
  + **RQ4.7:** When the user selects a specific class, the system will allow instructors to make any changes to the class title.
  + **RQ4.8:** When the user selects a specific class, the system will allow instructors to make any changes to the completion date.
  + **RQ4.9:** When the user selects a specific class, the system will allow instructors to make any changes to any relevant documentation to show their certification.
  + **RQ4.10:** The system will allow administrators the ability to search for a course based on the course title.
    - **RQ4.10a:** The system will allow administrators to edit the course title.
  + **RQ4.11:** The system will allow administrators the ability to search for a course based on the course id.
    - **RQ4.11a:** The system will allow administrators to edit the course title.
  + **RQ4.12:** The system will allow administrators the ability to search for a course based on the course completion date.
    - **RQ4.12a:** The system will allow administrators to edit the course completion date.
  + **RQ4.13:** The system will allow administrators the ability to search for a course based on the course comments.
    - **RQ4.13a:** The system will allow administrators to edit the course comments.
  + **RQ4.14:** The system will allow administrators the ability to search for a course based on the course hours.
    - **RQ4.14a:** The system will allow administrators to edit the course hours.
  + **RQ4.15:** The system will allow administrators the ability to search for a course based on the course certifications.
    - **RQ4.15a:** The system will allow administrators to edit the course certifications.
* **RQ5 - Instructor/Admin Add/Edit/Remove Course**
  + **RQ5.1:** The DPR calendar will have forms available to instructors and admins to add courses.
  + **RQ5.2:** The DPR calendar will have forms available to instructors and admins to schedule courses.
  + **RQ5.3:** The DPR calendar will have forms available to instructors and admins to edit courses.
  + **RQ5.4:** The DPR calendar will have forms available to instructors and admins to cancel courses.
  + **RQ5.5:** The DPR calendar will have forms available to instructors and admins to delete courses.
  + **RQ5.6:** When a course is submitted, the system will add it to the course list.
* **RQ6: Instructor Creation**
  + **RQ6.1:** The DPR calendar will have an add instructor form that can be accessed by administrators and instructors.
  + **RQ6.2:** The system will allow administrators and instructors to have the option to view a list of instructors.
  + **RQ6.3:** The system will allow administrators and instructors to have the option to add a new instructor.
  + **RQ6.4:** The system will allow any added instructors to be assigned to any existing class.
* **RQ7: Course Certification Functionality\***
  + **RQ7.1:** The DPR calendar will have the ability for admins to view course certifications by attendee.
  + **RQ7.2:** The DPR calendar will have the ability for admins to view course certifications by class.
  + **RQ7.3:** The system will allow admins to search course certifications based on attendee’s last name.
  + **RQ7.4:** The system will have an option to update the certification if necessary.
  + **RQ7.5:** The system will allow admins to be able to view any current certification associated with the attendee.
  + **RQ7.6:** The system will allow admins to be able to edit the current certification date.
* **RQ8: Attendee Lists/Roster**
  + **RQ8.1:** The DPR calendar will allow admins and instructors to view a list of all attendees in a class.
  + **RQ8.2:** The system will have a search form that can use the year of the class and the class title.
  + **RQ8.3:** When the class is found, the system will return a list of all enrollees.
* **RQ9: Evaluation Forms\***
  + **RQ9.1:** The DPR calendar will have the capability for admins to open a class for evaluation.
  + **RQ9.2:** The DPR calendar will have the capability for admins to create an evaluation form for a class.
  + **RQ9.3:** The DPR calendar will have the capability for admins to view evaluations for a class.
* **RQ10: Authentification**
  + **RQ10.1:** The DPR calendar features an authentication service used to login all users into the system.

**Non-Functional Requirements**

* **NFR1:**
  + **NFR1.1:** The DPR calendar system should be optimized for performance through response time by the user for all functionalities

## **Notes for Future Section:**

The requirements marked with an asterisk (\*) are functionalities that we designed the system to include, but were not implemented during our semester of the project. The sponsors may like these to be implemented for the future if this specific application is still worked on.

## **Constraints**

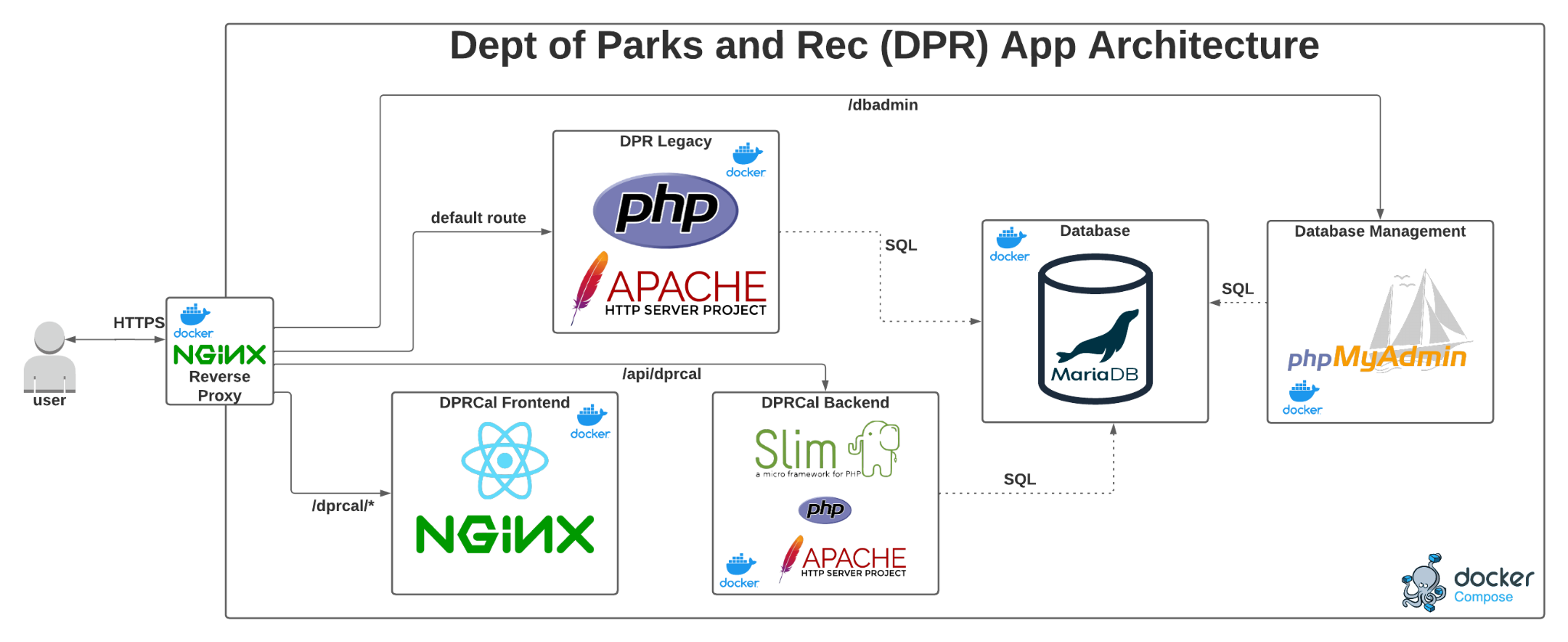
* The system shall be written in PHP7 for the legacy applications.
* The DPR Cal Backend shall be written in PHP using the PHP7 runtime.
* The system must be functional on current versions of MariaDB and Apache..
* The migration process adds the difficulties of containerization and modernizing programming languages.

