

# Visualisation of IoT Data

**Smart and Connected Devices I (ACDA5005)**



**Prifysgol Cymru**  
Y Drindod Dewi Sant  
**University of Wales**  
Trinity Saint David

---

*Siobhan Kelly Stephens – 1800709*

---

## Table of Contents

|                                                           |           |
|-----------------------------------------------------------|-----------|
| <b>1. Key files.....</b>                                  | <b>3</b>  |
| <b>2. Technologies Used .....</b>                         | <b>4</b>  |
| 2.1 PHP Laravel MVC Framework.....                        | 4         |
| 2.2 MySQL Database .....                                  | 6         |
| 2.3 Amazon Web Services (AWS) PHP SDK and S3 Storage..... | 7         |
| 2.4 GitHub and Heroku.....                                | 7         |
| 2.5 Chart.js .....                                        | 8         |
| <b>3. Application Functionality.....</b>                  | <b>9</b>  |
| 3.1 User Authentication and Authorisation .....           | 9         |
| 3.2 Searches Implemented.....                             | 13        |
| 3.3 Other Functionality .....                             | 31        |
| <b>4. Reflection.....</b>                                 | <b>34</b> |
| 4.1 Technologies Used .....                               | 34        |
| 4.2 Working Practices.....                                | 34        |
| 4.3 Searches Implemented.....                             | 34        |
| <b>5. References .....</b>                                | <b>35</b> |
| <b>Appendix .....</b>                                     | <b>36</b> |
| 5.1 <i>Appendix A</i> .....                               | 36        |
| 5.2 <i>Appendix B</i> .....                               | 38        |
| 5.3 <i>Appendix C</i> .....                               | 43        |
| 5.4 <i>Appendix F</i> .....                               | 44        |
| 5.5 <i>Appendix D</i> .....                               | 45        |
| 5.6 <i>Appendix E</i> .....                               | 46        |

## 1. Key files

The following files/folders contains the application code produced by the developer. Other files are produced by the framework.

Important Directory Structure for the Application (files to focus on):

```
|— ... #  
|— app # Contains the `Models`  
|   |— Http # Contains the `Controllers` (main logic)  
|   |— Controllers # `Main app logic`  
|— database  
|   |— migrations # Contains the `database migrations`  
|   |— seeds # Contains the seeders for the database  
|— public  
|   |— css # Contains the css file  
|   |— img # Contains required images  
|   |— js # Contains a single js file  
|— resources  
|   |— views # Contains the `Views`  
|   |— auth # Contains the Auth related views  
|   |— layouts # Contains the individual view content files  
|   |— partials # Contains the partial view files  
|— routes # Contains the `Routing` files  
|   |— web.php # Main routing file  
|— ...
```

*Figure 1. Directory structure of important files within the application*

## 2. Technologies Used

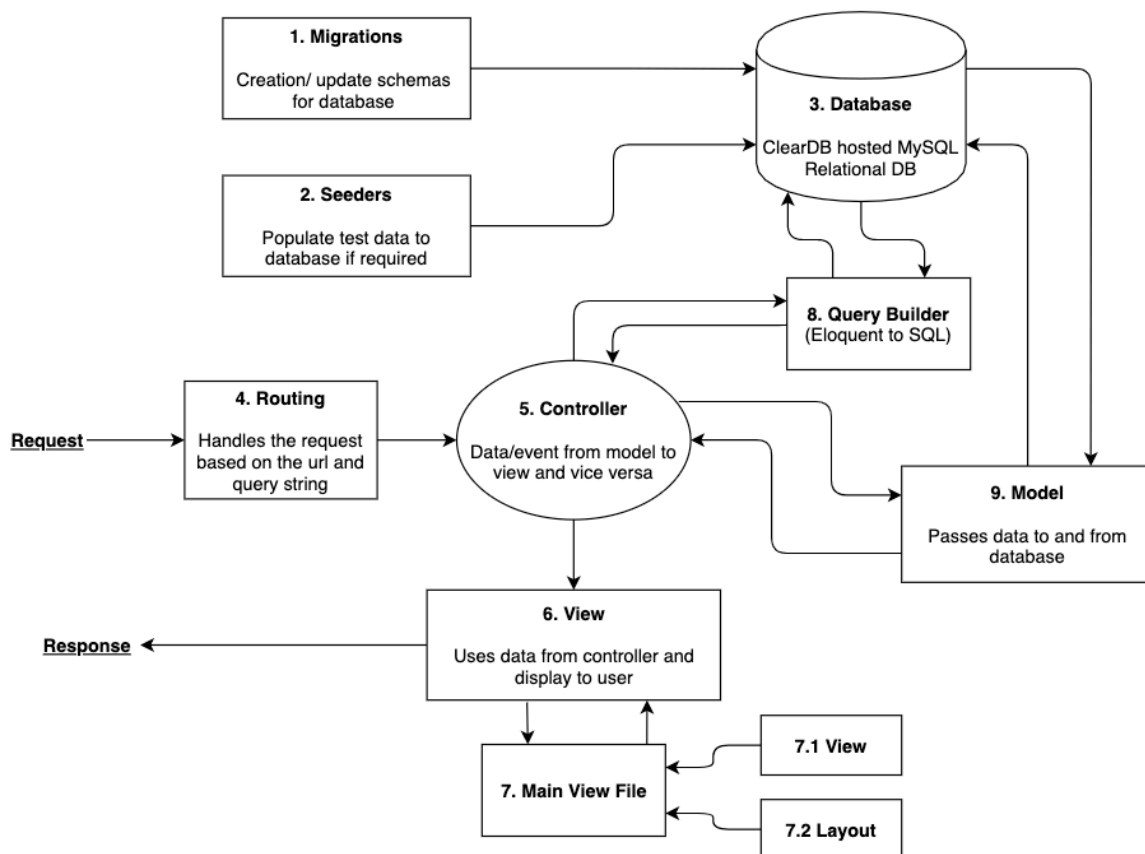
The main requirement of the application was to process and store rainfall sensor data which would later be displayed the user in multiple formats including graphs, along with providing security measures and user accounts.

