

Red Giant Diagnostics

Senior project

Andrew Horn | Oregon Institute of Technology | 2019 - 2020

Contents

[Table of Figures ii](#_Toc42581431)

[List of Acronyms iii](#_Toc42581432)

[Project Approval iv](#_Toc42581433)

[Approvals iv](#_Toc42581434)

[Abstract 1](#_Toc42581435)

[Outline 2](#_Toc42581436)

[Chapter 1: Introduction 3](#_Toc42581437)

[Overview 3](#_Toc42581438)

[Product Description 3](#_Toc42581439)

[Existing Products 3](#_Toc42581440)

[Summary 3](#_Toc42581441)

[Chapter 2: Background 4](#_Toc42581442)

[Overview 4](#_Toc42581443)

[Reprise License Manager 4](#_Toc42581444)

[Licensing 4](#_Toc42581445)

[Checkout Process 5](#_Toc42581446)

[Product Licenses 7](#_Toc42581447)

[Diagnostic Logs 7](#_Toc42581448)

[Summary 12](#_Toc42581449)

[Chapter 3: System General Description 12](#_Toc42581450)

[Overview 12](#_Toc42581451)

[Architecture 12](#_Toc42581452)

[Docker 12](#_Toc42581453)

[Functionality 14](#_Toc42581454)

[Docker 14](#_Toc42581455)

[Application Features 14](#_Toc42581456)

[Chapter 4: Detailed Description 16](#_Toc42581457)

[Overview 16](#_Toc42581458)

[API 16](#_Toc42581459)

# Table of Figures

[Figure 1: RLM Licensing Environment 4](#_Toc42593862)

[Figure 2: RLM License Checkout Time Chart 6](#_Toc42593863)

[Figure 3: RLM Server Log (1 of 4) 8](#_Toc42593864)

[Figure 4: RLM Server Log (2 of 4) 9](#_Toc42593865)

[Figure 5: RLM Server Log (3 of 4) 10](#_Toc42593866)

[Figure 6: RLM Server Log (4 0f 4) 11](#_Toc42593867)

[Figure 7: Docker Architecture 13](#_Toc42593868)

[Figure 8: Full LogFile UML Diagram 17](#_Toc42593869)

[Figure 9: AnalysisResult 17](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593870)

[Figure 10: DebugLog 18](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593871)

[Figure 11: IsvStatistics 18](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593872)

[Figure 12: LicenseFile 18](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593873)

[Figure 13: LicensePool 19](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593874)

[Figure 14: ProductLicense 19](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593875)

[Figure 15: RlmInstance 19](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593876)

[Figure 16: RlmStatistics 20](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593877)

[Figure 17: ServerStatus 20](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593878)

[Figure 18: Home Page (Logged Out) 22](#_Toc42593879)

[Figure 19: Login Form 22](file:///C:\Users\andyj\Google%20Drive\School\OIT\2019-2020\Andrew%20Horn%20-%20Red%20Giant%20Diagnostics%20Report.docx#_Toc42593880)

# List of Acronyms

|  |  |
| --- | --- |
| **API** | Application Programming Interface |
| **CSS** | Cascading StyleSheet |
| **DB** | Database |
| **DI** | Dependency Injection |
| **HTML** | HyperText Markup Language |
| **HTTP** | HyperText Transfer Protocol |
| **ISV** | Independent Service Vendor |
| **JSON** | JavaScript Object Notation |
| **JWT** | JSON Web Token |
| **MVC** | Model-View-Controller |
| **REST** | Representational State Transfer |
| **RLM** | Reprise License Manager |
| **SPA** | Single-Page Application |
| **UI** | User Interface |
| **URI** | Uniform Resource Identifier |
| **VFX** | Visual Effects |

# Project Approval

Andrew Horn’s senior project for the Bachelor of Science in Software Engineering Technology was accepted by the evaluation committee and the Department of Computer Systems Engineering Technology at Oregon Institute of Technology.

## Approvals

|  |
| --- |
|  |
| Sherry Yang, Ph.D., Portland-Metro Program Director |
|  |
| Reviewer 2 |
|  |
| Reviewer 3 |
|  |
| Reviewer 4 |

# Abstract

Red Giant is a software engineering company based in Portland, Oregon that specializes in the production of visual effects (VFX) plugins for video editing suites like Adobe Premiere Pro, Adobe After Effects, Maxon Cinema 4D, DaVinci Resolve, and others. In recent years, they have been shifting their marketing and licensing strategies away from permanent, single-use licenses (serial numbers) to subscription licensing and enterprise/team licensing support. Due to the different licensing strategies and the new customer-base, Red Giant created a Volume Support team dedicated to supporting enterprise customers.

Part of the Volume Support team’s duties include troubleshooting, diagnosing, and resolving issues with the Reprise License Manager (RLM) networking licensing software, which allows large teams to share a pool of licenses among teams; These licenses are checked out from RLM when needed and then checked back in when that workstation is no longer using Red Giant software, allowing the license to be checked out by a different workstation on the same network.

One of the most common resources in diagnosing issues with a malfunctioning license server is the diagnostic log produced by RLM, which is a plain-text file containing several hundred lines of text that the support engineers read in an effort to identify any possible problems and potential resolutions. This is a time-consuming, tedious process full of opportunity for human readers to make mistakes or misdiagnose a problem.

Red Giant Diagnostics tackles many of the problems that support engineers face by providing automation and analysis of these server logs, along with saving the logs and their analysis results to a database for storage and sharing between engineers. The application exists as a web app, accessible via a public URI, secured by an email/password login authentication system.

The app was built with a DotNet Core Web API backend and a Vue.js front-end application, secured by JSON Web Token (JWT) authorization and multiple levels of user permissions: None, User, and Administrator.