# Design

*Author(s): Justin Wald, Joshua Roddy*

*Reviewer(s)/Editor(s): Kyle Manahan*

## **High-Level Design**



*Figure 1: DPR Application high level architecture*

## Docker Compose

The new DPR architecture relies on containers orchestrated with Docker Compose, as seen in Figure 1. Containerizing different parts of the DPR system provides separate lifecycle and development cycles to constituent applications. This aids the development process by scoping applications appropriately and facilitating a more agile development process by allowing for application-specific development efforts rather than needing to work on an app monolith. Functions for the stack as a whole are separate such as the MariaDB container, NGINX reverse proxy, and the PHPMyAdmin container. Furthermore, the necessitation of explicit container port exposing improves networking visibility and security.

## Reverse Proxy (Application Entrypoint)

At the entrypoint of the DPR application stack is an Nginx reverse proxy. This web server accepts HTTPS traffic, and redirects HTTP traffic if needed, before redirecting that traffic to the appropriate container within the compose stack. This functionality will be used to serve the legacy DPR stack, the new DPRCal frontend, and the new DPRCal backend on the same host with their own web servers. In the future, the proxy will allow new services, which could use wildly different technologies, to be redirected with a single configuration change.

## DPRCal Frontend

The frontend of the rewritten DPRCal application is a single page React app that runs in its own container. Information from the backend is retrieved, as needed, from the backend container via the exposes REST api. This minimizes bandwidth and response as only information is transmitted for these calls. This differs from the legacy app which returned entire pages on actions such as form submits.

## DPRCal Backend (REST API)

The application logic of the new DPRCal application will be provided by an entirely new REST API built on the SLIM framework and written in PHP. This API will serve HTTP requests from the frontend that affect the data of the application such as the creation of a new course, the addition of a student, etc. These requests will be processed as needed within the backend and changes made directly to the MariaDB database.

## MariaDB Database

MariaDB has been separated into its own container, again to improve encapsulation and provide individual life and development cycle control. As part of the network configuration within the compose stack, applications that require database access will have direct access to the database container under a dns name. Database administration is handled by phpMyAdmin for system administrators to view raw database information or make changes as needed.

Authorization

Authentication and authorization are handled via a JSON Web Token Scheme. The user logs in using credentials that are checked against a salted hash in the DB. If these match, the user’s browser is provided with a signed JWT token that is stored in the browser’s cookies. This JWT includes the user’s username and role and is required to authenticate on any endpoints. When a backend endpoint is queried, a middleware layer verifies that the JWT is unmodified by checking the signature then parses the user’s role to ensure they are authenticated for that endpoint.

## **Low-Level Design**

## Database Design



*Figure 2: dprcal\_new Database Design*

Figure 2 shows the database design for the remade training calendar application. When investigating the orginal *dpr\_cal* databases, we found that it consisted of many disorganized databases and tables, so rather than attempting to normalize it and risk causing errors in the legacy stack, we created this much simpler database for DPR\_Cal. The core component is the creation of *Courses*, which can have an unlimited number of *Sections*. Sections relate back to their Course, but also have an *Instructor* and a *Roster* students. The “instructors” table is linked to *users* by its user\_id which will have to, but is composed of its own object with demographic information. A *Roster* has zero or more *Users*, which are students enrolled in the Section of the Course. This simplified database is well-organized and will be easy to expand off of as the next-generation DPR system continues to be developed.

