Douglas Swanson

Boeing  10/16/2022

Genesis: Simulation Configuration Management

Contents

[Summary 2](#_Toc116860111)

[Introduction 2](#_Toc116860112)

[Reason 2](#_Toc116860113)

[Objectives 2](#_Toc116860114)

[Innovation and Uniqueness 3](#_Toc116860115)

[Scope 3](#_Toc116860116)

[Organization of Proposal 3](#_Toc116860117)

[Literature Review and Related Work 3](#_Toc116860118)

[Methodology and Design 4](#_Toc116860119)

[Overall Design 4](#_Toc116860120)

[The Simulation Environment 5](#_Toc116860121)

[Scenarios and Templates 6](#_Toc116860122)

[Databases and Data Model 7](#_Toc116860123)

[Service Broker 9](#_Toc116860124)

[Services 9](#_Toc116860125)

[Frontend Application 10](#_Toc116860126)

[Documentation and Testing 11](#_Toc116860127)

[Deliverables 12](#_Toc116860128)

[Schedule 13](#_Toc116860129)

[Gantt Chart 13](#_Toc116860130)

[Milestones 13](#_Toc116860131)

[Completed Work 14](#_Toc116860132)

[Budget and Justification 14](#_Toc116860133)

[Sources 15](#_Toc116860134)

[Appendix A: Personal CV 16](#_Toc116860135)

[Appendix B: Current Project Code 16](#_Toc116860136)

# Summary

The GENESIS application is a proposed solution to the configuration of a Space Simulation currently in use by Boeing’s Virtual Warfare Center (VWC), which is a part of Boeing’s PhantomWorks division. The current problem with the simulation is that it is very difficult to create and configure a new simulation scenario in a reasonable amount of time. This application would allow the team to not only increase the speed in which they can develop a new scenario but also validate its inputs. This validation would give the team more confidence in the simulation when they run it either during an Operator In The Loop (OITL) event or for internal research and development studies. The GENSIS project will take the form of a flexible and high-speed framework that would allow the team to generate a new scenario form any network within the VWC. It would also allow the team’s software engineers to update and modify various modules to increase the capabilities of the software. All of this can be done using web technologies which would allow for containerization and deployment on any kind server. This project will be a dive into a method in which the team feels promising but still doesn’t have enough information to decide. So, in other words this a is test framework that may be flushed out further upon delivery or may even have some of it’s components completely redesigned. So this is a project to try new things and to see how some of these concepts work when actually implemented.

# Introduction

## Reason

The GENESIS simulation configuration management tool is a proposed solution to the slow turnaround and complex configuration of the Space Sim. The Space Sim is a simulation tool currently in use at the Boeing’s Virtual Warfare Center. This simulation allows the Space Team to run operator in the loop (OITL) training events and provides an environment to simulate and test future space architectures. The space team currently uses a Scenario Master Spreadsheet (SMS), along with a suite of accompanying software, to initialize scenarios and administer changes to objects in the simulation. The SMS is an Excel spreadsheet which currently gets the job done but, when more than one person starts editing a scenario it becomes an exercise in manually tracking changes.

## Objectives

In order to streamline the task of scenario generation, the GENESIS project should keep the following objectives in mind. Should be designed and built with modules in mind, like a modular framework. The inputs are key and should allow for validation. The framework should be well documented to allow for future developers to easily integrate new modules or fix current ones. The framework itself should have unit tests written. This framework should have some hooks written in to display the current scenario state to the user using proprietary tools like stellar. These tools would be rolled into the framework after it is delivered to the customer and would not be available until then. Should allow for the user to create a new scenario or load an existing one. Generate the appropriate schema from a configuration file. Populate the generated schema with data entered by the user. Display input forms so that a user can easily provide the data to be used during a scenario.

