UNIVERSITY OF REGINA

ENSE 477: SOFTWARE CAPSTONE PROJECT

SOFTWARE SYSTEMS ENGINEERING

Workshop Enterprise Resource Planning Suite System and Object Design Document

AUTHORS
Jonathan Wells
200328640
Konstantin Kharitonov
200354502

SUPERVISOR
Karim Naqvi
M.A.Sc., P.Eng.

April 9, 2019

Contents

List of Figures

1 Introduction

The Workshop Enterprise Resource Planning Suite is an administrative overview and task management software designed for the University of Regina's on-campus engineering workshop. This web application allows the workshop manager to access and process workorders, manage the inventory of the workshop and manage each project in the shop. The application can be accessed through any modern browser and has two main components, the frontend user interface for the workshop manager, and the backend which responds to the frontend requests.

The program is designed as a web application using the ASP.Net web API framework, written in the C# language, to handle the backend functionalities of the application. In order to bridge the backend with the database of the application, the system uses Entity Framework core, which allows for the backend to retain full functionality without referring to a predetermined database model.

The frontend makes use of a javascript framework called Vue.js, which allows for dynamic user interfaces and web applications that handle multiple events one page, much like the ERP suite. This allows the program to do multiple complex tasks on the same page, such as handle multiple workorders, and showcase the data that is stored on the database via the backend.

1.1 Background

Previous to the design of the ERP, the engineering workshop primarily stored all workorder data physically. All workorders were submitted in person on workorder forms, which are then stored into filled binders for recording purposes. All workorders ever submitted are kept in these binders and since they are filled out by hand, there are no copies made. Workorders, previous to this program, have not been copied.

1.2 Purpose

This system was designed to replace the previous methods of workorder, time, and inventory tracking, centralizing all aspects into one powerful application that can be accessed online. Workorders currently must be submitted via paper form directly to the workshop during its operating hours. The form must then be reviewed by the workshop manager and if accepted, future meetings are scheduled. All workorders submitted are then stored physically in binders, which date back to the opening of the workshop. All materials and inventory are also stored physically. This project intends to automate all workorders and have then be submitted and archived electronically. As well, the system is intended to track all scopes of projects, ranging from small miscellaneous tasks to larger scale projects in such a fashion that the workshop manager can schedule them effectively in advance.

1.3 Scope

ERP is designed as a Web API, such that it is run in browser and is able to be accessed from any computer with a sufficient internet connection. It will be a local application that will be primarily accessed by the workshop manager, who is this project's main client. Secondary clients include faculty and staff that wish to submit workorders over the ERP suite. The primary client is the only one intended to have full control of all features of the ERP suite.

The ERP Suite currently is planned to be exclusive to the engineering workshop based on its design as of the completion of this capstone project, as future work on this project will require a redesign to be re-purposed for future clients. The ideal future client for this program is for machine and workshop owners with a staff less than 50.

2 Design Overview

A general overview of the ERP application, and how those features interact with each other during use.

2.1 General Overview

The ERP is designed as a web application that the workshop manager, the main client of the program, can access the workshop's system from any computer using a web browser. The program in a web API, which connects to a remote database containing the workshop's data. The databases are hosted locally at the University of Regina, such that when used in the shop, information can accessed quickly and efficiently without having to deal with external server connections.

When a student or faculty member submits a workorder, it appears as a new item in the workorders page on the workshop manager's window. Each newly submitted workorder will be flagged as a new item, visually showing its status and that it requires attention. When a workorder is added to the system's calendar via the time tracking option, the system will flag its status to say that it is in progress, with the current estimated completion date included. Workorders that pass the current deadline will be flagged as being behind schedule. As well, the workshop manager can also flag any specific workorder as incomplete or cancelled on any workorders that have not been fully completed due to any particular restraints.

In the time tracking section, the workshop manager has access to all currently submitted time entries, showcased in a calendar. Further display options are available such that the user can view the entries in a daily, weekly or monthly format. The entries are first split into two main categories, billable and non-billable time entries. These entries showcase what time was spent working on workorder projects for different clients, and what was spent doing other activities, such as small and/or miscellaneous fixes and builds. Each section is further broken down, with billable time showcasing exactly which project was worked on for how long. With this information, the workshop manager can then bill the student or faculty member for the right amount of work.