## UML Class Diagram

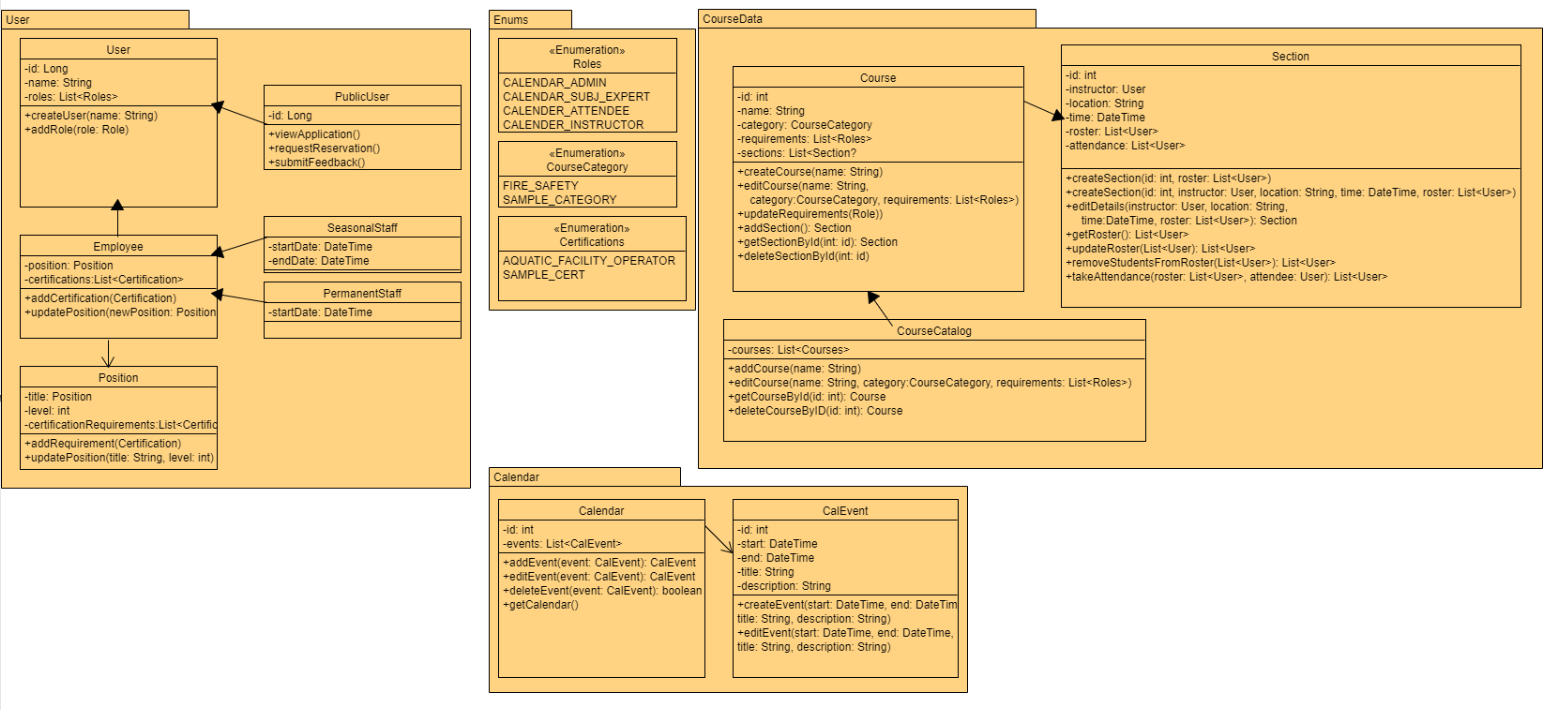
*Figure 2: Model Package Class Diagram*

Figure 2 shows the basic, class-level design of the Model package for the system. Specifically for DPRCal, the Calendar, Course, and Section objects are enough, but in scope of the entire application, the User Package will handle the backend of user sign-ons, registration, accommodation, and certification. This package will be expanded in further Iterations, but for Iteration 1, the important functionality is the ability to create a User and assign a Role, specifically the Admin. From an Admin log-on, a User will be able to access all API functions described in the API Endpoints section, which will send HTTP requests to the model classes. The Course object will maintain the requirements to take a course (things like certifications or prerequisites, which are handled in our system by adding roles to Users). Admin users in the calendar system will have access to the creation and modification of Courses from the CourseCatalog, from which they will also be able to create Sections for each Course. Sections can be created without an Instructor (the other primary User Role in the DPRCal application) time, or location, as all details are not always finalized when sections for classes need to be created, but in its final form, a Section will have an Instructor, time, and location.

The Calendar will pull Courses with their Sections from the CourseCatalog, using them to create Calendar Events, which will then display details for all Courses and Sections across the Calendar. Functionality that will be revisited after Iteration 1 will include the ability to filter Courses/Sections by times, locations, instructors, and requirements. Additionally, design for the API connection between frontend and backend will need to be further developed by the end of Iteration 1. The current design though, displays the first, necessary objects to be developed so that the goals set for Iteration 1 - the ability of an Admin user to log in, add Courses and Sections, and display them to the calendar.

## REST API Endpoints

As described in the DPRCal Backend section, the REST API serves HTTP requests from the frontend that connect with the backend operations and the database. The functionally of the endpoints is centered on the ability of Users to view, create, modify, and remove Courses and Sections that will populate the Calendar of DRPCal. Each request is specifically linked to an database objects that are shown in the Model class diagram.

The first three groups of endpoints are listed below by their operation and path. An example is provided as the first description. The full documentation of the API as an html file can be found at *documentation\DPR\_CAL\_API\_HTML* in the project, or hosted by SwaggerHub [here](https://app.swaggerhub.com/apis-docs/joshbroddy/dpr-cal_api/1.0.0). Each endpoint either takes in or returns a JSON object, each of which are described in the full documentation.

**Example Group:**

**/path: GET, POST**

**/{path\_variable}: GET, PUT, DELETE**

**/path: GET, POST**

**/path/{path\_variable}: GET**

**Authorization:**

**/auth**

**/token: POST**

**Users:**

**/users: GET, POST**

**/{id}: GET, PUT, DELETE**

**/{me}: GET**

**/role/{role}: GET**

**/user/{username}: GET**

**Courses:**

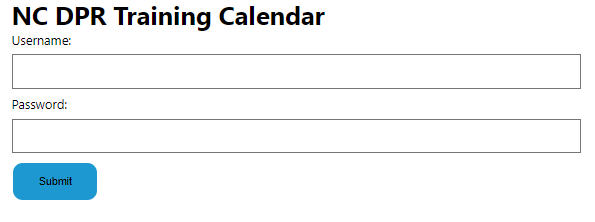
**/course: GET, POST**

**/{id}: GET, PUT, DELETE**

## **GUI Design**

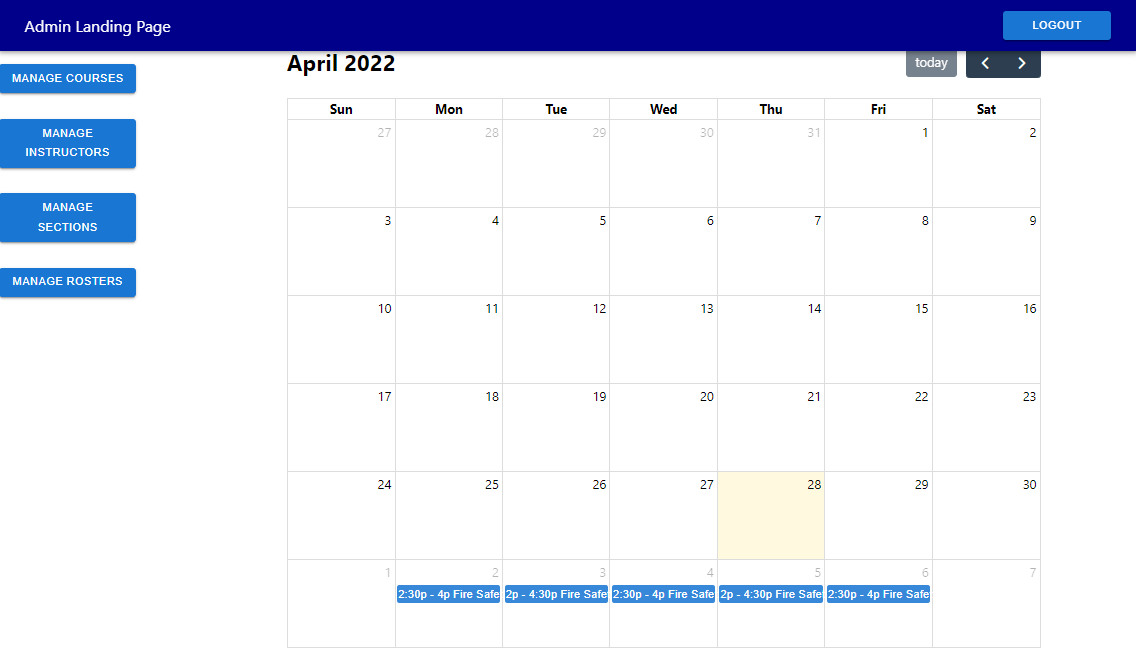
*Author(s): Colin Szatkowski,* [Cole Goodnight](mailto:nbgoodni@ncsu.edu)

*Reviewer(s)/Editor(s):* [Joshua Roddy](mailto:jbroddy@ncsu.edu)

**

*Figure 3 - Login Page*

This page shows the login for any user. Depending on the credentials put into the form and submitted, the user will have different home pages and different levels of access to the dpr calendar application.