### 2.1 HTML5, CSS3, Javascript, jQuery and Bootstrap

HTML5, CSS3, Javascript, jQuery and Bootstrap have been used to create an attractive, interactive web application. Bootstrap provides js and CSS classes to produce a responsive, mobilise first application. Many bootstrap features have been implemented, including the navbars, button classes and alerts.

### 2.2 PHP Laravel MVC Framework

Version 5.8.37 of the Laravel Framework was used to create the application with the official documentation [1] being followed throughout. It uses the Model-View-Controller pattern to separate the business logic from the visualisation of data. A diagram of the MVC flow used is in figure 2.



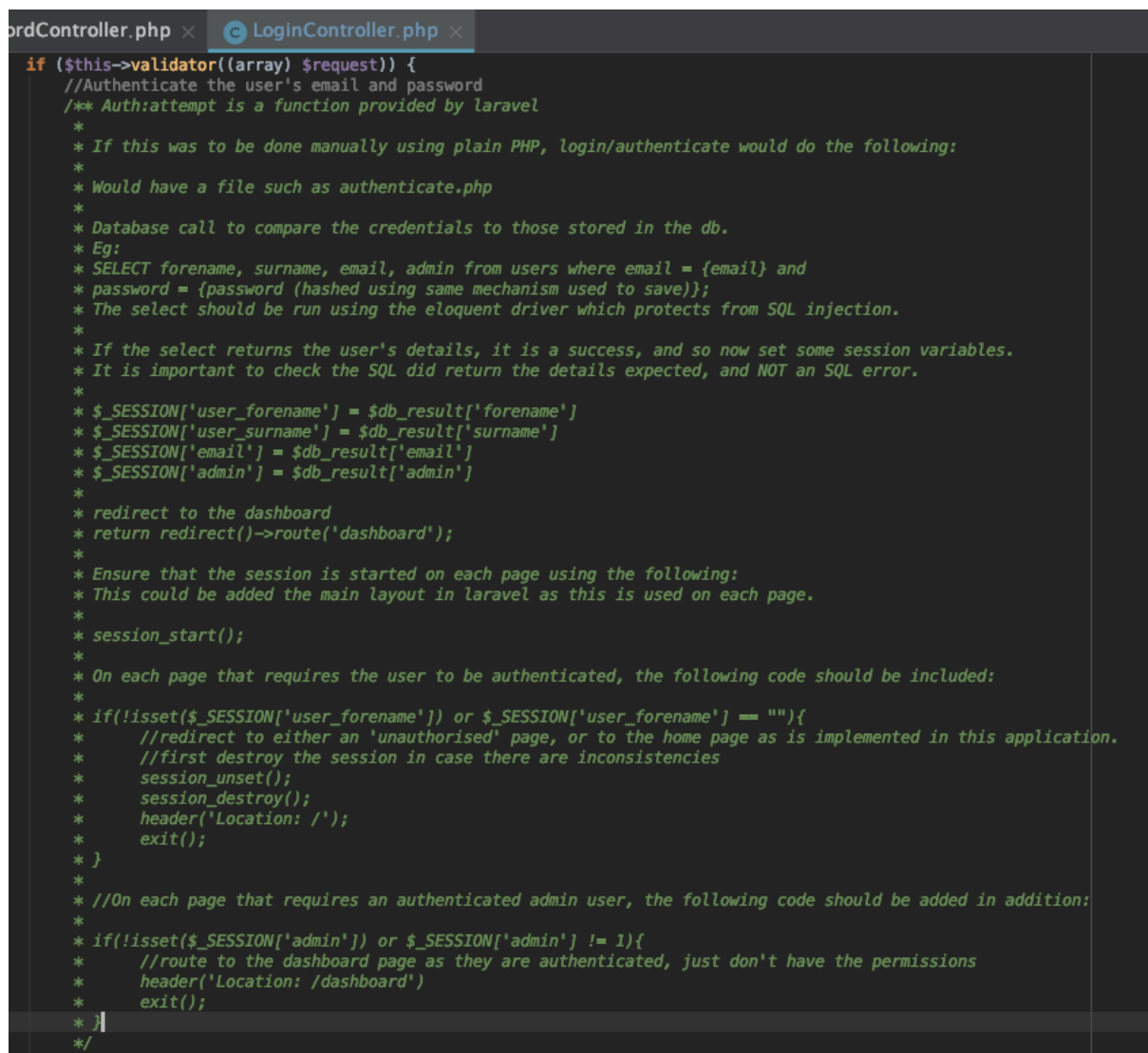
<https://laravel.com/docs/5.6/>

Figure 2. Rainfall-Sensor-App Model-View-Controller Diagram

The benefits of Laravel can be found in Appendix A, along with details about the local development environment.

