



Department of Computer Engineering  
Faculty of Engineering  
University of Peradeniya

# SMART INVENTORY MANAGEMENT SYSTEM -ITEM LOCATOR- **Project Report**

---

Group 13

E/18/098 Fernando K.A.I.

E/18/100 Fernando K.N.A.

E/18/155 Jayasundara J.W.K.R.B.

# SUMMARY

The aim of this document is to provide an overview of the entire completed project for the short semester co227 project. It will cover an area that will explain some of the major problems encountered during the project implementation and how these problems were overcome as well as detailed technical sections.

The Makerspace Lab of Computer engineering of the University of Peradeniya has many items and equipment that are needed for projects done at the lab. As the item count grows it will be hard to keep track of them. Hence, the project [smart inventory management system](#) was proposed to solve this problem. The goal of our project is to add a location service system to the existing project. The application has a lot of added features that will be talked about in more detail in this document. The application is a web based tool that has been implemented using the following technologies.

- PHP
- Laravel
- HTML
- Bootstrap 4
- Vue.js
- Laravel Livewire
- MySQL
- jQuery
- GitHub
- Git

The objective of this project is to improve the productivity in the Makerspace lab. It is achieved by implementing a search/location feature to the existing project which was done by E17 students.

# TABLE OF CONTENTS

<b>SUMMARY</b>	<b>1</b>
<b>TABLE OF CONTENTS</b>	<b>2</b>
<b>LIST OF FIGURES</b>	<b>3</b>
<b>INTRODUCTION</b>	<b>4</b>
Background information	4
The Purpose/Objectives of the Project	5
The Scope of the Work	5
<b>REQUIREMENTS</b>	<b>6</b>
User Requirements	6
Operational, Environmental and Other Requirements	8
<b>DESIGN AND DEVELOPMENT</b>	<b>9</b>
Process and Methodology	9
Technology Stack	10
Architecture	13
User Interaction and Design	14
<b>TESTING</b>	<b>17</b>
<b>PROJECT ISSUES / CHALLENGES / RISKS</b>	<b>22</b>
<b>IDEAS FOR SOLUTIONS/EXTENSIONS</b>	<b>27</b>
<b>CONCLUSION</b>	<b>28</b>
<b>REFERENCES</b>	<b>29</b>

## LIST OF FIGURES

<b>Figure</b>	<b>Page</b>
Figure 1 - Layout before upgrading to livewire tables	15
Figure 2 - Layout after upgrading to livewire tables	15
Figure 3 - Homepage	16
Figure 4 - Menu to change locations of item	23
Figure 5 - Error page	25
Figure 6 - Wrong definition of the relationship	26
Figure 7 - Correct definition of the relationship	26

# INTRODUCTION

## Background information

The number of makerspace lab equipments and components in the department has grown significantly over the last few years. Because of that managing them properly with manual documentation had become quite difficult. Allocating time slots for students to use stations, waiting for lecturers to approve them has become time consuming for both lecturers and students. Usually Technical Officers in the lab are responsible for managing the makerspace lab. They have to make sure the time slot allocation will not overlap and manage time allocations for all the students. For students to allocate the time slots they have to visit the lab 3,4 days prior to the scheduling and make an appointment. This was also a time consuming task for students as well. When students are in the lab and looking for tools and equipment, they ask for help from technical officers if they can't find them by themselves. So if the technical officers are not available it is difficult and confusing to find necessary items.

Because of all those reasons a smart solution to manage the inventory, manage time allocation and digitizing documentation on equipments had become a real need for the department.

As a solution for this SMART INVENTORY MANAGEMENT SYSTEM for the makerspace lab of the department of computer engineering was suggested.

In this short semester project, as a team we were tasked with adding the location services for this ongoing project.

## **The Purpose/Objectives of the Project**

The purpose of this project was to add location services for this existing project. We were tasked with adding searching methods for a user to find an equipment or a component and get information about where it is located in the makerspace lab. While that was the main goal of this project, we were tasked with implementing some other features too. They are,

- 1) A complete system to manage Consumables (Resistors, Capacitors etc) in the system
- 2) Change all the tables in the system into [rappasoft /laravel-livewire tables](#)

Some equipments and components have multiple locations. Therefore the system must support adding multiple locations for a single item. It must also support displaying the previously added multiple locations in the frontend and backend of the system accordingly.

## **The Scope of the Work**

The scope of the work included adding the search function to search for items and add the locations they are located in the makerspace lab. In addition to adding relevant database tables, relevant data needed to be added. For each item support for having multiple locations in the database was required. Implementing all the frontend views that was required for the item locator was in the scope of our tasks. Implementing all the backend controllers that is required for item locator was also in the scope. As the project was going on the decision was made to change the tables in the web application to livewire tables that will easily enable updating web pages without reloading the whole page. Adding the necessary data for a future AR implementation was also part of the project scope.