**

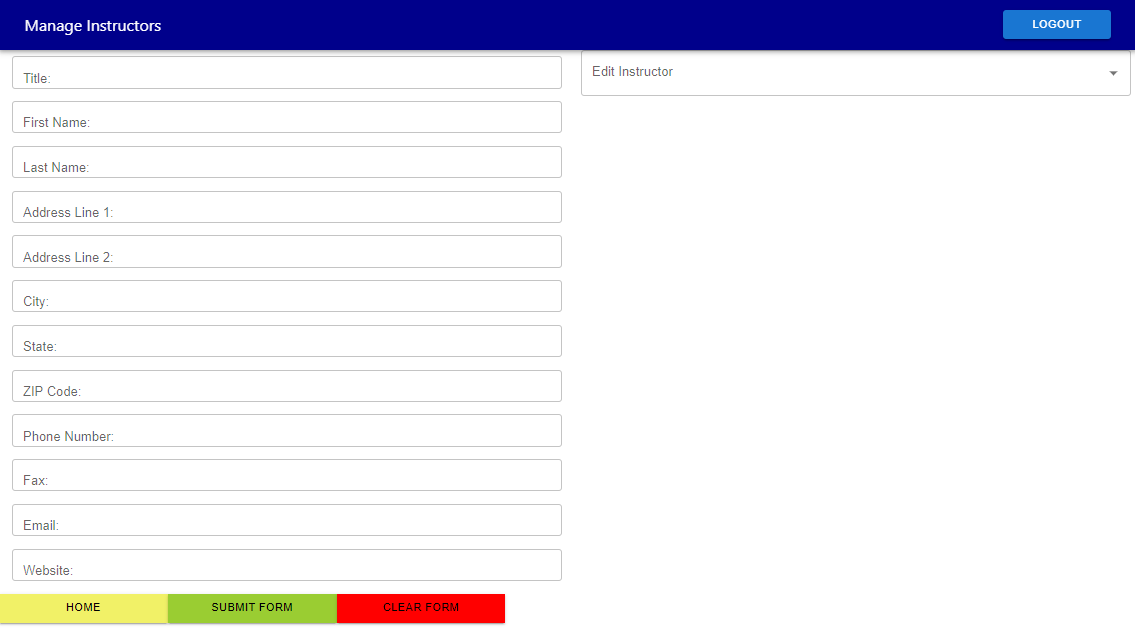
*Figure 4 - Admin Home Page*

The admin home page has access to all functionalities of the calendar. They can select manage courses to add/edit courses, manage instructors to add/edit instructors, manage sections to add/edit section listings of courses, or manage rosters to add/remove students from a roster. They can also view the calendar as seen on the right side of the screen. This calendar shows all the sections listed. They can select the arrows on the top right of the calendar to move forward or back month by month. The logout button will sign out the user and prevent further use of the application.

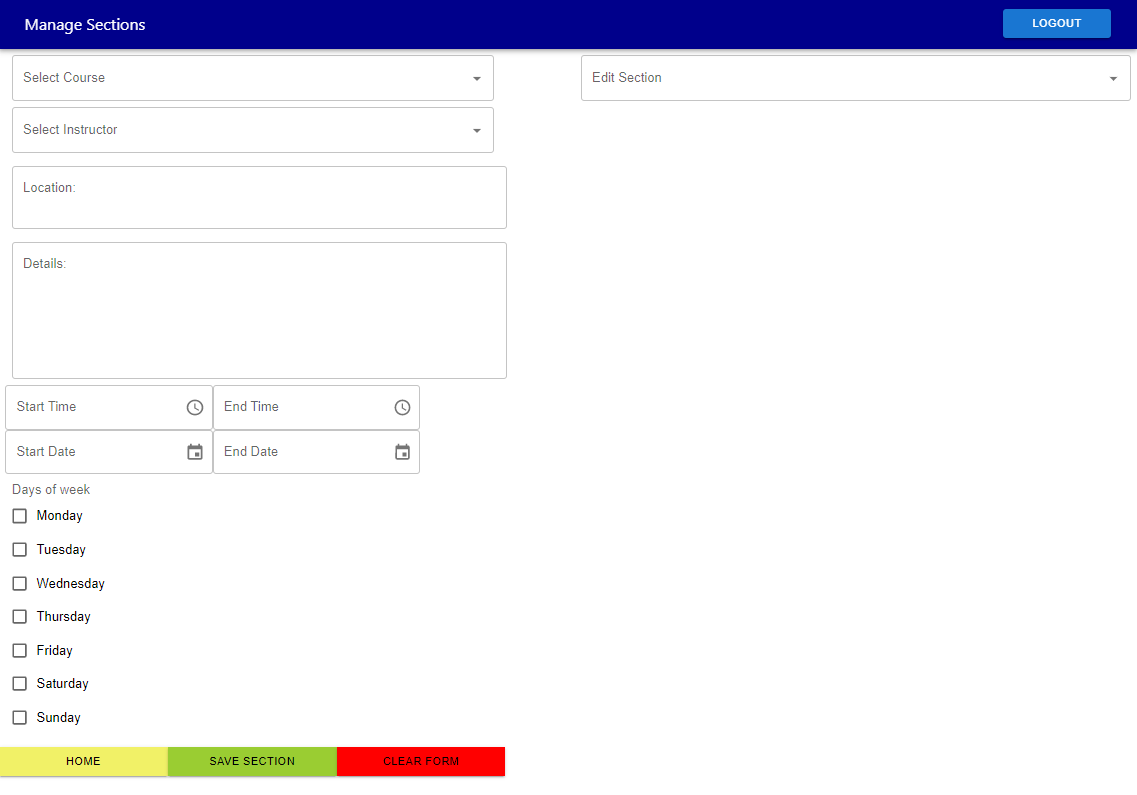
**

*Figure 5 - Manage Courses Page*

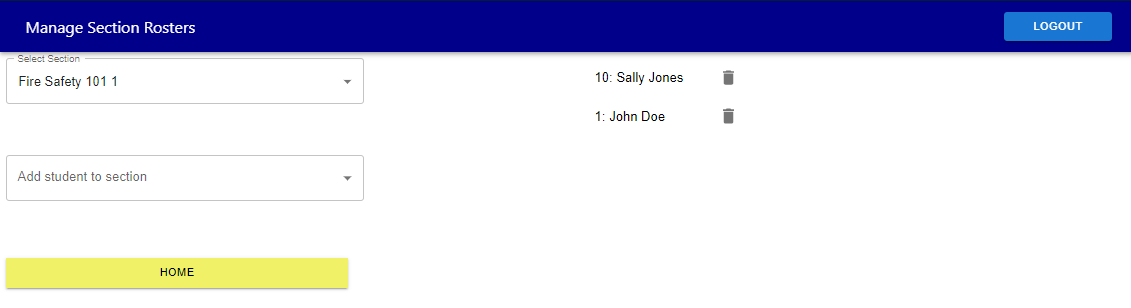
The manage courses page provides editing and adding courses. These courses have a name, category, description, and requirements. The edit course dropdown on the top left will allow a user to select the course they want to edit. This will then input the course in the form for editing. The save course button will add the course with the provided options or edit the course if one was selected. The clear form button empties the form and the home button returns to the home page.