## Innovation and Uniqueness

GENESIS isn’t necessarily innovative, but it is a more focused configuration management tool. Most of the tools on the market seem to be aimed at any solution on any platform anywhere. This project is specifically designed to handle the setup and configuration of the Space Simulation for the VWC and that is it. It needs to be specific in order to provide it’s highly specialized users the flexibility and robustness they desire. It also needs to be flexible enough to change at the same speed as the simulation itself. So, this tool is a super specialized configuration management tool designed to run in the Virtual Warfare Center’s private networks, sometimes without access to the greater internet.

## Scope

The scope of this GENESIS project for this applied project course is as follows. The project should define the basic framework for the project. This framework should allow a user to generate a new scenario or load an existing scenario. Should allow the user to generate basic spacecraft data for this particular scenario through a webpage interface. Should generate the database schema and populate them with the provided data.

## Organization of Proposal

This proposal will walk through the literature read to come up with the design being proposed and some of the related work. It will than jump into the Methodology and Design of the proposed GENESIS application framework. In this section the researched design will be presented along with some alternatives to this initial design should some of the lesser know or risker sections encounter problems. After covering the design of the framework, the deliverables of this project and the schedule to deliver those items will be discussed. A brief section on the budget and its justification will be discussed but, this section is short due to the lack of funding.

# Literature Review and Related Work

Currently the literature reviewed was used to get a general basis of what configuration management (CM), Service-Oriented Architecture (SOA), and Services are. Configuration Management is defined, by the Department of Defense as a system that “provides and orderly way to facilitate change...” (3). There are currently a few different software suites that perform software configuration management. One of the most well-known solutions would be Docker, which “takes away repetitive, mundane configuration tasks and is used throughout the development lifecycle for fast, easy and portable application development” (6). Docker’s direct competitor, Kubernetes is another software configuration management tool that handles much of the same activities using the same containerization model (8). On the far end of the software configuration spectrum there is Ansible which advertises itself as a full-service automation platform allowing its users to bring automation to their containers, security, applications, infrastructure, and cloud (1). On the opposite side, there is a tool called Config Cat which allows for a quick method of toggling features on individual applications (4).

A Service is “something useful a provider does for a consumer” (7) which can be anything ranging from a product on a shelf or something less tangible such as insurance or providing access to the internet. A service in regard to software is an application that has an interface that can accept requests from other sources, perform some action, and return a response. The action is what is desired by the source making the request. This interface is referred to as an application program interface (API) and it defines how to make a request of the service in question. There is a standard attached to this API called REST, which stands for Representational State Transfer. It outlines some rules that the API should follow. For instance, the API should use HTTP Methods explicitly, be stateless, outline its routes in URI like structure and accept and respond with data formatted using JSON (JavaScript Object Notation).

The services that directly deal with data manipulation are usually called CRUD services (11)(15), where CRUD stands for the actions it performs on the data. The actions are Create, Read, Update, and Destroy. These actions are mapped, if we are using REST principals(13), to specific HTTP requests. For instance, GET requests are mapped to Read actions, POST requests to Create actions, PUT requests to Update, and finally DELETE requests to Destroy actions.

A service broker is one method of providing the registry and discovery of services in applications designed to the SOA model. This application provides a central location in which the services running in the system can register themselves as available to receive requests(14). It also provides a method in which any component within the system can find out where any service within the system is running in order to send requests to it. Sometimes there is more to this application, especially in larger systems where multiples of any service may be running. This additional feature will provide the requesting entity with the address of the service with the least amount of work todo. This is called load balancing.