# Outline

* Chapter 1: Introduction
  + Overview
  + Target Audience
  + Product Description
  + Problem Statement
  + Summary
* Chapter 2: Background
  + Overview
  + Reprise License Manager
  + Product Licenses
  + Diagnostic Logs
  + Summary
* Chapter 3: System General Description
  + Overview
  + Architecture
  + Functionality
* Chapter 4: Detailed Description
  + DotNet Core Web API
  + Vue.js User Interface
* Chapter 5: Planning, Management, and Testing
  + Test-Driven Development
  + Azure DevOps
  + Engineering Notebook
* Chapter 6: Conclusion
  + Project Summary
  + Experiences
  + Future Improvements

# Chapter 1: Introduction

## Overview

This chapter will describe the problems and challenges that Red Giant Volume Support engineers face with supporting enterprise customers and how the Red Giant Diagnostics tool solves these problems. An outline of the remainder of this report is also included.

## Product Description

Red Giant Diagnostics is a web-based application (web app) that provides a parsing and analysis engine for Reprise License Manager (RLM) diagnostic logs/reports. These logs are produced by the RLM software and contain several hundred lines of plain text information about the configuration, status, and operation of the host machine as well as the license files, Independent Service Vendor (ISV) servers, and RLM server software itself.

Typically, an experienced support engineer would receive one of these files from a customer after that customer had submitted a support request. The engineer would then manually read the log, looking for any signs of misconfiguration, hardware incompatibility, network interference, etc. Some logs are more dense than others, some issues are easier to spot than others, but all analysis takes weeks to learn and even a highly skilled engineer can overlook a problem due to human error, being stressed or rushed, or mixing up customers and logs (for example, if working on multiple tickets at once).

Red Giant Diagnostics removes the human element from the analysis and identification of problems on a customer’s license server setup, drastically improving the reliability and efficiency of identification and improving the organization of an engineer’s workload.

## Existing Products

There are no currently existing products that provide this, nor any similar, service for Red Giant engineers. This product was designed specifically for Red Giant but could be modified for any company’s support team(s).

## Summary

Red Giant Diagnostics offers a unique and targeted solution to a tedious, repetitive, error-prone, but necessary step in providing Red Giant’s Tier 1 customers with the technical support they need.

# Chapter 2: Background

## Overview

This chapter provides more information about the RLM licensing server software and the diagnostic reports it generates. Examples of problems that can be discovered by reading these logs are also provided.

## Reprise License Manager

Red Giant licenses the Reprise License Manager software from Reprise Software to facilitate their enterprise licensing solution, allowing large teams to shared a pool of licenses without having to purchase an individual license for each user/workstation.

Customers install RLM on a “server” machine, which can be any machine on their network that doesn’t need to *use* the licenses and that will remain powered on anytime those licenses may be needed by a user - This can be a virtual machine on dedicated infrastructure to an old laptop in a closet down the hall (all of which are cases seen by support engineers).

### Licensing

The image below shows an example of the physical and logical setup of a typical network and RLM licensing environment.

A close up of a logo

Description automatically generated

Figure 1: RLM Licensing Environment

In the image above, we see a logical unit containing the RLM server software and four Independent Service Vendor (ISV) blocks. RLM is designed to manage licenses from multiple ISVs at once (from one to *n*, four is just an example). We also see four workstations, all connected to each other and RLM over a local area network (LAN) using the TCP/IP protocols. The network doesn’t have to be a local network, RLM can be hosted on a virtual machine in “the cloud” as easily as it can be hosted on a local machine.

In this scenario, the customer may have 4 workstations and only one or two licenses for product **Z**, (hosted by RLM, within the ISV from the manufacturer of **Z**). This means that up to **two** workstations can license product **Z** at any one time; If a third workstation needs to use product **Z**, it will have to wait for one of the first two workstations to return their license to RLM.

### Checkout Process

**Figure 2** (below) displays the protocol used by “client” machines when requesting a license from RLM (Here, “client” refers to any machine with the licensed software that may need to checkout a license from the “server”). The software on the client workstation contains a configuration file with the IP address (or fully qualified domain name (FQDN) or public URL) of the server machine along with the primary TCP port of the RLM process.

1. Client machine reads configuration file for IP and PORT
2. Client machine sends a UDP packet to IP:PORT inquiring if ISV X exists
   1. If no response, assumes RLM does not exist at that address
3. Server responds with UDP packet to client IP
   1. If ISV X is found, RLM responds with TCP port of ISV X
   2. If ISV is not found, RLM responds with a failure message
4. If client receives a confirmation with a TCP port (ISV), client initiates a TCP connection on IP:ISV
5. Client requests license for product Y
6. ISV responds to license request
   1. If license is available, delivers license data and marks license as unavailable
   2. If no license available, delivers failure message
7. Client uses license for some duration and then returns license via UDP to main RLM process

If there is an error or denial anywhere in the steps above, the client will close the connection (if one was opened) and apply a watermark to the video output, indicating the Red Giant software isn’t properly licensed.

A close up of a map

Description automatically generated

Figure 2: RLM License Checkout Time Chart

## Product Licenses

The licenses that are hosted within their respective ISV server (within RLM) are configured with the host machine’s IP address and MAC address, along with a set number of ‘seats’ (or total available licenses) and an expiration date. The IP and MAC addresses are used to ensure that the licenses are hosted on the designated server and that they haven’t been altered or copied to illegally increase the number of available licenses on the customer’s network. All data in the license is passed through a hashing algorithm to produce a license “key” that is included in license file. If anything in the license is (purposely or accidentally) modified, it will render the entire license (and all seats it contains) null and void.

## Diagnostic Logs

If the customer has trouble with Red Giant’s software and reaches out to a support engineer, the diagnostic logs from the main RLM process are a rich source of information about the customer’s setup, network, license usage, etc. These are all potential sources of licensing problems, so the log is one of the first steps in troubleshooting any customer support request.

The images on the following pages are taken from a real diagnostic log as an example of the information they contain, the format of the data, and the potential length of a log file.