The author chose to use Laravel over plain PHP for many reasons, firstly as she already had a good knowledge of plain PHP, she wanted to stretch herself to learn new skills, having never used the MVC pattern or Laravel previously. Secondly, due to Laravel's inbuilt libraries more focus could be spent on developing the core application. Also, Laravel includes many security features and dependency management, therefore a more robust and secure application could be built without reinventing the wheel.

Despite using the inbuilt methods, code comments suggest ways the same functionality could be implemented in plain PHP throughout the application, such as below in figure 3.



```

if ($this->validator((array) $request)) {
    //Authenticate the user's email and password
    /** Auth:attempt is a function provided by laravel
    *
    * If this was to be done manually using plain PHP, login/authenticate would do the following:
    *
    * Would have a file such as authenticate.php
    *
    * Database call to compare the credentials to those stored in the db.
    * Eg:
    * SELECT forename, surname, email, admin from users where email = {email} and
    * password = {password (hashed using same mechanism used to save)};
    * The select should be run using the eloquent driver which protects from SQL injection.
    *
    * If the select returns the user's details, it is a success, and so now set some session variables.
    * It is important to check the SQL did return the details expected, and NOT an SQL error.
    *
    * $SESSION['user_forename'] = $db_result['forename']
    * $SESSION['user_surname'] = $db_result['surname']
    * $SESSION['email'] = $db_result['email']
    * $SESSION['admin'] = $db_result['admin']
    *
    * redirect to the dashboard
    * return redirect()->route('dashboard');
    *
    * Ensure that the session is started on each page using the following:
    * This could be added the main layout in laravel as this is used on each page.
    *
    * session_start();
    *
    * On each page that requires the user to be authenticated, the following code should be included:
    *
    * if(!isset($_SESSION['user_forename']) or $_SESSION['user_forename'] == ""){
    *     //redirect to either an 'unauthorised' page, or to the home page as is implemented in this application.
    *     //first destroy the session in case there are inconsistencies
    *     session_unset();
    *     session_destroy();
    *     header('Location: /');
    *     exit();
    * }
    *
    * //On each page that requires an authenticated admin user, the following code should be added in addition:
    *
    * if(!isset($_SESSION['admin']) or $_SESSION['admin'] != 1){
    *     //route to the dashboard page as they are authenticated, just don't have the permissions
    *     header('Location: /dashboard')
    *     exit();
    * }
    */
}

```

Figure 3. Comments inside LoginController

### 2.2.1 Laravel Eloquent ORM

For information on Eloquent, see **Appendix A**.

The Eloquent query builder was used throughout, however in order to show understanding of SQL, there are comments to show the raw SQL equivalent (fig 4).

```
//Get the top 3 recorded rainfall amounts for each sensor
//SELECT * FROM sensor_data where sensor_id = {sensor_id} ORDER BY rainfall_mm DESC LIMIT 3;
$rainfall_amount_res[$i] = SensorData::where('sensor_id', $item)->orderBy('rainfall_mm', 'DESC')->take(3)->get();
```

Figure 4. Eloquent Query Builder vs Raw SQL (Annotated Code Screenshot from SearchSensorDataController)

Eloquent provides built in database connections, however in plain PHP, a database connection could be initiated using a script as follows in figure 5.

```
function connect_to_db(){
    /**
     * Declare variables to store the database host and credentials.
     * $username : the database user (often named Admin or Root, but a
     * user has been created by ClearDB using a unique string)
     * $password : the database password
     * $hostname : the hostname of the database (provided by clearDB.
     * Would be localhost if a local server was used.)
     */
    $hostname = "heroku_b8c95babb9ee949.us-cdbr-iron-east-04.cleardb.net";
    $username = "b459a5982e250f";
    $password = "{db-password-here}";

    //Create a connection to database using the above credentials
    $conn = new mysqli($hostname, $username, $password);

    //Check the connection has been successful or handle appropriately.
    if ($conn->connect_error) {
        //return a message to the calling function
        return("Connection failed. Error: " . $conn->connect_error);
    }
    //When the script ends the connection will automatically be closed.
}
```

Figure 5. Example database connection through mysqli and php

## 2.3 MySQL Database

A MySQL database was chosen as it is integrated within Laravel and provides an application (MySQL Workbench) to run commands and visualise data locally. SQLite was considered, however separation between the app and database was more suitable rather than a server-less database, due to the modularity of the MVC structure.

As the DB (database) must be accessed from both the local environment and the live application, it had to be hosted using ClearDB. A caveat of this is that the application can be slow to access as the requests are throttled by the provider on the free-tier [2].

The database does not store sensitive information however passwords must be obscured before they are written to the DB, and so they are hashed using 10 rounds of BCrypt for security.

The database structure and field lengths, migrations and seeders are described in more detail in **Appendix B**, along with the assumptions made for this project.

## 2.4 Amazon Web Services (AWS) PHP SDK and S3 Storage

As stated in **Appendix B**, to make the application more production like, the sensor data should be uploaded to a single location. S3 was the storage solution of choice as it's free and provides a simple PHP SDK to access the files.