# Methodology and Design

## Overall Design

The Genesis application is designed with a Service Oriented Architecture, that way each of its components may be run on any number of networked machines within any company network. To do this each GENESIS module is written as a service that will have an API in which the other applications can request data from them. Each of these services will register itself in the Redis database using keys that expire after a specified amount of time. These expiring keys force the services to send keep alive messages to the database to receive requests from the rest of the system. It will also remove a service from the database should that database fail and be unable to remove itself. These services will have RESTful APIs in order to standardize the interactions with them throughout the system. These APIs will be well documented so that no one must go into the code to find out what the service features are. The user interfaces will be run off a single Node.js server that will both serve the page to the user as well as allow that page to access the rest of the application, wherever it may be running. These user interfaces will be written as react applications and will allow the user to perform CRUD actions on the GENESIS scenarios and templates and on the scenario specific data, depending on which templates that the scenario has been configured to use. Figure 1 below shows the most complex configuration of this framework, each service is running on a separate machine. This is the operational case taken into account when designing the main design for this application framework and any simpler case will work, like all of the components running on the same development machine. As it turns out, this simpler case will be the fall back case should the fully distributed model be an unreachable goal during the time provided by this course.

Diagram, schematic

Description automatically generated

Figure 1: GENESIS Application Interactions

## The Simulation Environment

The Space Simulation can run on a network of computers, some of these computers are virtual machines that run the databases as services and some are configured as terminals in which a user can interact with the simulation. GENESIS will be designed to run within this environment, hence using SOA and building the various components as services. This will grant the application the flexibility to run on any of the open machines within the network without necessarily being tied to a particular machine, except for the database services. This exception will actually be beneficial for the service discovery portion since all of the components will know where the Redis database is in order to register itself or find where the other services are running. One initial thought was to run all of these services in a Kubernetes cluster, but after further research, setting up and maintaining said cluster would take most of the semester. Kubernetes allows for a lot of users to access an application as it attempts to load balance by spinning up additional services and allows for the application to grow over time. This is a lot of overhead work for this particular application since there will only be a handful of users at any given time and the application doesn’t need to be running 24/7, only while a scenarios is being constructed. So, the user can run any of the GENESIS components through preconfigured batch command files at the time of use, on any of the machines they are currently working on. Regardless of this fact, GENESIS will be designed to run across any number of machines within the network or be placed into containers to run on Kubernetes in the future, should the need arise.

## Scenarios and Templates

In GENESIS we are dealing with generating and managing scenarios to be used within the space simulation environment. These scenarios will have a name, a description, a time that it was last updated, a unique key, a flag denoting if the scenario tables have been generated, and a list of templates that define the scenario's constituent modules. A module is an application or group of applications that model an aspect of the simulation environment. For instance, the Space Domain Awareness (SDA) module of the simulation is made up of 4 separate applications that allow players to schedule sensors to go look at space objects in order to determine their current orbits. Another module would be the Comm Effects (CommFx) module which handles all of the communication networks used by the satellites, which is comprised of 1 full application and some scripting in another. The templates will define the data required by the module and will let GENSIS know how to store, configure, retrieve, and validate this data. The template will configure the data table, within MySQL, through preprepared SQL statements that will define the tables in which the data is stored. The data model in the template will be used by the interface to develop a form, or series of forms, to collect the information from the user. In order to determine if this data is correct, the user interface will also use the defined data model to perform validation on the user's input. These templates will be directly related to each version of a particular module, and as that module changes so too will it's template. So, each template entry in the database will have a name, a version, a file path to its configuration files, and a brief description.

The current idea is to store the specifics of a template within a folder in the file system and create a record in the GENESIS scenario database pointing to that directory. The data model, SQL statements, and application configuration model will reside as flat files within this directory. One alternative to this would to use either the MySQL or Redis NoSQL database to store this data. Both of these databases would provide data security and an easy method of retrieving the records, through queries. The downside to using these databases would be trying to define these template models using the relational or key value store, depending on which is chosen, paradigms. The file system seems like a good way to start because it would allow for quick development and data flexibility of the templates as the rest of the GENESIS solution is created, at the cost of retrieval speed. Once GENESIS has been developed further and gone through a few iterations, this data may be migrated in order to gain the benefits of the databases.