*Figure 6 - Manage Instructors Page*

The manage instructors page allows for adding and editing instructors. All fields must be filled in order to submit the form. If an option is selected from the drop down menu in the top right, then the selected instructor will be edited upon submitting the form. The clear form button empties the form and the home button returns to the home page.

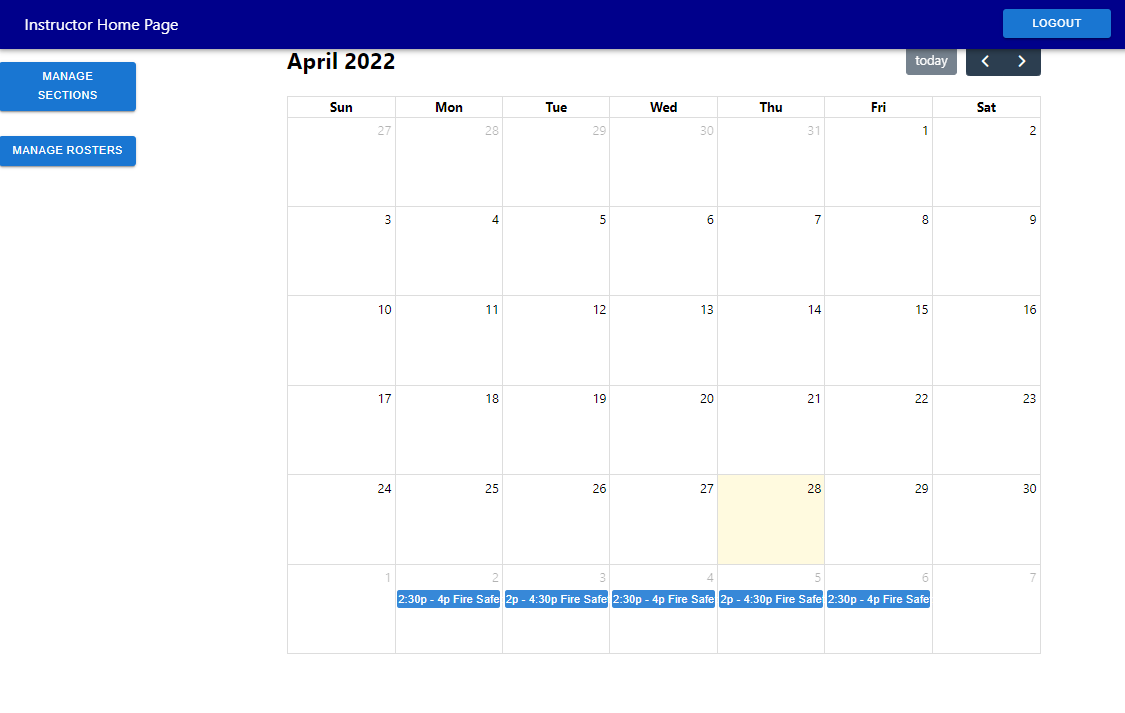
*Figure 7 - Manage Sections Page*

The manage sections page allows for adding and editing sections. This page includes fields of course and instructor which link to the instructor or course chosen. The time and date options are chosen using a clock pop-up and a date selector or text input. The days of the week options can be chosen for what days the class will meet, repeating from start date to end date. If edit section is selected, the save section button will edit the selected section, otherwise the save section button will add a new section. The clear form button empties the form and the home button returns to the home page.

**

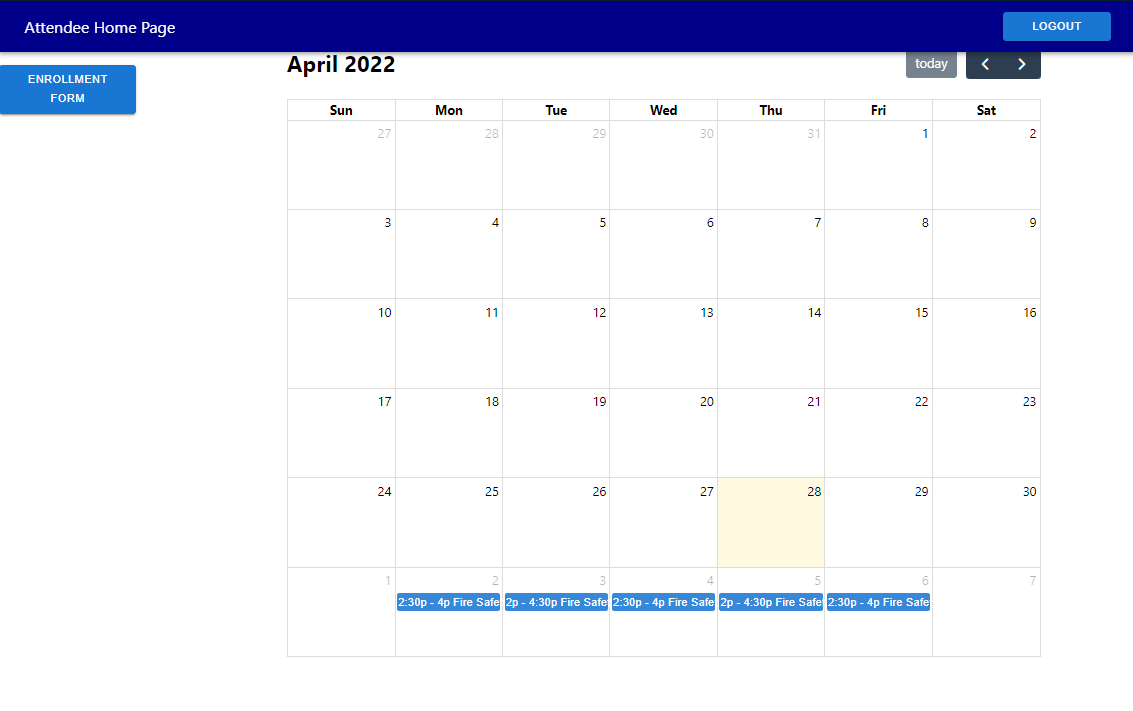
*Figure 8 - Manage Section Roster Page*

The manage roster form allows for a user to add or remove students from a section. The section is chosen from the dropdown and then a student is chosen from the next dropdown. Upon clicking the student, they will get added to the roster list. Select the trash can icon and the user will have the option to cancel operation or remove student from the roster. The home button returns to the home page.