The file processing functionality is thoroughly annotated within the code, as sampled below in figure 6.

```
* index function
*
* It was decided that an S3 bucket should be used so that different vendors could upload sensor data to an
* accessible centralised location.
*
* They could do this through programmatic access via the AWS S3 SDK and a pre-signed URL.
*
* Firstly grabs a list of files inside the rainfall-sensor-data-files AWS S3 bucket.
* If there are no files, return nothing.
*
* Exceptions are handled through try-catch blocks.
*
* The index function makes a call to the above function process_file
*/
public function index()
{
    /**
     * Get list of files currently in the S3 Bucket.
     * The only files in the bucket will be unprocessed files, as processed files are deleted if successfully saved to the DB.
     */
    try {

        $file_list = array(); //declare array to hold list of file names.
        $key = 0; //declare loop variable and initialise to 0.

        $s3 = AWS::createClient( 'name': 's3' ); //create new S3 client (from PHP SDK)

        //get the list of files (objects) in the S3 Bucket.
        $objects = $s3->listObjects([
            'Bucket' => config( 'key': 'AWS_BUCKET', 'default': 'rainfall-sensor-data-files' )
        ]);

        //Loop through the results and create an array of file names to use later.
        //check if null first.
        if ($objects['Contents'] != null){
            foreach ($objects['Contents'] as $object) {
                echo ' File {' . ($key + 1) . ' } : ' . $object['Key'] . ' , ' ; //testing variable
                $file_list[$key] = $object['Key'];
                $key++; //increment loop variable.
            }
        }

    } catch (Exception $e) { //catch any exceptions and flash a message to the user.
        //separate catch block based on action
        session()->flash('error', 'Error accessing files from S3. ');
        return view( view: 'layouts/processed-files' )->withData(null);
    }
}
```

Figure 6. Small snippet of annotated code from the ProcessTextFileController.

The process of getting file names from S3, calling S3 for the file contents, processing the data, writing to the database and finally deleting the file from S3 is not the most efficient and can be costly in request times. Also, the S3 bucket is only checked for new data when the *process-text files* process is triggered within the text-file view.

A way to improve this would be to use an AWS serverless-lambda function containing a script to check the bucket contents and write the data to the database. This lambda could be run at intervals such as every 2 hours and invoked by a CloudWatch event.

## 2.5 GitHub and Heroku

Throughout development, Git and GitHub have been used for source control. It was decided the application be hosted for staging and that Heroku would be suitable as it's easily

integrated with Laravel, provides free hosting and adequate performance. AWS (Amazon Web Services) was considered, however would've required the networking configuration even when using tools such as Elastic Beanstalk, and unlike Heroku did not have simple integration into GitHub for auto-deployments.

When code is pushed to the test branch, a pipeline deploys code to the Heroku site (see **Appendix C**).

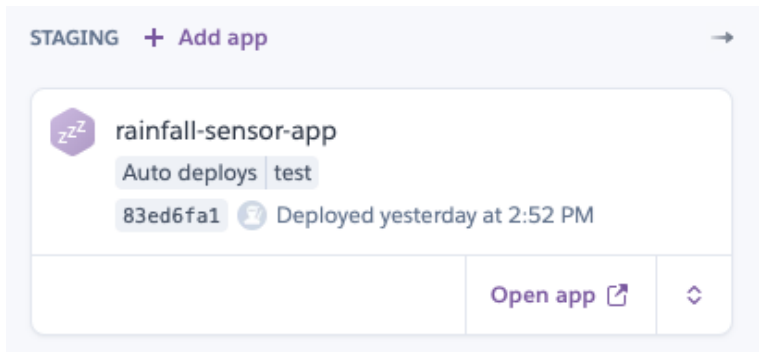


*Figure 7. Rainfall sensor app pipeline triggered by Git push to test*

## 2.6 Chart.js

See **Appendix D** for information on Chart.js and JpGraph. Chart.js was chosen because of its visual effects and functionality to dynamically build responsive, animated charts through Javascript.

JpGraph was also considered however as the image is generated on the server in a specific size before it's returned to the client, the images cannot be resized upwards if quality is to be retained. Also, as they are static images, it also lacks the user engagement compared to chart.js as it does not allow for animations or dynamic filtering using Javascript.

### 3. Application Functionality

#### 3.1 User Authentication and Authorisation

See **Appendix F** for full user functionality.

The inbuilt Auth methods have been used however annotated code within the app/Http/Controllers/Auth/LoginController suggests how authentication and security should be managed in plain PHP (see section 1.1).

There are two account types, standard and admin user. An admin is defined using a Boolean flag in the users table, which can only be set by a database administrator.

Non-authenticated users can only view the welcome, login, forgotten password, and registration page. If they attempt to access any other pages/routes they are redirected to the login page.

Non-admin users have access to all pages except file processing. If they attempt to access '/text-file' or '/process-files', they are redirected to the dashboard (code shown below).

```
{{!--The landing page for processing files - provides a link to run the processing --}}
@extends('layouts.mainlayout')
@section('content')
    @if(Auth::user()->admin==1)
        <div class="container mt-4">...</div>
    @else
        {{!-- If the user is not an admin, route them to the dashboard. --}}
        @php header('Location: /dashboard'); exit(); //Redirect user if they are not an admin @endphp
    @endif
@endsection
```