In order to properly use these templates the implementation will answer the following questions. How will the data model be formatted for proper storage in the flat file system. Will there be dependencies used between templates? If so how will they be defined? How will the user interface forms be generated from the data model? How will they validate the data that will go into the data model? How will the CRUD service for this module be defined in such a way that it can handle different iterations of data models and thus different database schemas.

Since this is the risky / innovative part of the application, the backup plan for the templates would be to get rid of them and head back and hard code the values into the services. Not a very nice solution since each module version would need it’s own service version as well. The service would also need to register with its own version so that the rest of the applications would have to check the version as well as the IP address in the service broker, see section below for more information on the service broker.

## Databases and Data Model

Since the space simulation environment currently has 2 databases running, the GENESIS framework will make optimal use of both of them. It will use the relational MySQL database to store the scenarios, templates, and scenario specific data. The first 2 will be stored in a GENESIS application specific schema that will contain this information which is useful by only GENESIS. The application itself will generate a new schema for each scenario that it creates, defined by the templates selected for that scenario. Redis will be used as the service broker within the framework, it will store each service’s IP address that way the front end knows where to send its requests for data to.

|  |  |  |
| --- | --- | --- |
| Column | Data Type | Description |
| PKey | int | Unique identifier for this module template, will be automatically assigned to each record by the database |
| Name | varchar(25) | A human readable identifier for this template |
| Version | varchar(10) | The version of software this template belongs to |
| FilePath | Text | The directory file path to the folder containing the template’s files. |
| Description | Text | A more in depth description of the template and its purpose. |
| Deprecated | bool | A flag indicating that this template is old and has been replaced with a newer version. |

Table 1: Template Data Model

|  |  |  |
| --- | --- | --- |
| Column | Data Type | Description |
| PKey | int | Unique identifier for this scenario, will be automatically assigned to each record by the database |
| Name | Varchar(25) | A human readable identifier for this scenario |
| Description | Text | An explanation of the purpose and or event that this scenario was used to support |
| Date Last Updated | Date Time | The last time that this scenario was updated by any user |
| Templates Used | Foreign Keys | The id’s of the templates used to create this scenario |

Table 2: Scenario Data Model

|  |  |  |
| --- | --- | --- |
| Column | Data Type | Description |
| PKey | int | Unique identifier for this scenario, will be automatically assigned to each record by the database. |
| Name | Varchar(25) | A human readable identifier for this scenario |
| Constellation | Varchar(25) | An explanation of the purpose and or event that this scenario was used to support |
| Rx | double | The starting x position of the vehicle |
| Ry | double | The starting y position of the vehicle |
| Rz | double | The starting z position of the vehicle |
| Vx | double | The starting x velocity of the vehicle |
| Vy | double | The starting y velocity of the vehicle |
| Vz | double | The starting z velocity of the vehicle |

Table 3: Spacecraft Module Spacecraft Data Model

|  |  |  |
| --- | --- | --- |
| Key | Data Type | Description |
| genesis.broker.[service name] | string | The ip address of the service specified in the key name. Set to expire after 30 seconds to aid in service clean up in case of service failure. Service should readd this key within the 30 second interval to show that the service is still up and running. |

Table 4: Redis service key model

The tables above contain the definitions of the data that the GENESIS application will be making use of. Tables 1 and 2 have the definition of the Template and Scenario files that are the data used to define these objects within the framework. Each scenario is made up of a number of module templates that define the data used within the scenarios. In this case we have a data model used in the initial spacecraft module template, as shown in Table 3. This data will be used by GENESIS to create an appropriate table within the scenario specific schema. This data template will also be used by the module service to perform CRUD actions on this table within the scenario schema and by the front end to validate the data that the user entered. Table 4 describes the key used within the Redis NoSQL key value based database for a registered service and how it should be used as a basic string type value.