# REQUIREMENTS

## User Requirements

In this system there are several types of users. Each user has slightly different requirements. Down below is a list of types of users in the Smart Inventory Management System.

- Admin

Admin needs to be able to change and delete all the data in the system. That includes adding, modifying or deleting equipment, components and stations and all the data such as name, picture, amount of the stocks left, location of the item that belongs to those categories. Adding new user types, modifying access levels of those user types is also a requirement for the admin.

- User

Users need to be able to search for items they are looking for in the makerspace lab. By typing the name of the item they have to get a page with the information about the item including the location of the item it is situated in the lab. Users also must be able to send fabrication requests to the lecturers for them to accept. That is for users to use printers in the lab. Another main requirement for users is to schedule a lab and a work desk for them to use.

- Lecturer

Lecturers' main requirement is to accept fabrication requests from students. They can accept or reject the requests depending on the situation. Apart from that main requirement lectures must also be able to get information on the equipment and components. Location service is also available for lecturers.

- Technical Officer

Technical officers should be able to approve and maintain the makerspace lab time and work desks allocation. Technical officers can update the item information such as the number of items left in consumables, adding new equipment, consumables to the system, and changing locations of items where it is located in the makerspace lab.

- Maintainer

Maintainers can maintain stock numbers of consumables and update data in the items' respective pages.



## **Operational, Environmental and Other Requirements**

The Smart Inventory Management System is expected to be hosted in a department server. So the web application should be able to be hosted in a server without having issues. The system is required to have adequate cyber security measures and have the capacity to serve the requests from the users in the computer engineering makerspace lab. That means the data in the database must be protected against attacks and the scalability for a growing number of users. The system web application should be fast and responsive regardless of some pages having heavy content and a search function that searches through all the equipment and consumables.

Since the system databases contain very dynamic data, meaning that the data is regularly updated by Technical officers and Maintenance users, that process should be easy for technical officers to use.

The system should be easy to understand and easy to use even for a new user. New batches that will be coming to the department may not have a clear understanding of the layout of the lab, equipment in it as well as how the web application works, the system should be easily understandable by using it for a few minutes.

As this is a web application, users of the system can use any device of their liking if the device can load the webpage. There is no requirement to install any softwares to his/her device.

# DESIGN AND DEVELOPMENT

## Process and Methodology

We decided to use agile methodology as the methodology to follow in the process of completing the project. First the tasks to be done were divided into sprints and each sprint was added to github as an issue with tasks (with the prefix [Group13] as there were 3 groups working on the same project). Those tasks were divided among our team members and they were completed within a week. At the end of completion we took a sprint review meeting with our project owner and our scrum master.

Those feedback helped us to do modifications to our system accordingly and because of that we were able to make changes without having to wait until the project was over and then do changes.

Using agile methodology and breaking tasks with sprints helped us to identify bugs along the way and fix them. If it weren't for the sprints there could have been more bugs and no way to identify them until a user finds a bug and reports it.

Another benefit was changing the user requirements and the task being done with the results of each sprints. That was done several times and was possible because of the agile methodology.

As a team we were using group chats and regular virtual meetings to keep up with tasks and to communicate the progress of tasks that were done.

## Technology Stack

- PHP Laravel

PHP Laravel was chosen for this project because it's a full stack framework. We wouldn't have to develop the backend and frontend as separate parts. With Laravel we could build them together. Laravel framework internally forces us to use the MVC architecture which keeps the code clean and readable for everybody. As Laravel is a full stack framework it has advanced features and development tools that facilitate rapid web application development that other frameworks don't have. Some say that the Laravel framework is quite old now. But Laravel suits this project very well as the scale of this project is in the level of about 500 users at any given time. Laravel also has built in Authentication and Authorization. It made the developing process much faster and easier. The Eloquent ORM is Laravel's built-in ORM implementation. Laravel has the best Object-relational Mapper as compared to the other frameworks out there. This Object-relational mapping allows you to interact with your database objects and database relationships using expressive syntax. All these reasons were why E17 students decided to choose Laravel for our project.

- MySQL

Laravel has built in support for MySQL. We can interact with MySQL using Laravel's Eloquent ORM which makes things easier for the development process. Even Though we used MySQL for our projects' database, we can change the database anytime we want because laravel supports that. Currently, Laravel supports MySQL, Postgres, SQLite, and SQL Server out of the box. Therefore, in the future if we want we can change the underlying database without much issue.

- SQLite

During the development of the system we used SQLite instead of MySQL because it was lightweight and easier to set up on our local machines.

- HTML + Bootstrap 4 / 5

Bootstrap was used as it was a free and beginner friendly front end framework. It also includes HTML and CSS-based design templates for forms, typography, buttons, navigation, tables, modals, image carousels, and many other components along with other optional JavaScript plugins. Later in the development, Bootstrap 4 was upgraded to Bootstrap 5 as we needed some new elements that were in Bootstrap 5.