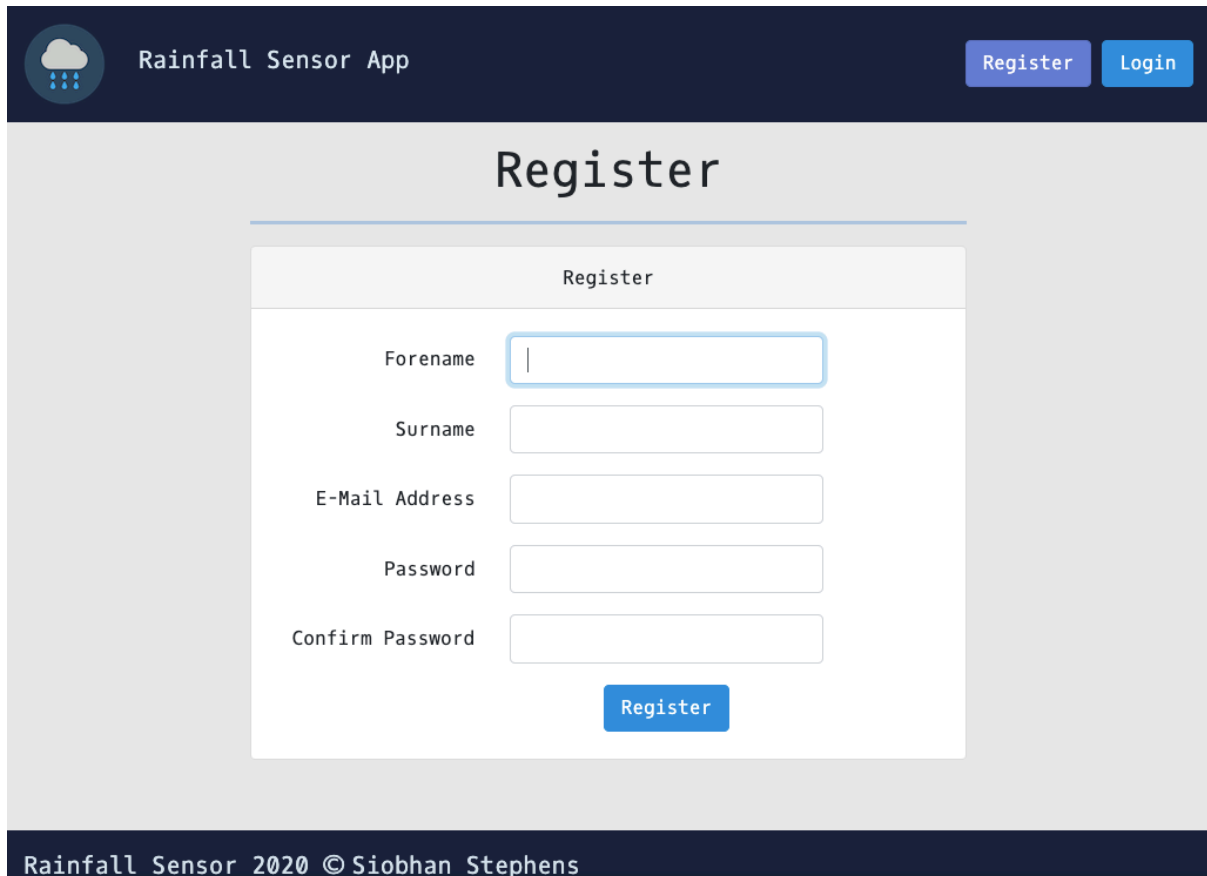
They also cannot view the sensor\_data\_id field in the data tables. Below is a side by side comparison of admin and non-admin view (fig 8).

| Sensor data for dates between 16/12/2019 and 16/12/2019 |                     |
|---------------------------------------------------------|---------------------|
| Sorted by most recent recorded events.                  |                     |
| Rainfall Amount                                         | Datetime Recorded   |
| 0                                                       | 16/12/2019 20:15:51 |
| 0                                                       | 16/12/2019 18:15:51 |
| 0                                                       | 16/12/2019 16:15:51 |
| 0                                                       | 16/12/2019 14:15:51 |
| 0                                                       | 16/12/2019 12:15:51 |
| 0                                                       | 16/12/2019 10:15:51 |

| Sensor data for dates between 16/12/2019 and 16/12/2019 |                 |                     |
|---------------------------------------------------------|-----------------|---------------------|
| Sorted by most recent recorded events.                  |                 |                     |
| Sensor Data Id                                          | Rainfall Amount | Datetime Recorded   |
| 272                                                     | 0               | 16/12/2019 20:15:51 |
| 262                                                     | 0               | 16/12/2019 18:15:51 |
| 252                                                     | 0               | 16/12/2019 16:15:51 |
| 242                                                     | 0               | 16/12/2019 14:15:51 |

Figure 8. Non-admin user result on the left, admin user result on the right

Users can register for an account and will be given non-admin access after confirming their email address. The registration form is validated using Laravel's validator, with consideration to field lengths, types, and unique values (fig 10).



The screenshot shows the 'Rainfall Sensor App' interface. At the top, there is a dark blue header with a cloud and rain icon on the left, the text 'Rainfall Sensor App' in the center, and 'Register' and 'Login' buttons on the right. Below the header, the main content area has a light gray background. In the center, there is a white box titled 'Register'. Inside this box, there are five input fields labeled 'Forename', 'Surname', 'E-Mail Address', 'Password', and 'Confirm Password'. A blue 'Register' button is positioned at the bottom right of the white box. At the very bottom of the page, a dark blue footer contains the text 'Rainfall Sensor 2020 © Siobhan Stephens'.