For the inventory section, the workshop manager has access to all of the materials and their respective data that is currently stored in the inventory table. Each entry has its own identifier, the price of which the material was last purchased for, information regarding the vendor who sells the product and any other relevant information.

2.2 Assumptions and Constraints

For the web API to function, a connection to the internet and a modern browser to run the application. As is designed currently, the software is to be primarily used for the University of Regina, as the program has access to the university's financial services. Currently, the ERP suite is not designed for mobile use and should be used mostly on a computer.

3 System Architecture

This sections outlines the ERP suite and the program's design.

3.1 Logical View

The project is split into two main categories for development; the frontend and the backend. The following diagram showcases the details of the interface and their following subcategories.

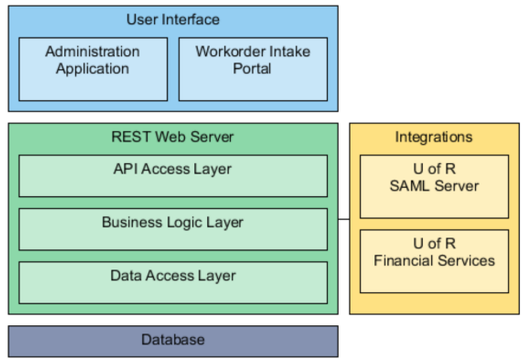

Figure 1: Categories of the ERP Suite

On the front end, the client, who has the most control over the entire application. When the client log into the system upon start up, they have access to the main page displaying all necessary data for the particular day. In progress workorders, upcoming meetings and other related data can be quickly viewed. Clicking on a particular issue will navigate to a more in detailed view of the project. On the left side bar, the client can then access each particular section.

The first section is the Workorders page, where all workorders stored in the database can be accessed. Search and filtering options are available for the client when trying to locate specifics. Each workorder, displayed in a table with all necessary information, can be viewed, edited or deleted. When viewed in a closer look, tat particular workorder page is loaded, displaying all relevant information, including reference id, materials currently associated in the order, and all dates that apply.

The second section accessible by the left navigational bar is the time tracking feature, where the workshop manager can access the full calendar view of every time entry into the system, whether it is billable and non-billable hours.

3.1.1 REST API

The server for the application was designed as a REST API. In the most basic terms, this means the server is stateless and represents the database through resources that are available for the client to access via URIs (universal resource identifiers). There are other aspects of being truly RESTful, but these are the ones of the most concern for this project.

To be stateless implies that the client holds all of the session information and is required to send all information needed for authentication and authorization in every request to the server. What this means in this case is that, on the first request to the server, the client would hit a login endpoint with credentials, receiving back a token that verifies its identity and credentials. This token contains all of the information needed to identify a client and identify it's authorization level. When that client makes another request for a resource, it sends that bearer token with the request, which the server can use to determine if that client has the appropriate permissions to access the requested resource. The really important part of being RESTful, however, is the representation of the system through resources. There are multiple levels to being RESTful with regards to this requirement and they are represented by the Richardson Maturity Model seen below:

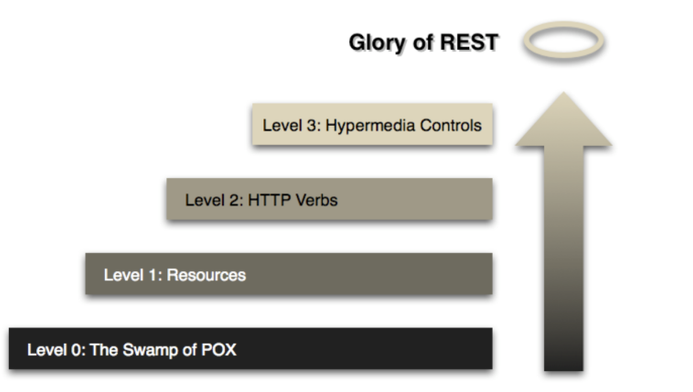

Figure 2: Steps toward Rest Scoure: https://martinfowler.com/articles/richardsonMaturityModel.html