- Livewire

Livewire is a full-stack framework for Laravel that makes building dynamic interfaces simple, without leaving the comfort of Laravel. In basic terms, We don't have to write javascript code in the frontend to send requests to the backend and update the frontend view without reloading the whole page. (Frontend sending the requests to the backend part is handled by the livewire framework ) This framework made it much easier to create the dynamic web pages required in our project. This was used in the location toggler menu. It sends the data back to the server as soon as the user ticks the checkbox in that page. Livewire was also used in the tables for searching, sorting and filtering.

- Github and git for version control

Github was used as we were already familiar with it (Instead of BitBucket) and we had a free pro version account provided to us by the University. Features like GitHub Actions were used to run the

PHPUnit tests on each commit to make sure that already implemented features were not broken in new commits.

- Some other technologies were used by the other groups such as jQuery..

## Architecture

When the web application is loaded the entry point for all the requests is the index.php file. It loads the rest of the Laravel framework and starts the application. After that when a user clicks on a link, it creates a HTTP request and it is being sent to the server. In response to that, the server generates a response and sends it to the user.

When a user clicks on a link it will follow the route it has been set to. That route will be directed to some targeted middleware that will check for different authentication methods. Middleware provides a convenient mechanism for filtering or examining HTTP requests entering your application. For example, Laravel includes a middleware that verifies if the user of your application is authenticated. If the user is not authenticated, the middleware will redirect the user to the login screen. However, if the user is authenticated, the middleware will allow the request to proceed further into the application.

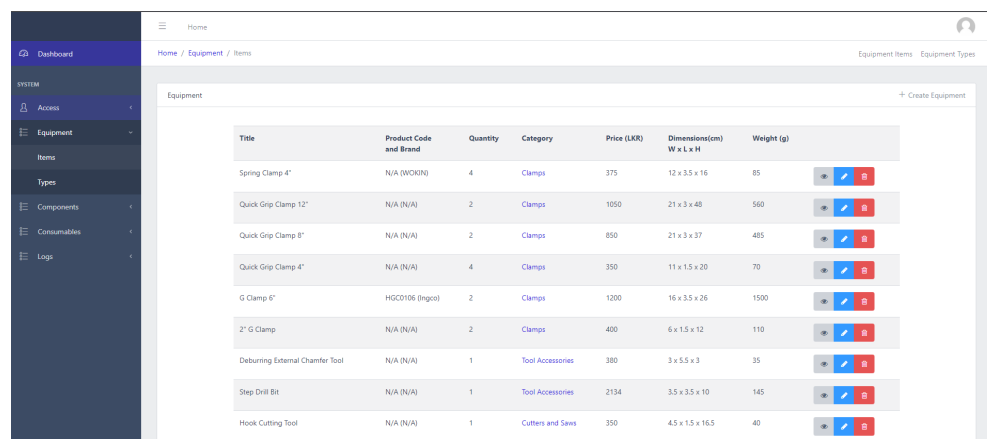
After the middleware the request is going to a controller class and there a response is generated. The response will travel back outward through the route's middleware, giving the application a chance to modify or examine the outgoing response.

The underlying architecture for the web application is pretty much set by the laravel framework and as the developers we had to create routes for all the pages, create controller classes for the activities, create models for them and add frontend web pages using blade syntax.

## User Interaction and Design

As design decisions we agreed to change all the tables that contain data of consumables, equipment, components and stations we updated the tables to laravel-livewire tables for an easy search. Down below is the layout of tables without livewire.