## Service Broker

A service broker in the SOA paradigm is an entity that knows what and where all of the services required by the application are running within a given environment. The broker will also handle requests from users in which service they should use, should there be more than one of the same service running at any given time. Since the number of users, at any given time, are low for the GENESIS application there will only be a single service running for each module. Knowing this, the service broker doesn’t need to have any logic regarding which service should be used and will return the information for the only requested service registered. The broker will need to handle service registration, service deregistration, service requests, and service failures. All of this can be handles by the key value store of the Redis database in the simulation environment. When a service registers it will create a key for itself containing its ip address. This key will expire after a 30 seconds of inactivity, so the service will need to update the key within 30 seconds or it will “deregister”. This method of handling services accounts for the case in which the service were to crash for some reason, it will automatically be removed from the broker and no more requests will be sent to it. If a client requests a service, and the service hasn’t been registered within the broker, they will be informed that the service is not available and they will need to go start the service. This process of not finding a service and than automatically starting it is one of the features of using Kubernetes, but as explained above, that will be a future add on. The service keys will be registered under the “*GENESIS.broker”* key space in Redis, so to find any running services, a requestor will search for keys of the “*GENESIS.broker.\*”* key type. After getting these key names it will search for the service(s) desired and make another request from Redis to get the IP address in which that service is currently running.

Should this implementation of the service broker fail to come to fruition, the service broker could be built into the Genesis scenario schema in MySQL and act in the same manner. Should that fail as well, it will be assumed that the services are running on the same machine as the database are within the environment. See figure 1.

## Services

The interesting part of this GENESIS application is the services themselves. They are built as RESTful CRUD services. The RESTful part of the service insures that it communicates through its API in a logical manner, see the literature review section for more details. The CRUD part tells us what it is doing to the data it is attached to, in this case Create Read Update and Destroy. These applications are built like this so that they can run on any machine within the network and be able to respond to requests made by the users through the front end application. To be able to run on any machine in the network, the service must tell the service broker which machine it is currently running on by registering itself under the machine’s IP address. Each of these services than has an application programming interface, API for short. This interface allows other applications to make requests from this service, but that is not all, it lets the developers of the other applications know how to make a request and what to expect as a response. All of these requests and responses are made with the HTTP protocol, as required by the REST standard.

The first service to be designed within the GENESIS framework is called the Genesis Scenario Service. This service is responsible for the CRUD actions on the Scenario and Template records within the Genesis Scenario schema within the MySQL database. On startup it will register itself as the Genesis Scenario Service with the service broker and await requests from the users. Upon receiving a request it will log the request itself, perform the desired action, logging any errors encountered, log the response, and than send the response to the requestor.

The second service will be designed for the Spacecraft Module of the space simulation environment. It will deal with the CRUD actions for the data model defined in the spacecraft module template selected for a specific scenario. It will do this by being general enough to accept any data model and apply it to any scenario specific schema within the database. It will follow the same process as the first service, with the registering and responding and all.

Both of these services will be written as separate Node Js “servers” using the open source Express package to assist with the request and response routing. It will also utilize a package called Winston as it’s logging library.

Should this initial course of action fail, possibly due to the complexity of dealing with multiple dynamic data models, a simpler course of action will be taken by assuming that the data model will be the same for each template. Kind of takes the use out of the template but that can be addressed at a later stage of development in the GENESIS application.

## Frontend Application

The frontend application would allow the users of the GENESIS application framework to dynamically create and configure space sim scenarios. This will be done by following the process laid out in figure 2 below.

Diagram

Description automatically generated

Figure 2: Front end logic

This application front end consists of 2 parts, the Node Js “server” that serves the pages to the user’s web browser and the application being served to the user. This server will be built like the services described above but will respond with the user interfaces defined by the URL routes requested by the user. It will also be the point that goes out and finds the services required by these user interfaces. It will do this by querying the service broker for the services required by GENESIS application as well as the templates that define any given scenario. Upon selecting a specific scenario, this server will also pull all of the required template files from the flat file storage location so that the user interface has knowledge of the data models required by the templates. The user interface section will have a fairly static form that allows the user to select or create scenarios and define templates used within the GENESIS framework. The part that gets interesting is the dynamic forms to fill out the data in a manner that is acceptable to the templates used within a given scenario. These template data models will define which fields need to be collected from the form as well as what values are acceptable. The form will use these acceptable values to validate the user’s input before the data is sent off to the modules data service.