The first level is achieved when the server represents the system as resources available via URIs. This is essentially the base level of being RESTful. The second level, the level with which this server is currently, means the resources and URIs can be properly manipulated through the use of HTTP verbs such as GET, POST, PUT, PATCH, and DELETE. This is referring to the fact that the meaning of a request should be implied through which verb the request is made with. For example, a POST request means a new resource or representation is being created at whatever URI (or endpoint) is hit, whereas a PUT means the resource located at the URI is being replaced with an updated version. The final level of this model is the representation of the system's and resources' available state changes through links generated by the server. This is generally referred to as HATEOAS (hypermedia as the engine of application state). The server is currently in the works of being converted to this level through the use of a response standard known as HAL (hypermedia application language) which attaches resources, embedded resources, and links to any given response. The point of this is that the client shouldn't have to have any background knowledge on how the server operates, the server should present it's possible actions to the client.

3.2 Hardware Architecture

Currently the application is a centralized system, with the client's operating machine and the server hosting the workshop's databases to be in the same location, thus allowing for the system to have a secure connection based on the university's internet. The type of server hosting the backend of the project is a REST web server, allowing the program to operate in a stateless environment, allowing the freedom of the application to transfer data to and from the database. This server is constructed using ASP.NET and entity framework, primarily written in the C# language.

3.3 Software Architecture

The three main pages are divided into their own sections, with each having their own specific connection to the back end, however, each page has a connection with each other as data is shared throughout the application. In total, each section requests and portrays the same data, but it is referenced in different ways. The following diagram showcases how each data table is related in the database.