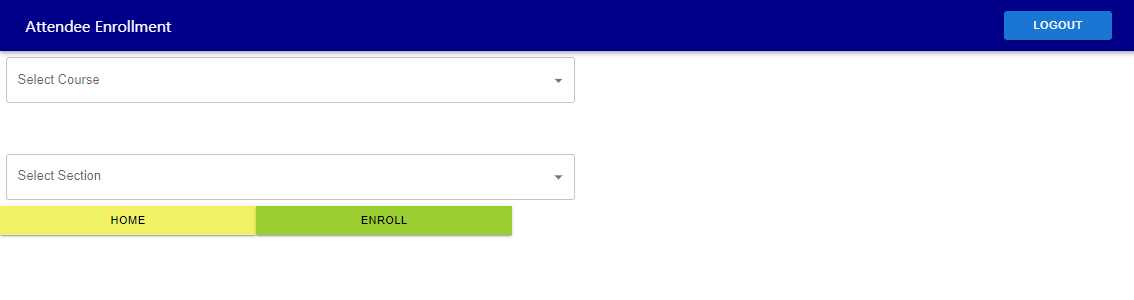
*Figure 9 - Manage Section Roster Page*

The instructor home page is similar to the admin home page, but has less permissions. They can manage sections and rosters, but cannot manage courses or instructors. The manage sections and roster pages are the same for the instructor and admin except the instructor may only edit the roster of sections they are assigned to.



*Figure 10 - Manage Section Roster Page*

The attendee home page only has the calendar view and the enrollment form. They can select this form to enroll in a section of a course.



*Figure 11 - Manage Section Roster Page*

The attendee enrollment form allows a student to choose any course and section and enroll themselves in this section using the enroll button.

# Implementation

*Author(s): Justin Wald, Matt Murdoch*

*Reviewer(s)/Editor(s): Cole Goodnight, Colin Szatkowski*

## **Iteration Definition & Current Status**

* Iteration 1 (Current Iteration)
  + Start date: 2-16-22
  + End date: 3-1-22
  + Features
    - The focus of this iteration is getting basic admin functionality up and running
      * Admin can create courses
        + RQ5.1, 5.2, 5.6
      * Admin can create instructors
        + RQ6
    - On the migration side, there are three goals
      * Get the app running with dependencies met - Completed
      * Get all the correct configurations - In progress
      * Code changes to bring it up to date with current software version - In progress
* Iteration 2
  + Start date: 3-2-22
  + End date: 3-27-22
  + Features
    - The focus of this iteration on the frontend was on finishing out admin functionality by allowing them to edit courses and instructors
      * Admin can edit an existing course
        + RQ5.3
      * Admin can edit an existing instructor
        + RQ6
    - On the backend, the focus was on finishing out the endpoints and database tables for instructors and sections
      * Endpoints and table for getting and editing an instructor
        + RQ6
      * Endpoints and table for getting, creating and modifying sections
        + RQ4
    - Migration
      * Finish updating code version, particularly finishing the update from PHP5 to PHP7
        + Document specific changes that were required for updating future applications
      * Create two different environments for development and production
* Iteration 3
  + Start date: 3-28-22
  + End date: 4-25-22
  + Features
    - The focus of this iteration was completing all remaining functionality for the DPRCal application
      * Authentication/Security - RQ10
        + Verifying username and password when logging in
        + Assigning a role when a user logs in, and sending them to the appropriate landing page
        + Securing backend endpoints so that only authenticated and authorized users may access them
      * Attendee functionality - RQ3
        + Enrollment form for attendees to view courses and enroll in them
      * Instructor functionality - RQ8
        + Form for instructors to create new sections
        + Form for instructors to view students in their sections and add/remove them
      * Calendar
        + Calendar that displays all scheduled sections in the system, that is shown on each landing page
    - Migration - RQ2
      * Create documentation to ensure that sponsors can install and maintain the system

All of the features outlined in these iterations were completed.

## **Security Considerations**

* Authentication
  + Our authentication scheme is a straightforward login page. When a user accesses the application, they must login. This login information is sent the backend’s /auth/token endpoint via HTTPS (to protect the plaintext credentials). The token endpoint accepts credentials and queries the database for the password hash of the user with the matching username, if one exists. This plaintext password provided by the user is verified using PHP’s password\_verify method against the hash for the retrieved user. This method automatically hashes the plaintext password the correct number of iterations and salts the hash based on the information stored in the database hash. If the hashes match, a JSON Web Token (JWT) is generated for the user with the user’s username and role as claims. This JWT is split into its header, payload, and signature and stored in different cookies in the client’s browser via the Set-Cookie response header for security. These JWT cookies will be provided in future API queries to validate requests.
* Authorization
  + Once the user’s browser receives the user’s JWT cookies, the user is ready to submit API queries. When the user submits a query, the JWT cookies are included in the request header. Requests to protected endpoints are intercepted by middleware on the backend that first recombines the sections of the JWT before verifying the signature of the JWT. This ensures the JWT hasn’t been tampered with and can be parsed for the user’s role. The parsed role is then compared against the permitted users for the given route and the request is allowed or disallowed.

## **Project Folder Structure**

/:

top\_level\_files:

# Files in the root directory

- .gitignore

- .docker-compose

- README.md

db\_persistence:

# This folder is mounted to the MariaDB container to

# provide a database persistence across deployments.

db\_schema:

# Database schema descriptor files will be stored here

# in order to populate the database with empty schema

# objects should the need arise.

envs:

# Environment files that store environment variables

# used to configure the compose build.

secrets:

# Empty directory where secret files will be placed at

# run time to be passed to containers.

documentation:

# DPR-stack level documentation (such as this file)

dprcal:

# Assets specific to the dprcal application live here.

# The file structure within an application's folder is

# more variable due to the nature of the application,

# but this is a general example.

backend:

# Many applications in the DPR stack will have seperate

# frontend and backend containers.

documentation:

# Documentation files.

middleware:

# Any middleware layers (such as for authentication / authorization are contained in this directory.

public:

# This directory is the entrpoint of the Slim application and contains the index.html file.

routes:

# Route implementations are implemented here in files that resemble the groups declared in index.html.

test:

# Test files.

lib:

# Library files.

config:

# Configuration files including the Dockerfile.

secrets:

# Secret files.

envs:

# Environment variable files.

frontend:

# Frontend for the application.

documentation:

# Documentation files.

test:

# Test files.

config:

# Configuration files including the Dockerfile.

src:

# Source files.

AdminOverview:

# Controls the landing page for any admins.

AppHeader:

# Controls the header.

AttendeeOverview:

# Controls the landing page for any attendees.

Calendar:

# Controls the Calender GUI.

CreateOrEditCourse:

# Used for the create/edit course pages.

CreateOrEditInstructor:

# Used for the create/edit instructor pages.

CreateOrEditSection:

# Used for the create/edit section pages.

EnrollmentForm:

# Used for attendees enrolling in course sections.

InstructorOverview:

# Controls the landing page for any instructors.

ManageRoster:

# Used for adding/removing attendees from course sections.

legacy\_application:

# Assets specific to the legacy application stack live here.

# The file structure within an application's folder is

# more variable due to the nature of the application,

# but this is a general example.

documentation:

# Documentation files.

src:

This structure is the same as in the current application. This was kept as such intentionally to provide as similar a project structure to the current application as possible. This includes the complete directory structure.

config:

# Configuration files including the Dockerfile and Apache conf files.

secrets:

# Secret files.

envs:

# Environment variable files.

## **Project Configuration/Settings**

## docker-compose.yaml

The docker compose file describes the stack configuration of the project. Within the compose file are individual services that are either built from a single Dockerfile or run from a pre-built image. These services get configured with their own port forwarding rules, volumes, environment variables and more. Via these mechanisms, the containers may support further configuration on a case-by-case basis. For example, the MariaDB container has its default root and user passwords populated by docker through mounting files containing the passwords in a secure directory and whose absolute paths are specified in environment variables at runtime.

## Dockerfiles

Dockerfiles provide build instructions on a service level. They specify a base image that has layers built on top of it to configure it appropriately. This includes running commands to install packages, copying source code files, setting environment variables and more. These Dockerfiles are the basis for a containerized structure and allow images to be built efficiently and quickly.

## .env files

Within the docker-compose, there are several instances of configuration variables that can be modified by the environment file of the compose environment. This allows us to modify variables of the compose stack at runtime to specify things like php version, mariadb version, etc. These environment variables are specified in separate files to facilitate significantly different runtime configurations. These are the available values and typical values:

### General ###

# Hostname the stack is being served at.

STACK\_HOSTNAME=auth.dpr.ncparks.gov

# Specify the deploy type to use. Any service that supports a dev vs prod env will use this variable to configure itself.

DEPLOY\_TYPE=dev

### Legacy ###

# URI path to servce the service at.

LEGACY\_PATH=/

# PHP version to build the image on.

LEGACY\_PHP\_VERSION=7.4

### MariaDB ###

# Version of MariaDB to use.

MARIADB\_VERSION=10.6

# Local directory to place database persistence files at.

MYSQL\_DATA\_DIR=./db\_persistence

# Root password file location to read password from.

MYSQL\_ROOT\_PASSWORD\_FILE=secrets/mysql\_root

### Nginx ###

# SSL certificate location for https.

SSL\_CRT\_FILE=secrets/domain.crt

# SSL encryption key file location for https.

SSL\_KEY\_FILE=secrets/domain.key

### DPRCalfrontend ###

# URI path for the service.

DPRCALFRONTEND\_PATH=/new/dprcal

### Dprcalbackend ###

# URI path for the service.

DPRCALBACKEND\_PATH=/api/dprcal

# PHP version for the backend.

DPRCAL\_BACKEND\_PHP\_VERSION=8.0

# Static file serving location for apache.

DPRCAL\_BACKEND\_APACHE\_DOCUMENT\_ROOT=/dprcal/backend/public/

# Path to the json file containing key info for the jwt key used to sign backend tokens

JWT\_KEY\_PATH=secrets/jwt\_key.json

### PHPMyAdmin ###

# URI path for the service.

PHPMYADMIN\_PATH=/dbadmin/

## nginx.conf

The nginx config file will configure the nginx reverse proxy that serves as the entrypoint to the DPR stack.

# Testing

*Author: Colin Szatkowski*

*Reviewer(s)/Editor(s): Kyle Manahan*

## **Overall View**

Our project, in accordance with standard testing practices, have leveraged both unit tests and automated black box tests. Our unit testing was performed with PHPunit and thoroughly tested our backend both locally and over the network by querying our API endpoints to ensure end-to-end reliability. During development, we also used Postman in order to manually test the API endpoints before having our PHPUnit automated tests functioning. To test our frontend, we opted to use Jest, a javascript testing framework, to streamline our automated blackbox testing by facilitating automated frontend interaction.

For testing, we made a special environment file (named dev.env) that can configure Docker compose to put the system in a testable state. This included specifying a separate database container than production and specified a different host name for testing purposes. Furthermore, given the nature of the compose stack, recreating an environment in a repeatable manner is made trivial by each constituent application being an immutable image with persistent data explicitly created by use of volumes.

| Testing Environment Dataset | |
| --- | --- |
| admin | admin role user |
| instructorJohn | instructor role user  **Subject:** Safety  **Title: <**Blank>  **First Name:** John  **Last Name:** Doe  **Address 1:** 123 Water Ln  **Address 2:** <Blank>  **City:** Raleigh  **State:** NC  **Zip:** 27603  **Phone:** 123-456-7890  **Fax:** <Blank>  **Email:** john@ncparks.gov  **Website:** <Blank>  **Instructor’s Subject(s):** Fire Safety 101 |
| instructorJane | instructor role user  **Subject:** Medical  **Title: <**Blank>  **First Name:** Jane  **Last Name:** Doe  **Address 1:** 321 Fire Ln  **Address 2:** <Blank>  **City:** Raleigh  **State:** NC  **Zip:** 27603  **Phone:** 111-222-3333  **Fax:** <Blank>  **Email:** jane@ncparks.gov  **Website:** <Blank>  **Instructor’s Subject(s):** <Blank> |
| courseFireSafety | **Course Title:** Fire Safety 101  **Min Class Size:** 8  **Max Class Size:** 30  **District:** Raleigh  **Park:** Dorothea Dix  **Location:** 1030 Richardson Dr, Raleigh, NC 27603  **Contact Info:** Email joe@ncparks.gov for any questions  **Comments:** <Blank> |
| courseFishing | **Course Title:** Ethical Fishing 101  **Min Class Size:** 5  **Max Class Size:** 15  **District:** Lake Norman  **Park:** Lake Norman State Park  **Location:** 759 State Park Road, Troutman, NC, 28166  **Contact Info:** Email lakenorman@ncparks.gov for any questions  **Comments:** Bring your own fishing rod or bring $10 to rent one! |
| courseFirstAid | **Course Title:** First Aid 101  **Min Class Size:** 20  **Max Class Size:** 50  **District:** <Blank>  **Park:** <Blank>  **Location:** 408 Ashe Ave, Raleigh, NC 27606  **Contact Info:** Email pullencenter@ncparks.gov for any questions  **Comments:** <Blank> |

## **Unit Testing**

For the backend, we used PHPUnit for our whitebox testing. We were able to achieve 100% method, ~70% statement, and ~70% branch coverage for all of our PHP backend using PHPUnit. Our frontend unit testing was done with the React testing library and was run using Jest. We used these technologies to mock our frontend and determine whether the appropriate components were present on the page. We also created tests to determine whether or not the page behaved as expected when certain user events, like clicks on buttons, were triggered. We were able to hit 70% statement coverage for CreateOrEditCourse, AdminInstructorPage, AdminLandingPage, App and LoginPage. Additionally, all the tests that we wrote were passing.