Figure 1 is the layout of the tables before livewire was added

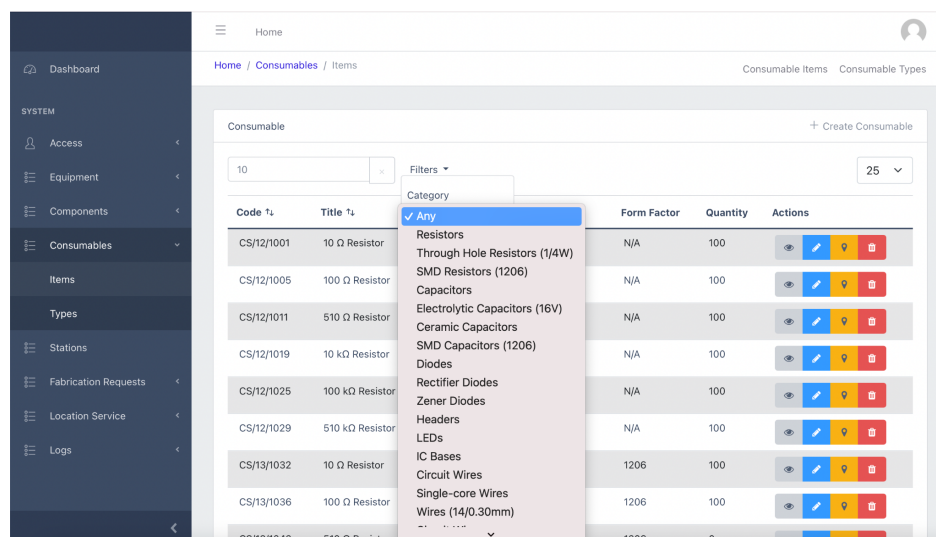


The screenshot shows a web application interface with a sidebar menu on the left containing 'Dashboard', 'Access', 'Equipment', 'Items', 'Types', 'Components', 'Consumables', and 'Logs'. The main content area is titled 'Equipment' and includes a '+ Create Equipment' button. It displays a table with the following columns: Title, Product Code and Brand, Quantity, Category, Price (LKR), Dimensions(cm) W x L x H, and Weight (g). The table lists various equipment items such as Spring Clamps, Quick Grip Clamps, G Clamps, and Deburring External Chamfer Tools.

Title	Product Code and Brand	Quantity	Category	Price (LKR)	Dimensions(cm) W x L x H	Weight (g)
Spring Clamp 4"	N/A (WORKING)	4	Clamps	375	12 x 3.5 x 16	85
Quick Grip Clamp 12"	N/A (N/A)	2	Clamps	1050	21 x 3 x 48	560
Quick Grip Clamp 8"	N/A (N/A)	2	Clamps	850	21 x 3 x 37	485
Quick Grip Clamp 4"	N/A (N/A)	4	Clamps	350	11 x 1.5 x 20	70
G Clamp 6"	HGCD106 (Ingco)	2	Clamps	1200	16 x 3.5 x 26	1500
2" G Clamp	N/A (N/A)	2	Clamps	400	6 x 1.5 x 12	110
Deburring External Chamfer Tool	N/A (N/A)	1	Tool Accessories	380	3 x 5.5 x 3	35
Step Drill Bit	N/A (N/A)	1	Tool Accessories	2134	3.5 x 3.5 x 10	145
Hook Cutting Tool	N/A (N/A)	1	Cutters and Saws	350	4.5 x 1.5 x 16.5	40

Figure 1 - Layout before upgrading to livewire tables

Figure 2 is the layout of tables after livewire was added to tables.



The screenshot shows the 'Consumables' table layout after the livewire upgrade. It features a sidebar menu with 'Dashboard', 'Access', 'Equipment', 'Items', 'Consumables', 'Types', 'Stations', 'Fabrication Requests', 'Location Service', and 'Logs'. The main content area is titled 'Consumable' and includes a '+ Create Consumable' button. A search bar with the value '10' and a 'Filters' dropdown are at the top. The table has columns for Code, Title, Form Factor, Quantity, and Actions. A category dropdown menu is open, showing options like 'Any', 'Resistors', 'Capacitors', 'Diodes', etc. The table lists various consumable items such as resistors and capacitors.

Code	Title	Form Factor	Quantity	Actions
CS/12/1001	10 Ω Resistor	N/A	100	[Eye] [Pencil] [Eraser] [Trash]
CS/12/1005	100 Ω Resistor	N/A	100	[Eye] [Pencil] [Eraser] [Trash]
CS/12/1011	510 Ω Resistor	N/A	100	[Eye] [Pencil] [Eraser] [Trash]
CS/12/1019	10 kΩ Resistor	N/A	100	[Eye] [Pencil] [Eraser] [Trash]
CS/12/1025	100 kΩ Resistor	N/A	100	[Eye] [Pencil] [Eraser] [Trash]
CS/12/1029	510 kΩ Resistor	N/A	100	[Eye] [Pencil] [Eraser] [Trash]
CS/13/1032	10 Ω Resistor	1206	100	[Eye] [Pencil] [Eraser] [Trash]
CS/13/1036	100 Ω Resistor	1206	100	[Eye] [Pencil] [Eraser] [Trash]
CS/13/1042	510 Ω Resistor	1206	0	[Eye] [Pencil] [Eraser] [Trash]

Figure 2 - Layout after upgrading to livewire tables

After adding laravel-livewire to the tables, search result is shown as the user types the keywords in the search bar (located at the top left of the table). Sorting was also available after the livewire was added to the tables. That makes it very easy for users to go through the search result. Filtering the results from the given drop down menu was also added with the implementation of laravel-livewire.

To improve the user experience and interaction with the system and to get an idea about the makerspace lab and its items, Search by location was added under the Location Services. By using that a new user can get an idea about what is available in specific areas of the makerspace lab. For an inexperienced user this would be really helpful. Below is a screenshot of the Search by location page.

Another design improvement we added was the frontend search which supports the search of makerspace lab items even without logging in to the system as any type of user. A guest user can search and get basic information including the location of the item in the lab from the frontend pages that are dedicated to the frontend. This will help for a quick search to get the availability of and the location of an item. Shown below is the design layout of frontend search.