Should this not work as planned, this front end can be simulated through prebuilt HTTP requests made to the rest of the application. These requests can be tied to specific buttons on a UI and a box can be used to display the data received as part of the response. Not super useful for collecting the data but it will show that the rest of the application is working as expected.

## Documentation and Testing

This GENESIS application will have unit test written as part of each of it’s component pieces, for instance the Scenario Service will have unit tests written to test it’s business logic and will have some prebuilt HTTP requests constructed to attempt to model all possible requests that the service may receive from outside sources. The unit tests will be written using the JEST library and the HTTP requests will be made through VS Code’s Thunder Client extension. These tests will allow the development team to continuously add to the GENESIS application in the future and be able to quickly tell if the changes made affected any of the expected results without having to manually test the application each time.

The application documentation will describe the architecture of the application and describe what each component is responsible for. It will describe the tests and what their expected results are. This documentation will be used to get the development team up to speed and allow them to answer questions about the application and it’s inner workings. It should give insight into how to configure and run the GENESIS application framework in any software environment.

Should these two things fail to be built into the application, it would not effect the operation of the application framework but will need to be added in shortly after the product is delivered in order to provide the benefits described above.

# Deliverables

The customer of the GENESIS application expects a few deliverables, the software of the GENESIS application, the tests used to validate the software works as designed, the documentation of how the software works, and a presentation of said application to the development team.

The software will be delivered through GitHub since it is online and can be accessed from a computer on the Boeing internal network. After the software has been retrieved, GENESIS will be configured to run in one of the development laboratories on the Boeing network. This configuration will also include setting up the GENESIS scenario database in MySQL. Next the unit tests will be ran on the application to show that the framework can and will work within the simulation environment. The application documentation will be delivered as the markdown files it was written in, all part of the GitHub repository downloaded. This documentation will be added to the internal documentation wiki so that the entire team can access and read it.

The presentation will get the development team up to speed on how the GENESIS application framework works and how they can go about modifying it to work with the proprietary data of the space simulation. This will include a demonstration of the working software as well as a talk about where the team will like to take the software in the future. The presentation will include a copy of the final report written for this course about the application created.

The system and hardware will be provided by the company.

# Schedule

## Gantt Chart

Graphical user interface

Description automatically generated

Figure 3: Project Gantt Chart

## Milestones

The GENESIS project is comprised of several milestones to complete the project. The first milestone is to build the Genesis scenario CRUD service. This milestone will complete the service that will handle all of the scenario and template data required for the GENESIS application framework to work properly. This task should take a week to complete and will include the service, allocated 5 days, as well as documentation and testing, 3 days to complete both. The next milestone is the Spacecraft Module service, which will handle the data required by the spacecraft module. This service is almost a direct copy of the service created in the first milestone, with some minor nuances with the data model and how to query it. Because of this fact, this milestone has been allotted 4 days in total to convert the first service into this new service. 2 days in converting the actual service itself and 2 days for the documentation and testing. The third milestone is the construction of the service broker, which has been allocated 5 days to complete. This will include setting up the data model, 1 day, and writing code into the services to register themselves, 3 days. It will also include the code in which the other GENESIS framework components will use to find and get the service data, should take about 3 days. The fourth milestone, and probably the most time intensive, is the Front End of the GENESIS framework. This has been further broken down into 2 pieces the development of the NodeJS “server” and the React UI component. Each has been given 5 days to complete. This is the most intensive task because user interfaces usually take a lot of tweaking of minor elements to get them to look presentable. The final milestone is a catch all bucket for all of the documentation and testing that may have fallen through the cracks in the previous milestones as well as documentation and testing of the framework as a whole. This milestone has been allocated 7 working days to complete.