Figure 9. User registration form

```
protected function validator(array $data)
{
    //Validate the new user data
    return Validator::make($data, [
        'forename' => ['required', 'string', 'max:60'],
        'surname' => ['required', 'string', 'max:60'],
        'email' => ['required', 'string', 'email', 'max:65', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'max:30', 'confirmed'],
    ]);
}
```

Figure 10. Validation for user registration form

Had this validation been done using plain PHP, the code below (fig 11) could be implemented, with a unique constraint being on the database email field.

```

public function validate_registration(array $registration_data){
    $forename = $registration_data['forename'];
    $surname = $registration_data['surname'];
    $email = $registration_data['email'];
    //password should be passed to the function in plaintext.
    $password = $registration_data['pwd'];
    //password regex from Srinivas, stackoverflow - https://stackoverflow.com/users/3070595/srinivas
    $password_regex = "^(?=.*(?=\S{8,}))?(?=\S*[a-z])(?=\S*[A-Z])(?=\S*[\d])\S*$";
    $name_regex = "/^[a-zA-Z\s,\.\'-\+$/1";
    //email regex from Michael Rushkott - http://lxr.php.net/xref/PHP_5_4/ext/filter/logical_filters.c#501
    $email_regex = '/^(?!(?:\x22?\x5C(\x00-\x7E)\x22)?(?:\x22?[\^\x5C\x22]\x22?)){255,})(?!(?:\x22?\x5C(\x00-\x7E)\x22)?(?:\x22?[\^\x5C\x22]\x22?))';

    //initialise a validation result - this will not change if all checks are passed.
    $validation_result[0] = true;
    $validation_result[1] = "Valid Data";

    //Check length of forename is between 2 and 60 characters, and that it matches an expected name format (A-Z, a-z, -, spaces etc).
    if(strlen($forename) < 2 or strlen($forename) > 60 or !preg_match($name_regex, $forename)){
        $validation_result[0] = false;
        $validation_result[1] = "Invalid Forename. Forename must be between 2 and 60 characters and contain letters, spaces and limited characters only.";
        return $validation_result;
    }
    //Check length of surname is between 2 and 60 characters, and that it matches an expected name format (A-Z, a-z, -, spaces etc).
    if(strlen($surname) < 2 or strlen($surname) > 60 or !preg_match($name_regex, $surname)){
        $validation_result[0] = false;
        $validation_result[1] = "Invalid Surname. Surname must be between 2 and 60 characters and contain letters, spaces and limited characters only.";
        return $validation_result;
    }
    //Check length of email is between 4 and 65 characters, and that it matches an expected email format.
    if(strlen($email) < 4 or strlen($email) > 65 or !preg_match($email_regex, $email)){
        $validation_result[0] = false;
        $validation_result[1] = "Invalid Email Address. Please provide a valid email address as you will need to confirm it.";
        return $validation_result;
    }
    //Check length of password is between 8 and 30 characters, and that it includes both uppercase and
    // lowercase characters, a digit and can contain special characters.
    if(!preg_match($password_regex, $password) or strlen($password) < 8 or strlen($password) > 30){
        $validation_result[0] = false;
        $validation_result[1] = "Invalid password. Password must be between 8 and 30 characters,
        contain at least 1 uppercase and lowercase letter, at least 1 digit, and
        may contain special characters.";
        return $validation_result;
    }
    return $validation_result;
}

```

Figure 11. Suggested registration form validation, if handled manually using plain PHP and regular expressions

Infall Sensor App

## Login

Login

E-Mail Address

Password

[Login](#) [Forgot Your Password?](#)

Figure 12. Login form

Infall Sensor App

## Reset Password

E-Mail Address

[Send Password Reset Link](#)

Figure 13. Reset password form

## Rainfall Sensor

**Hello!**

You are receiving this email because we received a password reset request for your account.

[Reset Password](#)

This password reset link will expire in 60 minutes.

If you did not request a password reset, no further action is required.

Regards,  
Rainfall Sensor

---

If you're having trouble clicking the "Reset Password" button, copy and paste the URL below into your web browser: <http://rainfall-sensor-app.herokuapp.com/password/reset/03bede4e8a1e73072f18cf2e68c85dda10884ac1830a8f81f16b4f5772e7b348?email=siobhan.k.stephens%40gmail.com>

*Figure 14. Password reset email*

## 3.2 Searches Implemented

The user may search for rainfall data using the following PHP form (fig 12). It uses Javascript/jQuery to hide/show fields based on the selection (fig 16).