Some of the data that is contained in a diagnostic log file includes (but is not limited to):

|  |  |
| --- | --- |
| * Date/Time of log generation * RLM details   + Version   + Working directory   + Number of running processes * Host Machine   + IP Address(es)   + MAC Address(es)   + Environment Variables * Debug Logs   + RLM   + ISVs | * Licenses   + Filename   + ID   + Assigned IP and MAC   + Products     - Seats     - Expiration     - Key * Statistics   + Connections   + Denials   + Restarts |

A screenshot of a social media post

Description automatically generated

Figure 3: RLM Server Log (1 of 4)

A screenshot of a social media post

Description automatically generated

Figure 4: RLM Server Log (2 of 4)

A screenshot of a social media post

Description automatically generated

Figure 5: RLM Server Log (3 of 4)

A screenshot of a social media post

Description automatically generated

Figure 6: RLM Server Log (4 0f 4)

## Summary

The floating license method, facilitated by the RLM server software, is an integral part of Red Giant’s enterprise sales and licensing strategy. The enterprise support team heavily relies on the diagnostic logs produced by the RLM software to diagnose and resolve licensing issues for customers. These logs are long, dense, and often confusing, even for experienced engineers; They are time-consuming to read and analyze and potential issues are often overlooked or misinterpreted. This contributes to longer response and resolution times for customer support tickets, which can have large, negative impacts on the customer’s production and the reputation of Red Giant.

# Chapter 3: System General Description

## Overview

This chapter will provide an overview of the general functionality and architecture of the Red Giant Diagnostics application, including its containerization and management within Docker.

## Architecture

### Docker

Red Giant Diagnostics exists as a web app with a DotNet Core API backend, reading and writing data to a MongoDB database and a PostgreSQL database, with a Vue.js client-side application. All components are containerized in Docker containers and managed by Docker Compose, which reads the application configuration from YAML files (docker-compose and Dockerfile). This allows the entire application (front and backend) along with its two database servers to be built and optimized for production and then launched with the command:

docker-compose up -d --build

With this command, the **docker compose** application on a development workstation or production web server will read the local **docker-compose.yml** file, which contains all the configuration for each “service” to create a networked group of Docker containers. The two databases are based on publicly available Docker images, so they will only need to be downloaded to the machine before use.

The Red Giant Diagnostics application needs to be built and configured before the container can be launched. There is a Dockerfile in the root directory that gives instructions to build the production version of the Vue app, then build the production version of the DotNet Core API app, and finally, an instruction to package them together in a Microsoft ASP.Net Core container.

The two flags modify the standard actions to ensure that:

|  |  |
| --- | --- |
| -d | The containers are launched in the background |
| --build | The full build process is used to create new instances of each container |

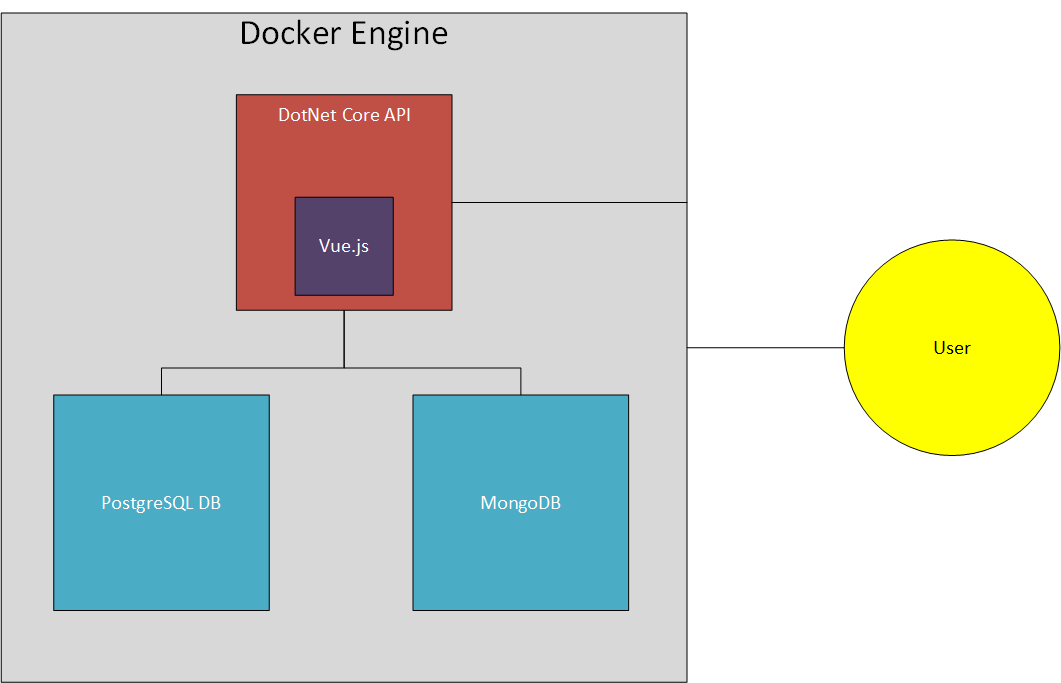


Figure 7: Docker Architecture

In **Figure 7**, we can see how the entire application is containerized and managed by the Docker Engine on the host machine. This separates the application into three main areas of concern: API, UI, and Persistence.

#### API

The API utilizes DotNet Core and the C# language to implement a RESTful interface, allowing the front-end user interface (UI) to access resources and perform actions on those resources via a standardized addressing scheme and without the need for the server to manage state.

There are three main controllers within the API, allowing users to access Logs, manage their Identity information, and a secure controller for Admins to perform maintenance tasks and edit/create/delete Identity and Log data for the entire application.

#### UI

The UI is a Vue.js Single-Page Application (SPA) utilizing the Vue Router to imitate a normal website with multiple pages, accessible via unique URIs (Uniform Resource Identifiers). The Vuex Store is used to manage application state while a user is logged in and performing tasks; The state is cached to the browser so that the user can close and reopen the site at a later time without starting from scratch. There is also an option to store the JSON Web Token (JWT), which is used to authenticate all HTTP requests from the client app, as a cookie on the user’s machine to allow them to remain logged in for up to two weeks before needing to be reauthenticated.