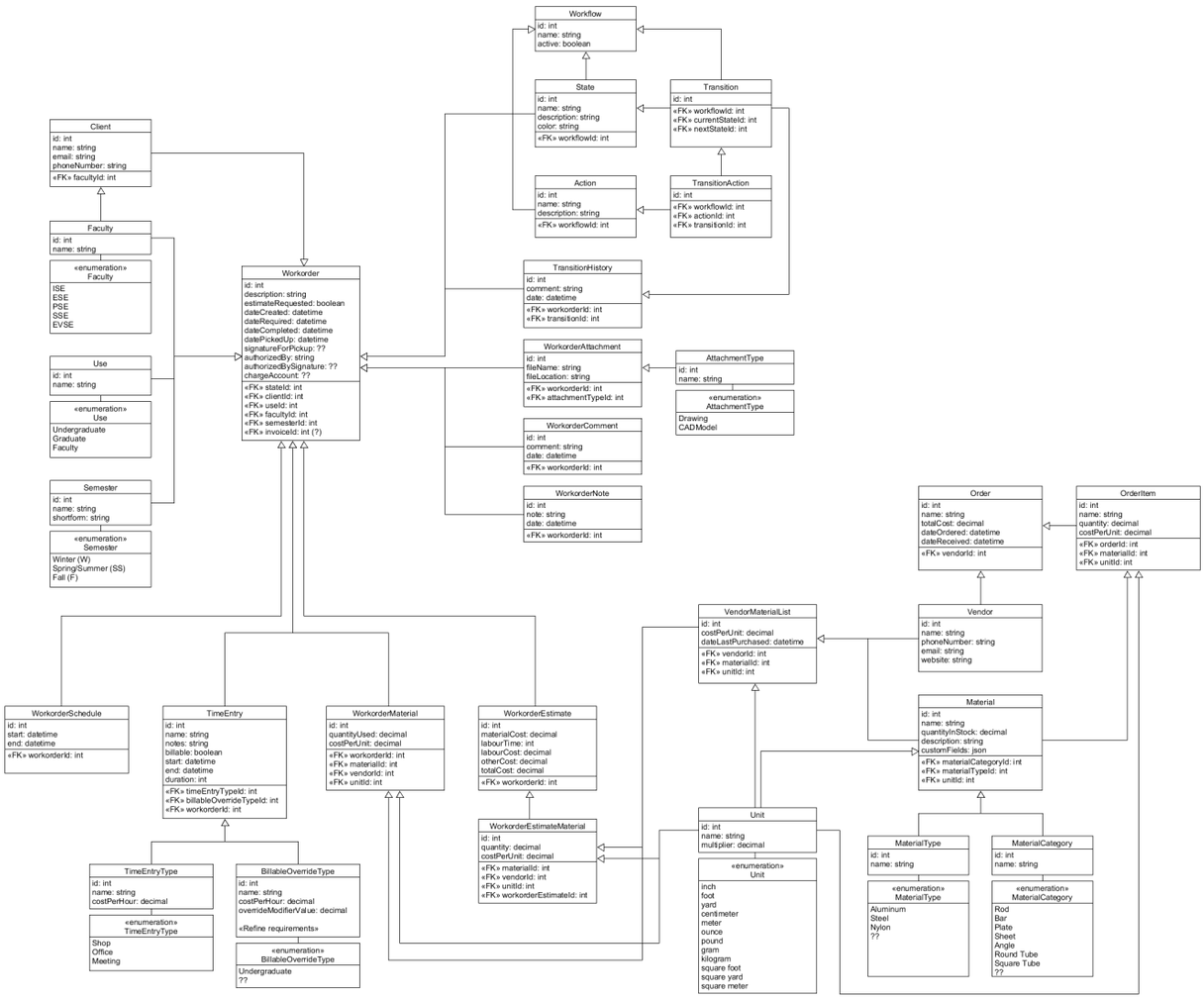

Figure 3: Full class diagram of the ERP

Since each page accesses the database in different ways, data is requested for and received in different fashions. In addition, the status of the workorder has its own data structure, symbolizing the different states that a potential workorder undergoes during the production. The workflow showcases how each different state of a workorder.

3.3.1 Workorders

On the workorders page, workorders are stored on the data are stored in the workorder table. Each entry includes the client, faculty, use, and semester information, as well as the dates of when the request was created and when the workorder is required to be finished, based on a workorder submission. The workorder table also stores the information associated to the workorder process, containing fields for statuses of a workorder, comments on the workorder, and any other additional attachments. The following figure showcases these relationships.

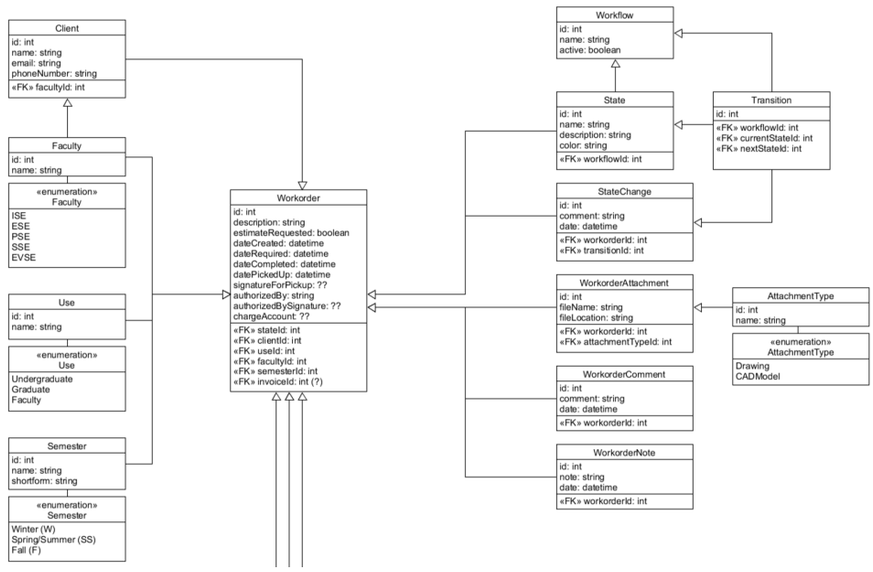

Figure 4: Workorker Class Diagram

3.3.2 Workflow

The workflow section details each state that a workorder goes through from submission to competition. Each state showcases exactly at what point of the process the workorder is at and is highlighted with the appropriate colour for easier understanding. Each state can transition to another at any point, either by the workshop manager or automatically when a certain deadline is passed. The following figure showcases the possible states that a workorder can have during the process.