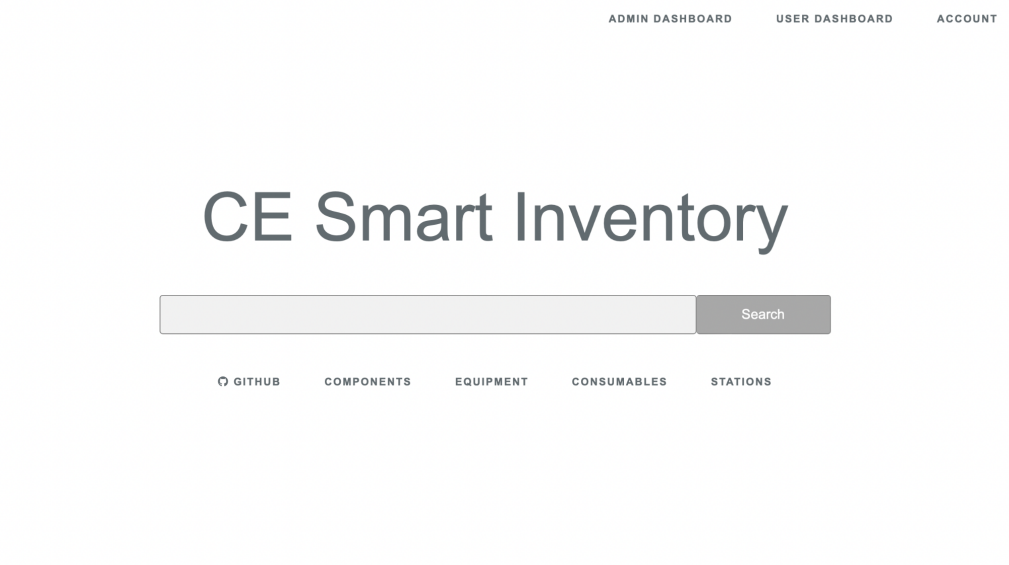


Figure 3 - Homepage



This search also uses the same methodology as the search function under location services. It will search for the keyword under the consumables, equipment, components. For raw materials and machines only the location of the item will be shown in the frontend search result as there is no frontend page for them currently. The search result is also shown in a grid with its image to make it easy for an user to find the item.

# TESTING

Testing was mainly done using the build in testing provided by Laravel. Laravel is built with testing in mind. In fact, support for testing with PHPUnit is included out of the box. Therefore, it was much easier to add tests to this application. We have added tests to every part of the app that we have created. Everything from the simplest things to the most complex things are added to tests to make sure that we don't break any existing functionality as we develop the app.

All the tests can be found at `./tests/Feature` directory. They can be run by the `'php artisan test'` command. Furthermore, the tests can be run much faster by using the `paratest` library. It runs the tests in parallel using the multiple cores available in the system. Parallel tests can be run using the `'php artisan test -p'` command. The default unit tests run in about 50 seconds and the parallel tests take about 18 seconds. That is a speedup of about 2.77 times. We had also created a GitHub Action to run the tests in every new commit to the repository.

The unit tests that were added by us are,

- 1) Consumable Item Tests  
(tests/Feature/Backend/Consumable/ConsumableItemTest.php)
  - a) An admin can access the list consumable page
  - b) An admin can access the create consumable page
  - c) An admin can access the show consumable page
  - d) An admin can access the delete consumable page
  - e) Create consumable requires validation
  - f) Update consumable requires validation
  - g) An consumable can be created
  - h) An consumable can be updated
  - i) Delete consumable item
  - j) Unauthorized user cannot delete consumable item
  - k) Shows single location
  - l) Shows multiple locations

## 2) Consumable Type Tests

(tests/Feature/Backend/Consumable/ConsumableTypeTest.php)

- a) An admin can access the list consumable type page
- b) An admin can access the create consumable type page
- c) An admin can access the show consumable type page
- d) An admin can access the delete consumable type page
- e) Create consumable type requires validation
- f) Update consumable type requires validation
- g) An consumable type can be created
- h) An consumable type can be updated
- i) Delete consumable type item
- j) Unauthorized user cannot delete consumable type item

## 3) Location Service Tests

(tests/Feature/Backend/LocationService/LocationServiceTest.php)

- a) Item locations table exists
- b) Locations table exists
- c) Equipment shows location
- d) Component shows location
- e) Consumables shows location
- f) Machines shows location
- g) Raw materials shows location
- h) Deleting component will remove entry in item location table
- i) Deleting equipment will remove entry in item location table
- j) Deleting consumable will remove entry in item location table
- k) Deleting machine will remove entry in item location table
- l) Deleting raw material will remove entry in item location table

#### 4) Location Menu Tests

(tests/Feature/Backend/LocationService/LocationsMenuTest.php)

- a) Admin can see locations menu
- b) Menu shows all locations
- c) Can create new location
- d) Newly created location shows up on index
- e) Can edit location name
- f) Can delete location parent location