#### Persistence

There are two databases used to persist application data: A Mongo database and a PostgreSQL database. When an authenticated user “saves” a log, it stores the log data as a Document in the Mongo database; All user data is stored as Records in the PostgreSQL database. Only administrators can register or remove users from the application, they can also adjust user access rights (None, User, Administrator).

## Functionality

### Overview

Per the configuration files and environment variables, the DotNet Core API application is listening on port 80 (the standard HTTP port), which is mapped from the host machine’s port 80 to the DotNet Core container’s port 80. This allows all HTTP requests received on the host machine’s port 80 to be routed into the Docker container (all ports are closed by default in Docker containers, they must be explicitly opened and mapped).

When the user navigates to the URI for Red Giant Diagnostics in their web browser (which defaults to TCP port 80), it will connect to the host machine and be directed to the Docker Engine, which then redirects the request to the DotNet Core container. Requests sent to the DotNet Core container that don’t contain “/api” in the URI are handled by returning the **index.html** file from the container’s *wwwroot* folder - This delivers the production-optimized Vue.js HTML, CSS, and JavaScript files to the user’s browser.

From this point, the UI exists in the user’s browser as a single-page application (SPA), meaning that all the files needed for the application are downloaded once and exist as a single web page; Any requests for data are made asynchronously using the **Axios** library behind the scenes. Since the user never actually downloads a new web page, the experience seems seamless and responds much faster than a typical website.

### Application Features

#### Core Features

The typical workflow for a user of this application will follow this pattern:

1. Navigate to Red Giant Diagnostics
   1. This retrieves the Vue.js application files from the server
   2. The user never makes a standard request to the server again, as all requests are performed by the Vue.js app and are handled by the DotNet Core API container
2. Upload a diagnostic log
   1. The log is uploaded to the API where it is immediately parsed and analyzed, storing all log data in a single JSON object
   2. The JSON object is then returned to the Vue.js frontend
3. View analysis results
   1. The Vue.js application loads the analysis results component
   2. The JSON object is used to fill the various sections within the results view
   3. Analysis results are prominently displayed and categorized by severity

#### Authorized Features

If the user is logged in, the following functionality is unlocked:

* Log saving
  + The JSON data can be saved to the Mongo database and is assigned the current user as the “owner”
  + A title and comments can be added to the JSON data and edited for the lifetime of the log object
  + An ID is created and assigned to the log object, allowing it to be retrieved from the database
  + Logs are shareable by default; Navigating to **/api/v2/logs/{id}** will retrieve that log from the database and display the data and analysis results
* Profile
  + Each user has a profile section where all their saved logs are displayed in a paginated, sortable list
  + Logs can be opened directly from this table, allowing the user to retrieve and edit/update log details at any time
* Account Settings
  + Users can edit their Email address and Password, which are used to authenticate the user during login
  + Passwords are automatically hashed in the API; Plaintext passwords are never saved anywhere in the app

#### Administrative Features

If the user is logged in and has administrative privileges, the navigation bar will display a new icon and link, allowing the user to access the following functionality:

* User Management
  + Administrators can create new user accounts
  + Administrators can delete user accounts
  + Administrators can change user access rights between three levels:
    - None: The user account data is saved but the user cannot login
    - User: The user can login to gain access to the authorized features
    - Administrator: The user can login and access the authorized features as well as the elevated administrative features
* Log Management
  + Administrators can view all logs, not only their own
  + Administrators can delete logs belonging to any user
  + Administrators can edit the Title and Comments on any log

**Note:** Administrators cannot delete their own account, nor can they change their own access rights; These tasks must be performed by a different administrative user.

# Chapter 4: Detailed Description

## Overview

This chapter will dive into the details of the backend API and User Interface components of Red Giant Diagnostics.

## API

The backend API component is a DotNet Core Web API project, written entirely in C#. It implements the Model and Controller components of a Model-View-Controller (MVC) design pattern, where the Vue.js frontend application implements the View portion.

### Models

The API uses a series of models to structure and store data for the diagnostic logs, while it utilizes the built-in IdentityUser model provided by the DotNet Core framework (see **Future Works** for comments on the decision to use the IdentityUser model).

The primary model is the **LogFile**, which contains all data about a log, including metadata such as the user who saved the log, the date and time the log was saved to the database, a title, and user comments/notes. The figure on the following page (**Figure 8**) shows the UML diagram for the complete log model family, which separates different sections and pieces of information from a diagnostic log into classes to store that information in memory and in the database. The full diagram contains ten (10) different data storage classes, including the top-level **LogFile** object itself. Many classes contain collections of other classes, leading to a highly aggregated data storage design pattern.

The following sections will cover each of these data storage classes in more detail, presented in alphabetical order.

A screenshot of a cell phone

Description automatically generated

Figure 8: Full LogFile UML Diagram

#### AnalysisResult

The AnalysisResult class stores some information about a problem discovered with the diagnostic log during the analysis.

A screenshot of a cell phone

Description automatically generatedThe class contains two enums indicating the Type of the result (ExpiredLicense, NoLicensesFound, MismatchedIp, MismatchedMac, MismatchedIsvPort, AllLicensesInUse, NearlyAllLicensesInUse, and MultipleRlmInstances) and the level or severity of the issue (Suggestion, Warning, and Error). The class also contains a detailed message that is presented to the user in the UI.

Figure 9: AnalysisResult

#### DebugLog

A screenshot of a cell phone

Description automatically generatedThe **DebugLog** class is a very simple data-storage class, consisting of only the filename from which the output came from and a collection of strings to hold the file data.

Figure 10: DebugLog

#### IsvStatistics

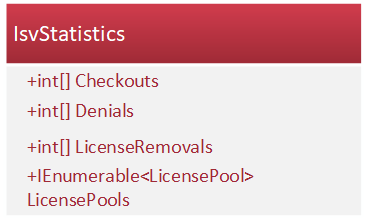
The **IsvStatistics** class inherits from the **RlmStatistics** class, which provides it with the ServerName string, StartTimes array, Messages integer array, and Connections integer array. This class extends those parameters to add additional integer arrays for Checkouts, Denials, and LicenseRemovals, along with a collection of **LicensePool** objects.