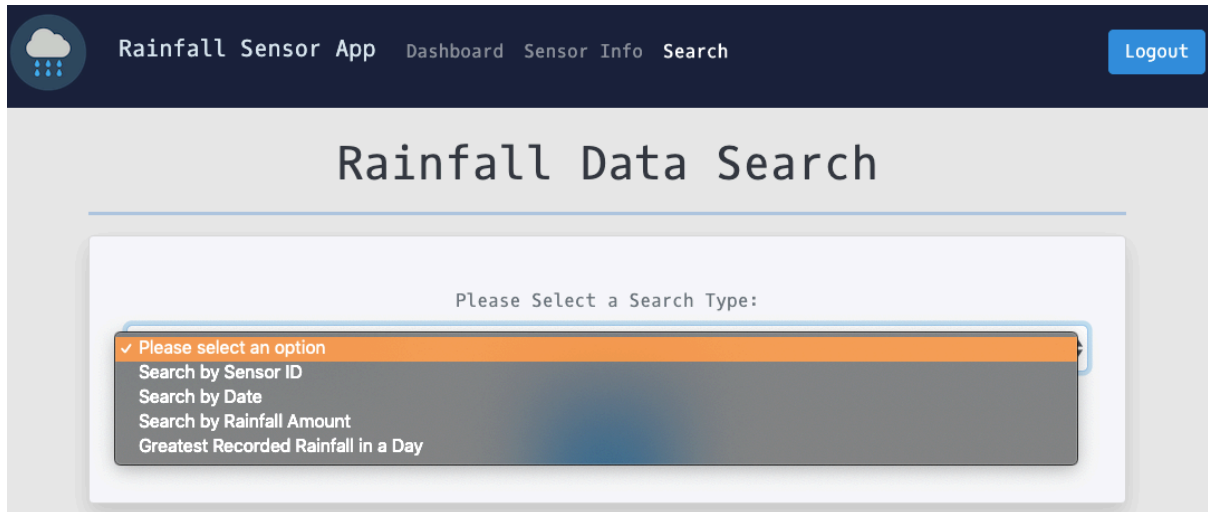


Figure 15. Search Form

```
$(function () {
    var rainfall_amt_field = $('#rainfall_amount_field');
    var date_field = $('#date_field');
    var sensor_id_field = $('#sensor_id_field');
    var search_type = $('#search_type');

    //hide the search fields on page load
    rainfall_amt_field.hide();
    date_field.hide();
    sensor_id_field.hide();

    //function triggered if the dropdown value is changed.
    search_type.change(function(){

        //hide the extra fields and clear their values
        rainfall_amt_field.hide();
        rainfall_amt_field.val( '');
        date_field.hide();
        date_field.val( '');
        sensor_id_field.hide();
        sensor_id_field.val( '');

        //show the fields based on the value of the dropdown.
        if(search_type.val() == 'sensor_id'){
            sensor_id_field.show();
        }else if(search_type.val() == 'date'){
            date_field.show();
        }else if(search_type.val() == 'rainfall'){
            rainfall_amt_field.show();
        }
    });
});
```

Figure 16. public/js/search-form.js file

The SearchSensorData controller processes searches using switch statements based on search type. The build\_return function aids in the reusability of the code, as it can be called with different parameters by different search types and could be reused if a new search was to be added.

Each rainfall search uses a different view to display the graph/table. The conditional logic for this is within the rainfall-charts view (fig 17).

```

{{-- The result page for the search-form. The individual search result sections are
included in below using the if statements. --}}
@extends('layouts.mainlayout')
@section('content')
    @if(Auth::user())
        {{--If the search type isn't set, route to the search form --}}
        @php if(!isset($type)){ header("Location: /search-form"); exit(); } @endphp

        <div class="album text-muted">
            <div class="container py-4 pb-5">
                <h1 class="text-center">Rainfall Data Search</h1>
                <hr class="bold-hr"/>

                <h3 class="text-center mt-4">Search Results</h3>
                <hr class="subheading-bold-hr-blue"/>

                @if($data == null or count($data)<1)
                    <div class="alert alert-info mx-auto text-center p-3">
                        <h5 class="text-center">No Data Available for this Search.</h5><br/>
                        <a class="mx-auto btn btn-info text-white" href="/search-form"><i class="fas fa-undo"></i> Back to Search</a>
                    </div>
                @else
                    <div class="mb-4 text-center">
                        <button onclick="window.print()" class="text-center no-print text-white btn btn-info">Print Results</button>
                    </div>
                    {{--Build different charts based on the search type--}}
                    @if($type == "sensor_id")
                        @include('layouts.partials.sensor_id_search')
                    @elseif($type == "date")
                        @include('layouts.partials.date_search')
                    @elseif($type == "rainfall")
                        @include('layouts.partials.rainfall_search')
                    @elseif($type == "wettest")
                        @include('layouts.partials.wettest_search')
                    @endif
                    <div class="mt-4 mx-auto text-center">
                        <hr class="bold-hr"/>
                        <a class="mx-auto btn btn-info text-white" href="/search-form"><i class="fas fa-undo"></i> Back to Search</a>
                    </div>
                @endif
            </div>
        </div>
    @endif
@endsection

```

Figure 17. Conditional includes based on search type and result

### 3.2.1 Search by Sensor ID

The sensor id search returns a maximum of 30 results for a given sensor (fig 18). This is useful to find out the intervals at which data is being recorded for the sensor, and to find out when it was last recording rainfall data.

Figure 18. Search by Sensor ID

Error.

The sensor id must be an integer.

Please try again.