#### 5) Location Toggler Livewire Tests

(tests/Feature/Backend/LocationService/LocationsTogglerLivewireTest.php)

- a) Addlocation function adds equipment item location to db
- b) Addlocation function adds component item location to db
- c) Addlocation function adds consumable item location to db
- d) Addlocation function adds machine location to db
- e) Addlocation function adds raw materials location to db
- f) Addlocation function removes equipment item location from db
- g) Addlocation function removes component item location from db
- h) Addlocation function removes machines location from db
- i) Addlocation function removes raw materials location from db
- j) Addlocation function equipment location change shows up in frontend
- k) Addlocation function component location change shows up in frontend
- l) Addlocation function consumable location change shows up in frontend
- m) Location toggler page has livewire
- n) Location toggler shows all available locations
- o) Creating new raw materials redirects to locations toggler
- p) Creating new machines redirects to locations toggler

6) Normal Search Test

(tests/Feature/Backend/Search/NormalSearchTest.php)

- a) Admin can load search page
- b) Admin can search for items
- c) Will show error when keyword is empty
- d) View has search results
- e) Search results has href
- f) Search results has resistor results
- g) Search results links are working
- h) Search results has 0 results please check spellings text
- i) Newly added equipment is shown in results
- j) Newly added consumable is shown in results
- k) Newly added component is shown in results
- l) Newly added machine is shown in results
- m) Newly added raw material is shown in results

7) Reverse Search Test

(tests/Feature/Backend/Search/ReverseSearchTest.php)

- a) Admin can load reverse search
- b) Admin can search for items
- c) Search results has href
- d) New equipment will appear in reverse search
- e) New component will appear in reverse search
- f) New consumable will appear in reverse search
- g) New machine will appear in reverse search
- h) New raw material will appear in reverse search

These tests were very useful to us while we were developing the application. It prevented us from breaking previous features numerous times. For example, one time. A team member changed the inventory code of the Raw Material items without knowing that the inventory code change would affect the Search feature in the application. The inventory code change affected the search feature resulting in the raw materials not showing up on the search results page. This would have never been caught if the tests were not there. Immediately after the inventory code change was committed to github. A test failed and then we were able to fix the error. There were many other incidents that were similar to this.

## PROJECT ISSUES / CHALLENGES / RISKS

### 1) New to Laravel

All 3 of us never had any experience with the Laravel Framework. We had to learn Laravel from the beginning to complete this project as this was the first web application that we had to complete using PHP Laravel. However, the documentation of the Laravel framework was really helpful in learning the framework. The documentation had clearly explained the important points of the framework.

### 2) New to GitHub and git

We were new to Github and git. The idea of commits and branches were new to us. Mr. Nuwan Jaliyagoda helped us immensely on how to use github and git for this project.

### 3) Laravel Eloquent

We had studied MySQL in CO226, but laravel uses an inbuilt model system called Laravel Eloquent. It's an object relational mapper that's designed to work with SQL databases but the code is based on PHP instead of MySQL. It was a bit confusing at first to find out that there was a system like this in place. But with time we got more comfortable with using the Eloquent model in Laravel and found it to be more user friendly than using SQL directly to do CRUD operations with the database.

### 4) Problems with SQLite and MySQL

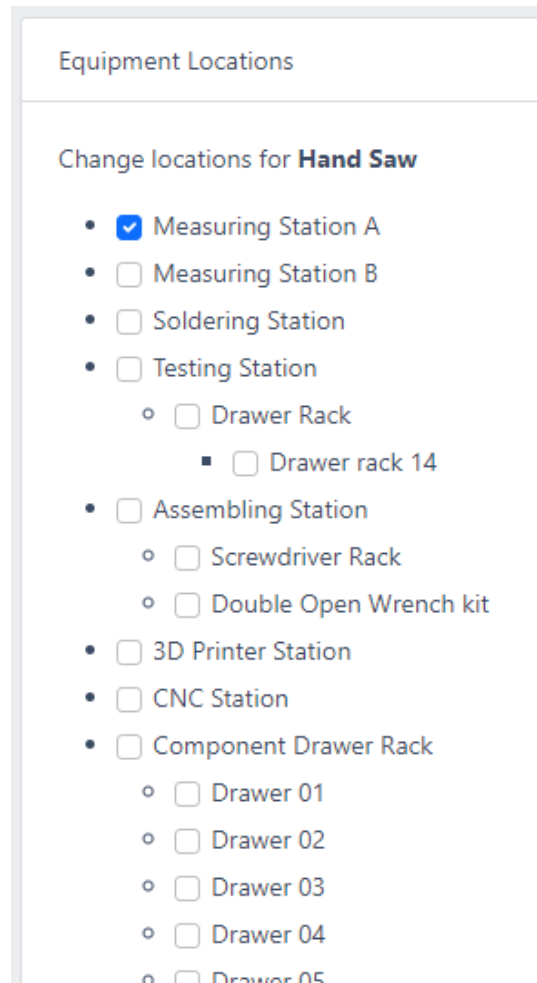
One of our team members used SQLite instead of MySQL in his development environment. There was an issue with the seeders for the database. SQLites initial id number for the records was 0 and MySQL doesn't support the id numbers starting from 0 and needed 1 as the first id number. This was a bit hard to track down. But Mr. Nuwan Jaliyagoda helped us with this issue.