Figure 11: IsvStatistics

#### LicenseFile

A screenshot of a cell phone

Description automatically generatedThe **LicenseFile** class contains all the information about a license file present on the RLM server. These licenses contain the name of the ISV to which they belong, a unique ID number, the assigned IP address, MAC address, and primary communication port, along with an assigned ISV port, if one was given.

Figure 12: LicenseFile

The class also contains a collection of **ProductLicense** objects, representing each product for which this license was created.

#### LicensePool

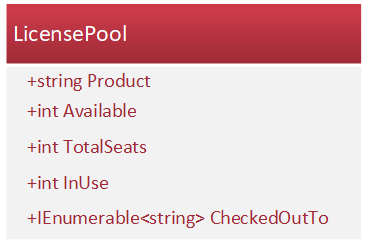
The **LicensePool** class is a simple data storage object containing the name of the Product, the total (maximum) number of seats available, the number of seats in use at the time the diagnostic log was generated, and the number of seats available (calculated by subtracting the number of seats in use from the total number of seats). This class also contains a collection of strings representing the users and machines to which licenses for this product are assigned.

Figure 13: LicensePool

#### ProductLicense

A screenshot of a cell phone

Description automatically generatedThe **ProductLicense** class contains licensing data specific to a single product (full license files often contain licenses for several products). The data contained within this object includes the date the license was issued, the date at which this license will expire, the name of the product for which this license unlocks, the total number of seats, and whether this license is for a render-only version of the product or if it unlocks the full version.

Figure 14: ProductLicense

#### RlmInstance

A screenshot of a cell phone

Description automatically generatedThe **RlmInstance** class contains information about a single instance of the RLM software detected running on the host machine. Each RLM instance object will contain the version of the RLM software, the command used to start the service, the directory in which the RLM software is located, its unique process ID (assigned by the operating system), its primary communication port, and the web port (on which it serves the administrative interface).

Figure 15: RlmInstance

It also contains a collection of integers representing alternate communication ports (this may be empty if only the primary is configured) and a collection of strings representing the names of all the ISV servers being managed by this instance.

Finally, it contains a flag indicating whether this is the “current” instance of RLM or the instance of RLM from which the diagnostic log was generated.

#### RlmStatistics

A screenshot of a cell phone

Description automatically generatedThe **RlmStatistics** class contains some information about the operation of this RLM instance, including the name of the RLM server (which is almost always “rlm”), an array of DateTime objects storing the start times for the messages and connections arrays.

These three arrays make up the header and two rows of information within a table in the diagnostic log.

Figure 16: RlmStatistics

#### ServerStatus

A screenshot of a cell phone

Description automatically generatedThe **ServerStatus** class contains very basic information about the status of an individual ISV server (multiple ISV servers can exist within one master RLM process).

Figure 17: ServerStatus

The data contained in this class includes the name of the ISV server (typically the name of the company who provided the licenses), the currently assigned ISV port (this can be assigned by the provider of the license or, if left unassigned, will be assigned by RLM on startup), and the number of restarts this ISV server has experienced.

### Controllers

This section will describe each of the API controllers along with their features and the functionality they provide to the frontend application.

#### LogsController

The basis of the application is the LogsController, which provides endpoints for each of the basic CRUD operations for the log files. There is a fully public “Upload” endpoint that allows anyone to upload a diagnostic log file to be parsed and analyzed, which returns the JSON object representing the log file without an ID or any metadata. If the user is logged in, they can then send a “Save” request to the API, which will then assign that user as the owner, assign a unique ID to the log, store the log metadata, and then save the full log object to the Mongo database.

A log can be retrieved from the controller using its ID without any authorization, this allows engineers to share results among one another or even with a customer. If the engineer wants to perform any saving, updating, or deleting of log objects, they will need to be logged into the application and they must be the log owner.

#### IdentityController

The IdentityController provides access for users to “Login,” which will authenticate their credentials and return a JWT to the frontend application; This JWT is then included in all subsequent HTTP request headers, allowing the user to be identified and to access restricted resources. Users can update their email address and password; however, they cannot change their permission levels – This action must be performed by an administrator.

#### AdminController

A separate controller exists for all administrative tasks and is secured with an extra layer of authorization, requiring the HTTP requests to come from a user who has been assigned to the *Administrator* role.

The AdminController provides the operational endpoints for many administrative tasks within the frontend application, such as viewing all application users and all logs (along with to whom the logs belong). Administrators can also edit any user’s email address, password, and permissions – Including denying the user access to the application – or deleting the user account entirely. Administrators must manually create a new user account, as no public registration method exists; This provides a layer of security so that only approved Red Giant support engineers have access to the application (aside from the public log parsing and analysis features).

Similarly, administrators can edit log metadata such as the Title and Comments; They can also reassign the log to a different user, changing the log’s OwnerId field to the new user’s ID. Administrators can also delete logs, regardless of who owns the log object.

### Services

To accomplish all the tasks necessary within the API, several services were designed and are injected into the controllers using DotNet Core’s Dependency Injection (DI) framework. DI is used throughout the API application to reduce the coupling of controllers and services, allowing the controllers to operate without worrying about what implementation of the service interface they receive during runtime.

#### IFileService

The IFileService interface encapsulates the responsibility of retrieving a File object from the HTTP POST request during a log file upload or save operation; This file object must be uploaded from the user and then streamed into memory. This service reads the HTTP request data and streams the file object into a text string containing all the text data from the log; This string is then returned to the controller.

#### IIdentityService

The IIdentityService provides a full suite of features for managing users and identity information, including authenticating users, providing APIs for all CRUD operations for both Users and Roles.

#### ILogsService

The ILogsService provides a similar set of features to those of the IIdentityService, providing APIs for the full suite of CRUD operations for log objects, along with a special “Parse” method that reads a string of raw data and returns a fully parsed LogFile object.