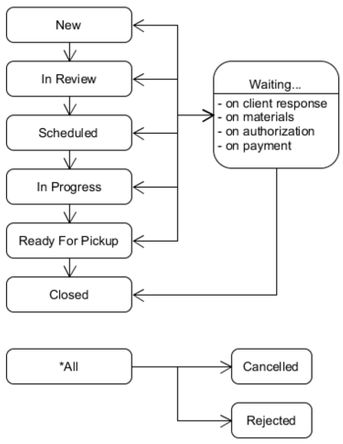

Figure 5: The flow of a Workorder

Each state uses a transition period which highlights the change of the state from one to another. The actual action of moving from one state to another is stored, with the information as to why the change had occurred and when it was done. The following diagram showcases the relationships of these data points.

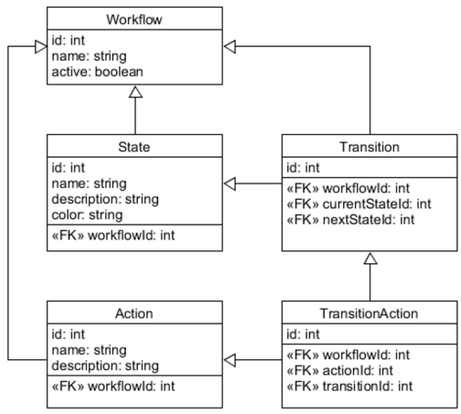

Figure 6: The workflow class diagram

3.3.3 Project Management

For the project management system, each time entry involves having a specific type and the ability to insert billable time to an entry. Each Entry can then be considered an Event, which all those involved in the entry can be notified. The following class diagram showcases these relationships.

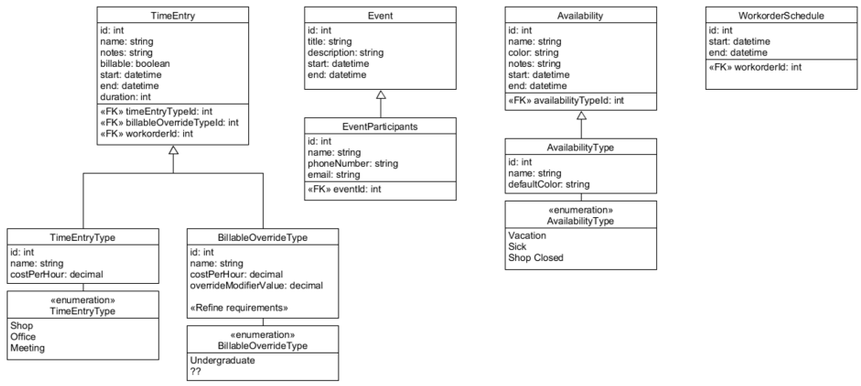

Figure 7: Project Management Class Diagram

3.3.4 Inventory

The following class diagram showcases the inventory tables, and what goes into each particular material upon entry into the system. Every material has a type and falls into a particular category. There are a certain amount of it, stored using the appropriate unit. Each material is sold from a vendor, and each material is ordered from these specific vendors in specific orders.