### 5) Issue with composer.lock on Windows and Linux

Composer.lock file was mistakenly added to the .gitignore file so that it would not be tracked by git. Unit tests on our local tests ran fine. But the tests run by github actions failed. It was time consuming to find out that the composer.lock file was the reason for the tests failing. It was solved after we added the composer.lock file back into git.

## 6) Locations Toggler page

As there can be multiple items in the lab. The system needed to support multiple locations. Therefore, the locations feature must be able to add more than one location for a single item to show that there are multiple items in those locations. In the page where we set the locations for the item. We needed the locations to show up in a hierarchical way shown below.



The screenshot shows a web interface titled "Equipment Locations". Below the title is a section "Change locations for **Hand Saw**". This section contains a list of locations, each preceded by a bullet point and a checkbox. The locations are: "Measuring Station A" (checked), "Measuring Station B", "Soldering Station", "Testing Station", "Assembling Station", "3D Printer Station", "CNC Station", and "Component Drawer Rack". The "Testing Station" and "Assembling Station" are expanded, showing sub-locations. "Testing Station" has a sub-location "Drawer Rack", which is further expanded to show "Drawer rack 14". "Assembling Station" has sub-locations "Screwdriver Rack" and "Double Open Wrench kit". The "Component Drawer Rack" is expanded to show a list of drawers: "Drawer 01", "Drawer 02", "Drawer 03", "Drawer 04", and "Drawer 05".

- ☒ Measuring Station A
- ☐ Measuring Station B
- ☐ Soldering Station
- ☐ Testing Station
  - ☐ Drawer Rack
    - ☐ Drawer rack 14
- ☐ Assembling Station
  - ☐ Screwdriver Rack
  - ☐ Double Open Wrench kit
- ☐ 3D Printer Station
- ☐ CNC Station
- ☐ Component Drawer Rack
  - ☐ Drawer 01
  - ☐ Drawer 02
  - ☐ Drawer 03
  - ☐ Drawer 04
  - ☐ Drawer 05

Figure 4 - Menu to change locations of item

As there are multiple steps in the hierarchy, we couldn't use a simple for loop to generate the location hierarchy. We had to use a recursive approach and it was a bit confusing to do. It was made even



more complicated by the fact that livewire was used to create this page.

This is a simple explanation of how this was achieved in the component edit location page. `ComponentItemControllers EditLocation` method retrieves the `Location` records that have the `makerspace lab` as the parent location (It will only retrieve the locations that are in the first level in the hierarchy). It then sends those locations to the edit-location view. The view runs a for loop and generates the livewire component “location- hierarchy- for- edit-location”. The component checks if there are child locations for that particular location and calls the same livewire component if there are child locations. Below are the file paths if you are interested in looking at how it works.

- `app/Http/Controllers/Backend/ComponentItemController.php` `editLocation` method
- `resources/views/backend/component/items/edit-location.blade.php`
- `resources/views/backend/partials/location-hierarchy-for-edit-location.blade.php`
- `resources/views/livewire/locations-toggler.blade.php`
- `app/Http/Livewire/LocationsToggler.php`

## 7) Error that occurred when trying to implement the livewire location toggler.

While developing the livewire location toggler component. The error shown below kept popping up.