#### ITokenService

The ITokenService provides encapsulation for the necessary processes of authenticating, reading, and generating JWTs.

## User Interface

The client-side application consists of a Vue.js-powered SPA to provide an intuitive and stylish interface for the application users. This section will describe the features and use cases for the UI.

### Log Upload and Parsing

The image below displays the landing page/home screen for the user interface. In this case, the user is not logged in, so the navigation bar only provides two options: Home and Login.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedFigure 18: Home Page (Logged Out)

If the user clicks the Login button, a small dropdown form will appear (see right), allowing them to enter their email and password, select whether they want to remain logged in even if the app is closed (“Remember Me”), and a button to process the login request.

Figure 19: Login Form

If the user clicks the file upload form input (either in the bar or on the “Browse” button), they will be prompted to select a .txt file. Once a file is selected, they can click the “Go” button, which will upload the file to the API, where it will be parsed and analyzed; The results will be returned to the frontend application and the user will be automatically navigated to the Log Results page (see **Figure 20** below).

A screenshot of a social media post

Description automatically generated

Figure 20: Log Results (No Problems Found)

The log results page organizes all information from a log file into separate tabs, helping the engineers find the information they are looking for without overwhelming them.

These tabs include:

|  |  |
| --- | --- |
| * Results * Licenses * ISV Servers * License Pools | * Statistics * Logs * RLM Instances |

The following sections will cover the information presented in each of these tabs in the order they are presented to the user (from left to right).

#### Results

The results tab displays the results of the log analysis, including any errors, warnings, or suggestions, making any problems immediately and clearly visible to the engineer. This way, issues such as expired licenses, mismatched IP or MAC addresses, exhausted seats, and others can be spotted immediately after upload, instead of being spotted after spending time carefully reviewing the text log.

#### Licenses

The licenses tab consolidates each license file present on the server into a collapsible panel, providing further organization to the support engineers. In the image below, we can see what it looks like when the license panel is closed – The filename and expiration date are visible, allowing the engineer to quickly scan the licenses to find one they may be looking for. A small button allows them to open and close panels as they need.

A screenshot of a social media post

Description automatically generated

Figure 21: Licenses - All Closed

In the image above, the expiration date is highlighted green to indicate that the license is still valid. If the license were expired, or if no expiration date was found, this text would be highlighted red to indicate a problem.

The figure below shows an example of a license panel after it has been opened. The first section displays and compares the server details for which the license was assigned and for that which was detected on the host machine.

In this example, we can see the IP addresses and MAC addresses highlighted in green to indicate that they both have a positive match. Since the host machine could have multiple IP addresses or multiple MAC addresses, the app also ensures that the assigned information matches with the *primary*IP and MAC address of the host machine.

If there weren’t a match for either (or both), they would be highlighted red to make that mismatch clearly visible to the user.

A screenshot of a cell phone

Description automatically generated

Figure 22: License with Panel Open

In this example, we can also see the table of licensed products contained within this particular license file, including the product name, the number of full seats, and the number of render-only seats (which unlock render machines but not editing workstations).

#### ISV Servers

The image below displays an example of what is contained within the **ISV Servers** tab. In this example, there were four separate ISV servers running within this one RLM instance.