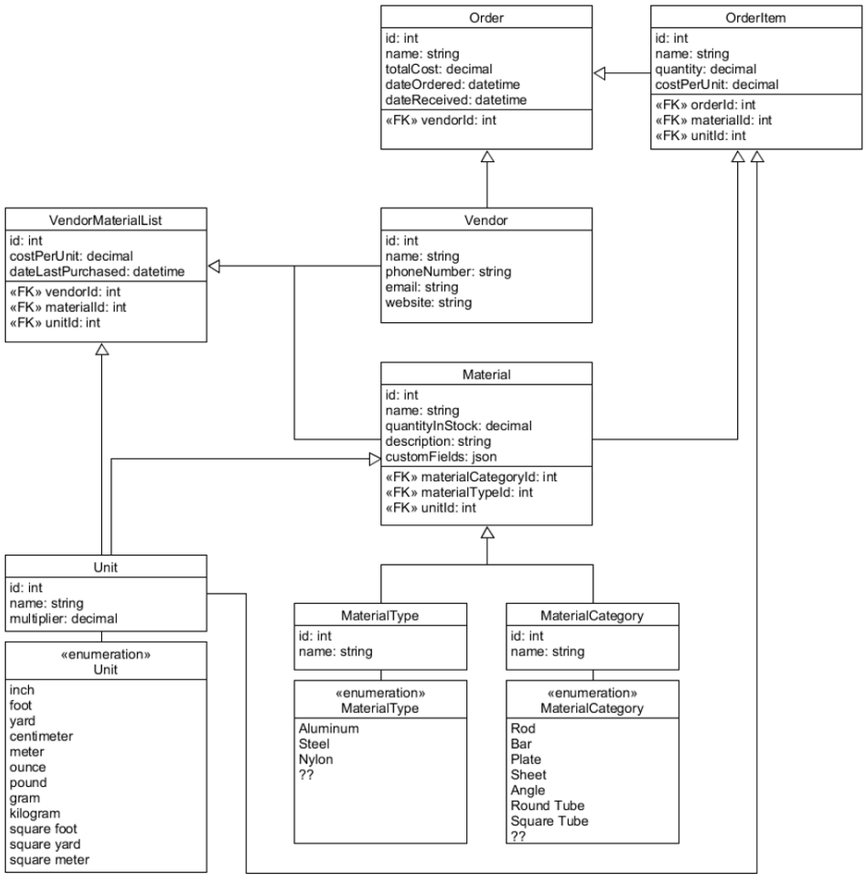

Figure 8: Inventory Class Diagram

4 System Design

While creating the ERP suite, the following protocols and design patterns were used in order to perfect the web application such that is was effective in its functionality and comfortable to for the workshop manager to use.

4.1 Frontend Design

The frontend of the program was created using a javascript framework called Vue.js, which is designed for web APIs and asynchronous CRUD functionality. While other similar frontend frameworks such as Razor Pages, which is the option provided by ASP.NET, and Angular, Google's own javascript framework, can provide the functionality that would be needed for the operation of the ERP suite, the decision to ultimately to use Vue was because of its extensive documentation and popularity in the web development world. It was the framework that had the most documented features available for use, allowing for more options and features be possible to implement inside the program. A typical Vue file has three distinct sections for the HTML code, styling and scripting, meaning that all of the features for a particular page is developed on a single page without having to switch in between files. This coupled with the fact that since Vue can run using a node packaging manager, it is able to be developed asynchronously in a development sever, are the reasonys as to why this framework was the primary choice for the ERP suite. As well, Vue allows for more personalization, without having to be locked into a template designed by a framework.

4.1.1 UI Standards

In the ERP suite, there are a set of user interface standards that are followed for consistency throughout the frontend design.

4.1.1.1 Fonts Throughout the entire program, consistency in the font choice is crucial as to simplify the design of the program. The headings of the program as well as any standout text appearing on the site, such as in the side navigational bars on either side, use the Montserrat font. For the actual content of the program, the Roboto font was used.

4.1.1.2 Palette In its current status, the ERP suite uses a blue-gray colour palette was used. While other, brighter colours are introduced throughout the application, these are primarily used for notification purposes, such as defining the status of workorders or reminders regarding deadlines approaching. To contrast these vibrant colours, a softer and calmer colour palette was needed such that notifications can still pop out for the user, signifying the importance of the task linked to them. The following figure showcases the particular palette used for the ERP suite. Attached are the hexadecimal numbers for each colour in the palette.

Figure 9: Colour palette used for the ERP suite

4.2 Backend Design

For the ERP suite, the backend aspect of the application is built using the ASP.NET web API framework. It uses the C# programming language, and uses the ASP.NET packages and libraries allows for asynchronous access to the application's database as well as a connection to the frontend. Because of the nature of ASP.NET, the frontend can send requests during any time during runtime and can be a truly dynamic system.