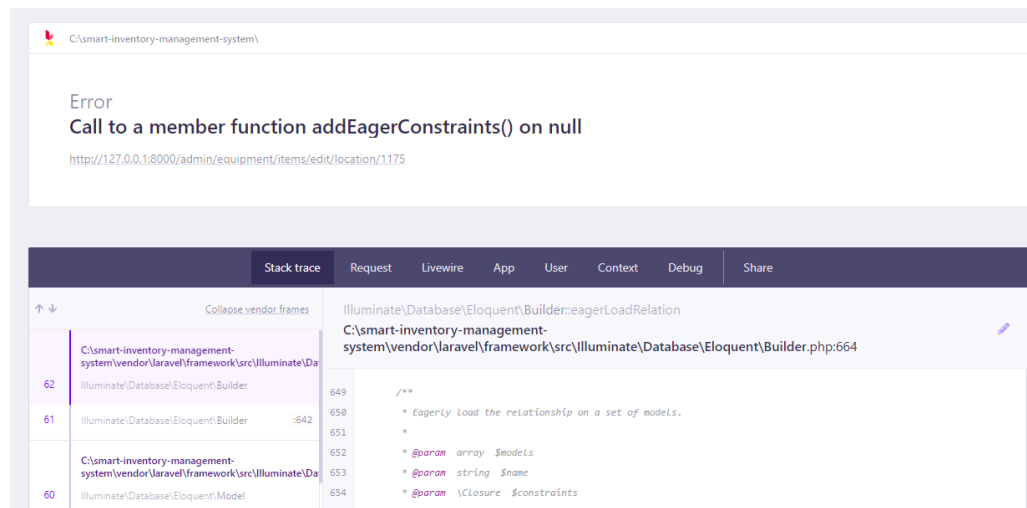


Figure 5 - Error page

The error stack didn't show where the error occurred in an understandable way. Searching for this error on google showed that it usually occurs by a wrong foreign key definition in the model class. Our team got on a zoom call and started to find how to fix this error. We all spent about 4 hours trying to figure out what was wrong. We looked at the ItemLocations class' model and made sure that all the foreign keys were written correctly. But still the error was not fixed. At last we found out that the EquipmentItem class' foreign key for equipment type had a definition of,

```

16 // Link the Equipment Type table
17 public function equipment_type()
18 {
19     if ($this->equipment_type_id != null) return $this->belongsTo
20         ( related: EquipmentType::class, foreignKey: 'equipment_type_id', ownerKey: 'id');
21     return null;
22 }

```

Figure 6 - Wrong definition of the relationship

Which has an if condition and returns nulls in some cases. This was the cause for the previously stated error. We changed it to,

```

16 // Link the Equipment Type table
17 public function equipment_type()
18 {
19     // Do not change.
20     return $this->belongsTo( related: EquipmentType::class, foreignKey: 'equipment_type_id',
21                             ownerKey: 'id');
22 }

```

Figure 7 - Correct definition of the relationship

After that the error was fixed. We spent a great deal of time on solving this issue.

## IDEAS FOR SOLUTIONS/EXTENSIONS

From the beginning of the project we had an AR related helper tool idea for the system as an extension for this project. Specifically we had the idea of adding an AR tool to the web application that will use the camera of the user's smartphone and when pointed at a table it will graphically display details about the tools it is pointed at by the camera.

This task was planned to be done by putting an AR marker by each desk and the system will detect that from the input of the camera. After the AR marker is detected the system knows exactly what desk the user is pointing at and that is used to identify tools. The system is to be fed with the X,Y,Z coordinates of the tools respective to the top of the table where the AR marker is located. By using AR marker location and X,Y,Z coordinates from the database the system will detect and show details of the tool.

The underlying requirements were added to the system by our team. That is we added the database that will hold the X,Y,Z coordinate for the tool. This is also updatable by the web interface under editing the item.

With the use of that and adding AR marker reading libraries the Smart Inventory Management System can be equipped with AR helping for a new user to identify tools.

## CONCLUSION

In this short semester project we completed the project by adding the item locator functionality to the Smart Inventory Management System. By doing so we created necessary databases, migrated the databases with proper data, created relations among the databases, enabled the capability for items to have multiple locations in the database, added search model which will enabled search functionality, created routes for the item locator, created view for them. Apart from the backend search we added a frontend search for guest users as well.

Adding Images for the consumables, equipment and components was done. All the tables in the web application were modified with laravel-livewire for better search and sort functionality.

As a base for the extensibility as an AR solution, we added necessary underlying databases to hold X,Y,Z coordinates of the tools.

For all the tasks we did, tests were added to minimize the possibility of having bugs and issues in the system.

After merging all three groups work for the development branch and testing it there, the system can be hosted in a department server. First it would be best to go for a testing phase with a limited number of users and get user feedback of bugs or issues with the system and after that the system would be able to launch for all the users in the department of computer engineering.

## REFERENCES

- 1) “Laravel - The PHP Framework For Web Artisans.” Laravel - The PHP Framework For Web Artisans, laravel.com, <https://laravel.com/docs/8.x/installation>. Accessed 3 June 2022.
- 2) The PHP Framework For Web Artisans. (n.d.). Laravel. Retrieved June 3, 2022, from <https://laravel.com/docs/9.x/lifecycle>
- 3) The PHP Framework For Web Artisans. (n.d.). Laravel. Retrieved June 3, 2022, from <https://laravel.com/docs/9.x/routing>
- 4) Laravel - Controllers. (n.d.). Tutorialspoint. Retrieved June 7, 2022, from [https://www.tutorialspoint.com/laravel/laravel\\_controllers.htm](https://www.tutorialspoint.com/laravel/laravel_controllers.htm)
- 5) A Guide to Technical Report Writing, Retrieved August 7, 2022, from [https://ias.ieee.org/images/files/CMD/2020/2020-01-16\\_IET\\_Technical\\_Report\\_Writing\\_Guidelines.pdf](https://ias.ieee.org/images/files/CMD/2020/2020-01-16_IET_Technical_Report_Writing_Guidelines.pdf)