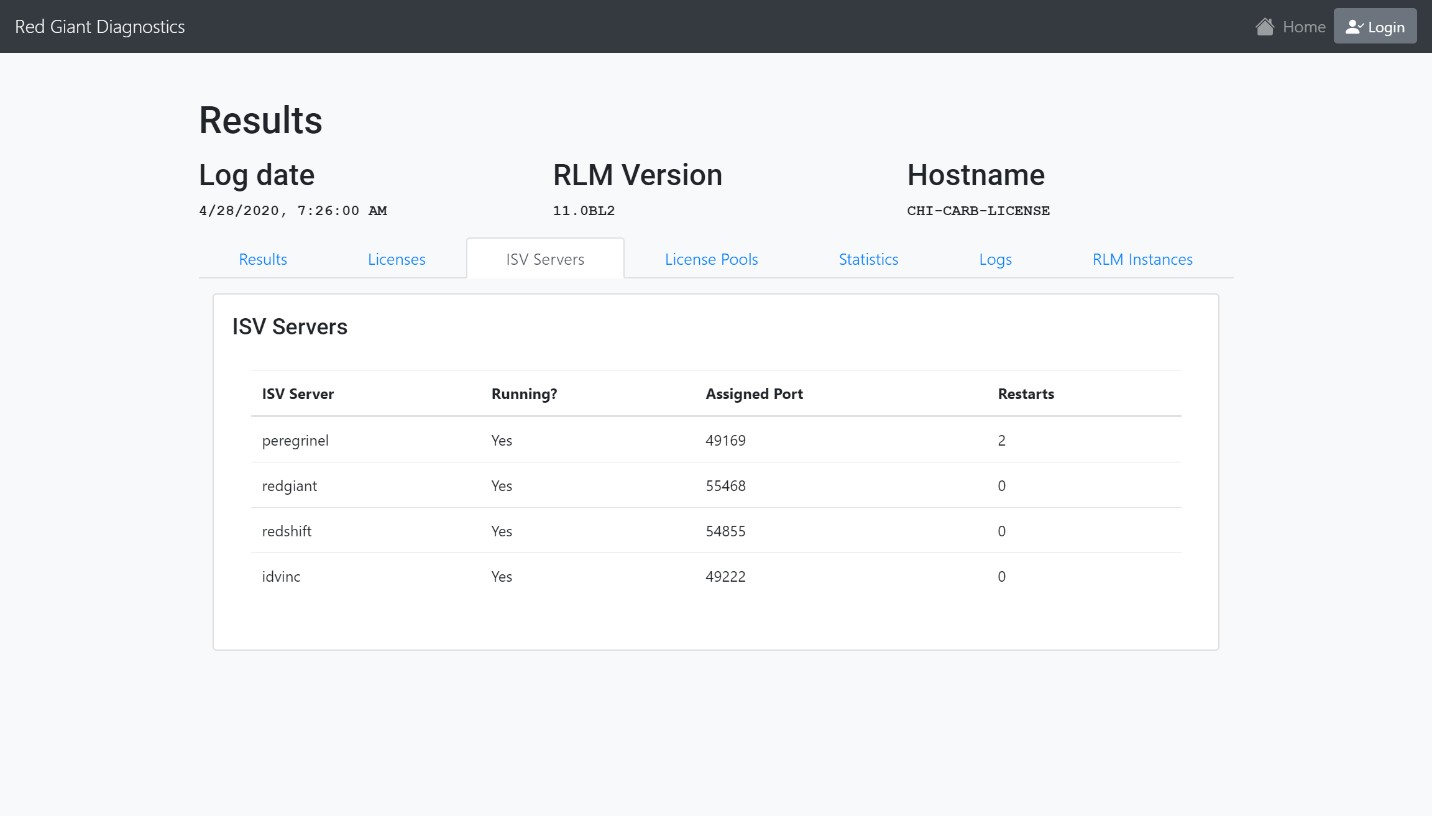


Figure 23: ISV Servers

We can see the information displayed for each ISV server includes the name, whether or not it was running when the diagnostic log was generated, the assigned ISV port, and the number of restarts since the RLM server was started. If an ISV server is not running, a “No” will be displayed in the **Running?** column and the licenses within that ISV server will not be available on the customer’s network.

#### License Pools

The figure below displays the License Pools tab, which organizes the license pools for each ISV server into an outlined and padded container, called a “Card”.

A screenshot of a social media post

Description automatically generated

Figure 24: License Pools

In this example, there is a card for each ISV server (two are visible here); Each card contains the server name, and a row for each product that has been detected in use. For each product, the card will display the product name, the total number of seats available in the license, the number of seats currently in use, and the users/machines using each seat along with the date and time that seat was checked out.

If no licenses are in use for an ISV server, the message below will be displayed.

A close up of a logo

Description automatically generated

Figure 25: License Pool - No License in Use

#### Statistics

The image below displays an example of the information displayed within the Statistics tab.

A screenshot of a social media post

Description automatically generated

Figure 26: Statistics

Each ISV server gets a card and table to display its usage and performance statistics. These include the number of messages received and the number of connections made, along with the number of successful license checkouts, the number of license requests denied, and the number of licenses forcibly retracted from a user. Each of these number is tracked for three different time periods: Since the ISV server was started, since the most recent midnight, and “Recent,” which is typically the last four (4) hours but varies (this is determined by the RLM software).

#### Logs

The image below displays an example of what is contained within the Logs tab.

A screenshot of a social media post

Description automatically generated

Figure 27: Logs

In this example, we can see that each log is organized into a collapsible container to allow the engineers to find the log they are looking for without having to wade through hundreds of lines of text output. The user can click the “Open” button to open the panel and display the full debug log output and then click “Close” to collapse the panel again.

#### RLM Instances

In the image below, we can see an example of the RLM Instances tab.

A screenshot of a cell phone

Description automatically generated

Figure 28: RLM Instances

In this example, there are a total of five (5) instances of the RLM software that have been detected running on this host machine. The number “5” is highlighted red to indicate that this could be causing problems for the customer, as there should only be one (1) instance of RLM running at a time – Multiple instances could cause conflicts over TCP ports, producing odd licensing behavior.

A screenshot of a cell phone

Description automatically generatedEach server is displayed in a table with its version number, working directory, primary and alternate communication ports, the web GUI port, its child ISV servers, and the “Is Current Instance” flag to indicate which instance is the instance that generated this diagnostic log.

Figure 29: Log Save Form

### Log Saving and Metadata

If the application user is logged in, they will be given additional options after uploading a log. A “Save” button will become visible in the log results page, above the result data and below the main navigation bar. Clicking the button will open a form (see left) containing a field for the Title and a textarea for any Comments the user wishes to save.

Once the log has been saved, or if the user opens a log that was previously saved, this button will change to say “Update” and it will contain the previously saved Title and Comments, allowing the user to modify/update the metadata and save it again. See the image below for an example.

A screenshot of a cell phone

Description automatically generated

Figure 30: Log Update Form

### User Profile

If a user is logged in, they will be given a “Profile” option in the main navigation bar at the top of the page. Clicking this link will direct them to their profile, where they can find a table displaying the Title and Comments of every log they have saved, along with a Delete button for each log that will allow them to delete the log data from the database. See below for an example.

A screenshot of a cell phone

Description automatically generated

Figure 31: Profile with Log Table

The title of each log in the table is an active link that will load the logs results page for that log, allowing users to revisit saved logs whenever needed. Clicking the Delete button will prompt the user for confirmation before permanently deleting the log from the application.

A screenshot of a cell phone

Description automatically generated

Figure 32: Log Delete Prompt

### Account Settings

Clicking the “Account” link in the main navigation bar will direct the user to their account settings page, which will allow them to update the email address and password used to authenticate when they login.

A screenshot of a cell phone

Description automatically generated

Figure 33: User Account Settings

The email address and password can be changed independently, allowing the user to change one without affecting the other. The password field also contains validation, requiring the current password as well as the new password and a matching new password confirmation. These fields will automatically display any validation errors and the “Save” button will be disabled until all requirements are met.

### Administrative Portal

If the logged in user is a member of the Administrator role within the application, an extra option will appear in the navigation bar: **Manage.** Clicking this link will direct the user to the Administrative Portal, complete with its own navigation bar and advanced features.

#### Analytics

The image below displays the main home page within the admin portal; This page displays some analytics regarding the usage of the app.

A screenshot of a cell phone

Description automatically generated

Figure 34: Admin Analytics (1 of 2)

The first chart is a distribution of logs saved to the database per month, grouped by year: As time passes, more years will be available to view. This chart allows the engineers to detect patterns in customer support requests throughout the year. The second chart (see **Figure 35** below) displays the frequency distribution of analysis results by type and severity.