## Completed Work

The GENESIS project has already started being developed and has a few things already completed. The overall design of the project has been flushed out and the component responsibilities have been determined. The first service is almost complete, it has all of the CRUD interactions with the database completed just needs to flush out the API routes that call those interactions from requests. It also needs a method to log to a source using middleware as well as completing the API documentation and testing. This service will be directly ported to the second service and then tweaked to work with a dynamic data model. Along those lines, all of the data models have been determined and flushed out for the services to use, either in their CRUD interactions or in registering themselves.

# Budget and Justification

The GENESIS project will be developed with no budget. This project is basically research into one method of creating this application framework. So, knowing this all packages used by the development team will be self generated or found through open-source means. The software will be developed on the team’s personal machines using free software for documentation, testing, and development work.

# Sources

1. Ansible, Red Hat. “Ansible Is Simple It Automation.” Ansible Is Simple IT Automation, Red Hat, <https://www.ansible.com/>.
2. CHEN, YINONG. Service-Oriented Computing and System Integration: Software Iot Big Data and AI as Services. KENDALL HUNT, 2017.
3. “CM Overview.” Chief Information Officer U.S. Department of Defense, United States Department of Defense, <https://dodcio.defense.gov/Library/DoD-Architecture-Framework/dodaf20_cm_overview/>.
4. ConfigCat. “Configcat Feature Flag Service.” ConfigCat, ConfigCat, 1 Mar. 2022, <https://configcat.com/#product>.
5. Dikmans, Lonneke, and Ronald Van Luttikhuizen. *SOA Made Simple*. Packt Publishing, 2012.
6. “Home.” Docker, 24 Aug. 2022, <https://www.docker.com/>.
7. Jiang, Peishi, et al. “A Service-Oriented Architecture for Coupling Web Service Models Using the Basic Model Interface (BMI).” Environmental Modelling and Software, vol. 92, June 2017, pp. 107–118., <https://doi.org/https://doi.org/10.1016/j.envsoft.2017.01.021>.
8. “Production-Grade Container Orchestration.” Kubernetes, The Linux Foundation, <https://kubernetes.io/>.
9. Sen, Kaushik. “Top 10 Configuration Management Tools You Need to Know about: Upguard.” UpGuard, UpGuard, Inc, 12 May 2022, <https://www.upguard.com/blog/configuration-management-tools#toc-2>.
10. Whyte, Jennifer, et al. “Managing Change in the Delivery of Complex Projects: Configuration Management, Asset Information and ‘Big Data.’” International Journal of Project Management, vol. 34, no. 2, Feb. 2016, pp. 339–351., <https://doi.org/https://doi.org/10.1016/j.ijproman.2015.02.006>.
11. “Crud - MDN Web Docs Glossary: Definitions of Web-Related Terms: MDN.” MDN Web Docs Glossary: Definitions of Web-Related Terms | MDN, <https://developer.mozilla.org/en-US/docs/Glossary/CRUD>.
12. Pattinson, Tamara. “Relational vs Non-Relational Databases.” Relational vs Non-Relational Databases, 19 Oct. 2021, <https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases>.
13. Rodriguez, Alex. “Introduction to RESTful Web Services.” IBM Developer, Nov. 2008, <https://developer.ibm.com/articles/ws-restful/>.
14. “SOA Service Registry/Repository and Service Discovery.” Coforge, https://www.coforge.com/blog/soa-service-registry/repository-and-service-discovery.
15. “What Is Crud?” Codecademy, <https://www.codecademy.com/article/what-is-crud>.
16. Personal CV

CV will be submitted as a separate document for ease of viewing

1. Current Project Code

Code will be submitted as a separate zip file for conciseness and viewing in an IDE versus word document format.