**Excluded From Unit Tests**

GUIs: Our GUIs did not have unit tests, and were instead tested manually and through using Jest. Usability testing was also done manually.

## **Acceptance Testing**

For the frontend, we used Jest to interface with React and automate testing. We had 70% statement coverage for our frontend.

Preconditions for all tests: The system is running with no previous data in the test dataset

| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| 1 | Precondition: User is on the NCDPR home page.  1. User clicks the link to the dprcal training application | The login page for dprcal appears. | The login page for dprcal appears. |
| 2 | Precondition: User is on the dprcal login page.  1. The user enters “admin” in the username field  2. The user enters “password” in the password field  3. The user clicks the “login” button | The admin home page for dprcal appears. | The admin home page for dprcal appears. |
| 3 | Precondition: User is on the dprcal login page.  1. The user enters “admin” in the username field  2. The user enters “wrongpassword” into password field  3. The user clicks the “login” button | An error message appears stating that the username or password entered was incorrect. | An error message appears stating that the username or password entered was incorrect. |
| 4 | Precondition: User is on the dprcal admin home page and is logged in with the admin role.  1. The user clicks the “Add New Course” button | The Add New Course page appears | The Add New Course page appears |
| 5 | Precondition: User is on the Add New Course page and there is currently no courses added to the system  1. The user enters the following information:  **Course Title:** Fire Safety 101  **Min Class Size:** 8  **Max Class Size:** 30  **District:** Raleigh  **Park:** Dorothea Dix  **Location:** 1030 Richardson Dr, Raleigh, NC 27603  **Contact Info:** Email joe@ncparks.gov for any questions  **Comments:** <Blank>  2. The user clicks the “Add Course” button | A “Class created successfully” message appears and the course is added to the database | A “Class created successfully” message appears and the course is added to the database |
| 6 | Precondition: User is on the Add New Course page  1. The user enters the following information:  **Course Title:** <Blank>  **Min Class Size:** 8  **Max Class Size:** 30  **District:** Raleigh  **Park:** Dorothea Dix  **Location:** 1030 Richardson Dr, Raleigh, NC 27603  **Contact Info:** Email joe@ncparks.gov for any questions  **Comments:** <Blank>  2. The user clicks the “Add Course” button | An error message is displayed informing the user that a course must have a course title | An error message is displayed informing the user that a course must have a course title |
| 7 | Precondition: User is on the admin home page  1. The user clicks the “Add Instructor” button | The Add Instructor page appears | The Add Instructor page appears |
| 8 | Precondition: User is on the Add Instructor page and the “Fire Safety 101” course has been added  1. The user enters the following information:  **Subject:** Safety  **Title: <**Blank>  **First Name:** John  **Last Name:** Doe  **Address 1:** 123 Water Ln  **Address 2:** <Blank>  **City:** Raleigh  **State:** NC  **Zip:** 27603  **Phone:** 123-456-7890  **Fax:** <Blank>  **Email:** john@ncparks.gov  **Website:** <Blank>  **Instructor’s Subject(s):** Fire Safety 101  2. The user clicks the “Add Course to Instructor” button  3. The user selects “Fire Safety 101” as the course to be added to the instructor  4. The user clicks the “Add Instructor” button | A drop down box populated with courses in the system appears  A “Instructor added successfully” message appears and the Instructor is added to the database | A drop down box populated with courses in the system appears  A “Instructor added successfully” message appears and the Instructor is added to the database |
| 9 | Precondition: User is on the Add Instructor page  1. The user enters the following information:  **Subject:** <Blank>  **Title: <**Blank>  **First Name:** John  **Last Name:** Doe  **Address 1:** 123 Water Ln  **Address 2:** <Blank>  **City:** Raleigh  **State:** <Blank>  **Zip:** 27603  **Phone:** 123-456-7890  **Fax:** <Blank>  **Email:** john@ncparks.gov  **Website:** <Blank>  2. The user clicks the “Add Instructor” button | An error message is displayed informing the user that the subject field and the state field cannot be left blank. | An error message is displayed informing the user that the subject field and the state field cannot be left blank. |
| 10 | Preconditions:   * User is on the admin home page * Instructor “John Doe” exists in the system with no courses assigned to them * Course “Fire Safety 101” has been created with no instructors assigned to it   1. The user clicks the “Edit Instructor” button.  2. The user selects “John Doe” from the list of instructors  3. The user clicks the “Add course to instructor” button  4. The user selects “Fire Safety 101” from the list of courses.  5. The user clicks the “Submit” button  6. The user clicks the “Home” button | The Edit Instructor page appears  A drop down box populated with courses in the system appears  A “Fire Safety 101 has successfully been added to John Doe” message appears | The Edit Instructor page appears  A drop down box populated with courses in the system appears  A “Fire Safety 101 has successfully been added to John Doe” message appears |

# Suggestions for Future Teams

*Author(s): Joshua Roddy*

*Reviewer(s)/Editor(s): Colin Szatkowski*

## **Additional Calendar Functionality**

The requirements marked with an asterisk (\*) are functionalities that the system was designed to include, but were not implemented during this first semester of the project. The sponsors may like these to be implemented for the future if this specific application is still worked on.

## **Future Remakes**

There are many applications that are maintained by the NCDPR, and remaking one or several of these applications could be a suitable task for a future project. One that could potentially be remade is an application that NCDPR uses for searching documents, or some of the applications they use for managing finances.

## **Developing Next Project**

The containerization done in this project allows for additional apps to be developed separately from the rest of the system. This means that the next team to take this project will have the freedom to choose whatever technologies they and the sponsors agree on for re-making new applications, or whatever other task they may decide to do.

The legacy database is large and unwieldy, with many tables containing columns of information that may be irrelevant to the function the table serves. The legacy database must remain in use for the remaining applications; however, so rather than trying to normalize it, you may find it easier to take the fields of their tables, and link them with the *dprCal\_new* database that we created for the calendar application. There may be some names that need to be changed, but things like the *Users* table could be reused for other applications.