A screenshot of a cell phone

Description automatically generated

Figure 35: Analysis Result Frequency Distribution

#### Logs Administration

Clicking on the “Logs” option in the admin navigation panel will load the Admin Logs view, containing a table of all logs saved in the application, including the log Title, Owner, Comments, and a Delete button. See the image below for an example.

A screenshot of a cell phone

Description automatically generated

Figure 36: Logs Administration

the Administrator can sort the logs in this table by upload date, owner, and title, as well as delete any log using its Delete button, or click on the title name to view that log’s results and details.

#### User Administration

Like the Logs Administration page, the User Administration page allows an Administrator to perform advanced functions with users of the app. In this section, Administrators can view all the users of the app in a table, which displays their email address, roles, and a Delete button for each user (excluding the admin’s own account).

A screenshot of a social media post

Description automatically generated

Figure 37: User Administration

In the image above, there are two users of the application: [admin@domain.com](mailto:admin@domain.com) and [user@domain.com](mailto:user@domain.com). The [admin@domain.com](mailto:admin@domain.com) account is serving in both the User roles and the Administrator role; They are also currently logged in, so the Delete button is not present in that table row. The [user@domain.com](mailto:user@domain.com) account is serving in only the User role; Since they are not the currently logged in user, a Delete button is present to allow the admin to delete that user account.

Selecting one of the user accounts in this table will prompt a user edit form to appear below the table (see example below).

A screenshot of a cell phone

Description automatically generated

Figure 38: Administrative User Edit Form

In this example, we can see all the fields that an administrator of the application can edit for any user account, including the user’s email address and password, as well as their assigned user roles.

If the user is set as an Administrator, they are automatically also assigned to the User role. If the user is removed from the User role, they will not be able to serve as an Administrator and they will also be prevented from logging in to the application at all.

#### User Registration

The only way to create a new user account in the app is for an administrator to create the account on behalf of the new user. To do this, the administrator must navigate to the administration portal and click on the **User Registration** link in the navigation panel. This will open the user registration/creation form (see below for an example).

A screenshot of a social media post

Description automatically generated

Figure 39: Admin User Registration

All fields are required, and the Password and Confirm Password fields must match before the Submit button is enabled. The image below gives an example of an invalid password state and the messages that are displayed to the user.

A close up of a logo

Description automatically generated

Figure 40: Mismatched Password Validation Message

# Chapter 5: Planning, Management, and Testing

## Overview

This section will provide details on the project management and project planning tools and strategies used, as well as the application testing performed.

## Project Planning and Management

### Scrum

This project followed the Scrum project management paradigm, with sprints lasting two weeks (including weekends). I restarted the project late in the year, beginning on April 6, giving me a little under five (5) sprints to complete the project from scratch. I had begun basic planning, requirements gathering, and early design over the summer, so when I began my first sprint, I was able to immediately build my backlog and begin working on tasks.

At the beginning of each sprint, I spent the first day planning my tasks and grooming the backlog until I had a collection of tasks and items that I felt I could complete in two weeks. I didn’t perform sprint retrospectives, nor did I have any kind of daily “stand up.”

### Azure DevOps

Prior to this project, nearly all my experience with version control was with Git and GitHub; All my project tracking experience was with Trello. Because I wanted this project to be as much of a learning experience as possible, and because I wanted to use tools that were more likely to be used in industry, I decided to use Git as my version control system, but to use Azure DevOps as my primary remote repository and my project tracking tool.

Azure DevOps provides a software development repository and project management suite, including everything from version control repositories, backlogs and sprint planning boards, to “Pipelines” for Continuous Integration/Continuous Development (CI/CD). For this project, I would use the version control repository, sprint planning tools, and some of the CI/CD tools.

By the end of the project, all branches merged into the Development branch are automatically built and all 423 unit tests would be automatically run. Any merge requests from any branch into the master branch would have to be built and all unit tests had to pass before it would be allowed to merge – My goal was to only merge from Development into master and all features and bug-fix branches would be merged into Development until a milestone or task was complete.

## Testing

I followed a Test-Driven Development (TDD) paradigm during the development of the DotNet Core API application, which drove many of the design choices for the Controllers, Services, Factories, and other interfaces. At the end of the project, I had 423 unit tests for the DotNet Core API project that covered every sunny and rainy-day scenario for each method of each Controller, Service, Factory, and Helper class.

Unfortunately, because I started the project so late in the year, I ran out of time to implement unit tests for the Vue.js frontend application. I did spend time researching a few different testing frameworks and had even starting learning how to write unit tests using the Vue Test Utils and Jest frameworks but ran out of time to implement them.

Throughout the project, I had the engineers at Red Giant perform usability testing to provide feedback on the application, report any bugs, request additional features, or to talk about any potential improvements.

# Conclusion

## Project Summary

I feel proud of what I accomplished with this project, especially given the circumstances (moving from Oregon to Alabama in August and finishing school entirely remotely) and the short amount of time I gave myself to re-develop the application using entirely new tools and frameworks (the first version/prototype was developed using Node.js and Express.js for the backend and Pug.js templated frontend).

At the time of this submission, the application is hosted on a live web server so that the Red Giant support engineers can use it in their daily tasks. I plan on registering a domain for it, as it is currently only addressed by a public IPv4 address.

## Experiences

After developing the same application using two entirely different technology stacks, I am glad that I decided to use DotNet Core and Vue.js instead of relying on Node.js and Express.js with a simple, templated frontend. To me, DotNet Core feels like a more professional, high-performance, and more reliable backend framework than Node.js and Express.js. I know that Node.js and Express.js are very widely used throughout industry, but they always felt like a more “hackish” way of building a web application; Possibly (probably), that’s because I never treated them like a professional-grade technology.

## Future Works

There isn’t much that I have left to do or would like to do for the application, if I were given more time to work on it. I would like to implement full unit-testing for the frontend and even some automated integration testing using Selenium. I do plan on supporting the application for the foreseeable future and incrementally improving its analysis engine to provide more functionality and value to the engineers at Red Giant.

However, I think this is a solid and reliable tool and I am proud of all that I have learned.