Several other options were considered during the planning and design phases. other options for server side programming such as PHP, a completely server side language and Node.js, which is a javascript framework that allows for the use of the language as a server side scripting. However, both ultimately were not applicable for the purposes of this program. The main advantage of using ASP.NET over these options is is the use of the C# language, and the access to the framework's libraries which are used throughout the program.

The backend is broken up into 5 separate projects, which each providing a certain function which combine during runtime. The 5 different projects are API, Common, Models, Repositiories, and Services.

Currently, the backend architecture is broken into two primary layers, the API layer and the repository layer, with the communication between them utilizing the exact models (otherwise seen as the exact representation of the database model). This puts the business logic in the API layer with the repository layer being the layer in which the database is constructed and Entity Framework operates.

The future plan for the backend is to pull the business logic out of the API layer and into a separate services layer that separates the repository from the API. This way, the representation of the data can be tailored for exactly what any given endpoint needs through the use of DTOs (data transfer objects). These are DTOs are received from the client at the API layer, then sent through to the associated service for that endpoint. Here, the DTO is mapped to the actual representation of the data and operated on before being mapped back to the response DTO and sent back to the API and then the client.

4.2.1 API

In the API project, the start up and controller code is located. The start up code is necessary for each web API to run, creating the access each library that will be used throughout the program. This program also configures the different database contexts, a feature of Entity Framework Core that enables the system to create a connection to the database in an abstract fashion. As well, connection with the frontend web pages are established.

The way that ASP.NET is designed, controllers are used to create the endpoints which will send the necessary data to the frontend. Each endpoint is created using the CRUD methodology, using the GET, POST, PUT and DELETE requests. Each request states its type of message, and send a response back to the frontend, depending on the type of request made. Data that is requested by the frontend will be sent if the request is valid and if the particular entry exists. Before any requests are made by the user, they musted be authorized by the account controller. The system will return a 403 error if non-authorized requests are made.

4.2.2 Common and Services

The common and services projects of the backend are more passive in nature, as these two sections host the dependencies and the connections to the libraries that is needed for an ASP.NET API to function properly. These two projects are meant as read only and do not see any actual changes during the development of the ERP suite.

4.2.3 Models

The models project hosts the structure of the each data table that populates the database. Each data table is created exists as a C# class, which each data variable publicly stored. Each varible exists as its own entry in a SQL table. When a class inherits from another, a variable of that particular class is created. Each model is organized to its own folder dependent on its particular section of the ERP suite. Actual data is not stored in these variables, but the database tables will be created based on their structure and relationships with one another.

4.2.4 Repositories

This is the project that hosts the contexts which upon building all the projects, will create the database based on how the context maps out the database. Each table that needs to be created will have its own database set, which the controller then references to when accessing the database. These database set variables a different name then that of the models, such that when these are referenced, the actual database tables are accessed and not the models in the model project. As well, any data that must manually loaded into a particular database, such as the workorder state table as it hosts a description based on the purpose of this state, will written here, thus appearing in the table when the database is created. The method uses is called OnModelCreating, and each particular data table with required data is created as a new entry, such that they will be written into the table upon runtime. Upon the creation of the database, an inital migration must be run such that the backend knows what kind of a database is needed and in what context. By giving the context created in this section, the database will then reflect said context.

4.3 Database Design

The main aspect for communicating with the system's database is Entity Framework Core. It allows the backend, which is written using ASP.NET, to work with and send requests to the database using .NET objects instead of the standard MySQL database language. This automates the database requests without having to write the SQL code for every request. Since the web API is designed in an asynchronous use, creating this layer of abstraction allows for the program run multiple requests with less actual scripting.

In its current design, the database will be hosted on the machine of the workshop manager located in the workshop, with the service connecting to the university internet when the application is running. Since the database does not need to store massive amounts of data, a larger server on university campus or a non-local server is not required but the application is open to these options in future implementations.