Figure 19. Validation message when 'abc' was entered for sensor ID

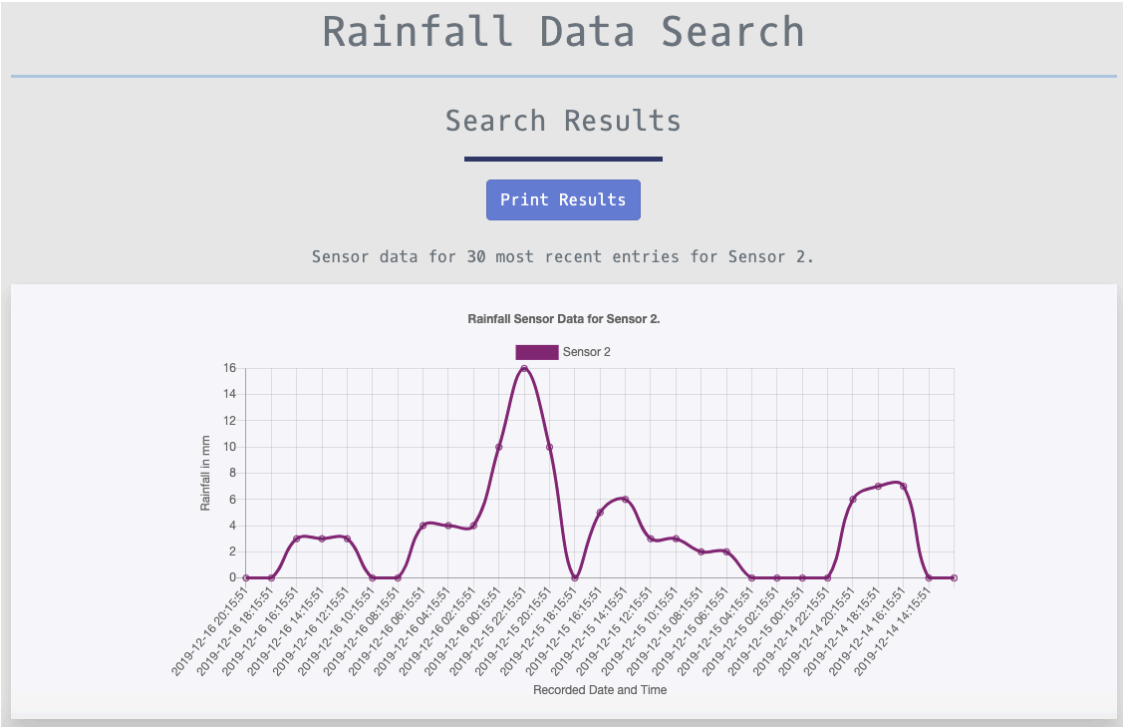


Figure 20. Sensor ID 2 search result graph

30 Most Recent Sensor Data Items for Sensor 2

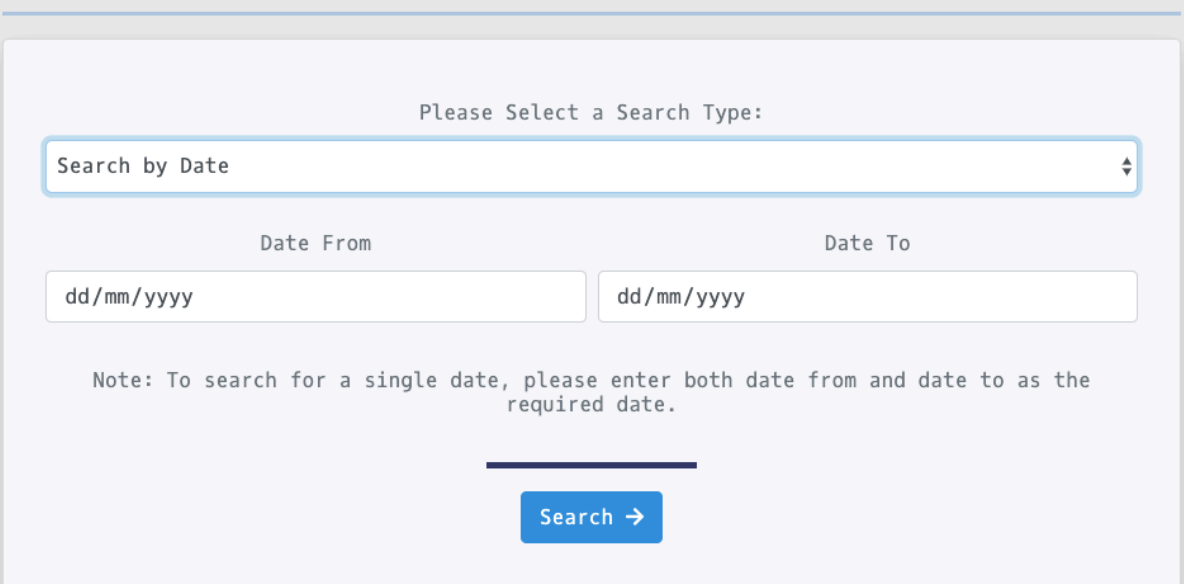
| Rainfall Amount | Datetime Recorded   |
|-----------------|---------------------|
| 0               | 16/12/2019 20:15:51 |
| 0               | 16/12/2019 18:15:51 |
| 3               | 16/12/2019 16:15:51 |
| 3               | 16/12/2019 14:15:51 |
| 3               | 16/12/2019 12:15:51 |
| 0               | 16/12/2019 10:15:51 |
| 0               | 16/12/2019 08:15:51 |
| 4               | 16/12/2019 06:15:51 |
| 4               | 16/12/2019 04:15:51 |
| 4               | 16/12/2019 02:15:51 |

Figure 21. Sensor ID 2 search result table

A future improvement would be to return only the first 10 results and include pagination so more information can be returned in total without overloading the user.

### 3.2.2 Search by Date

A search for filtering results by date has been implemented. The user may enter dates up to and including the current date, and if a single day is required the user can enter the same date twice (fig 22).



The form is titled "Please Select a Search Type:" and features a dropdown menu with "Search by Date" selected. Below this are two input fields labeled "Date From" and "Date To", both containing the placeholder "dd/mm/yyyy". A note below the fields states: "Note: To search for a single date, please enter both date from and date to as the required date." At the bottom, there is a blue "Search →" button.

Figure 22. Search by Date Form

The result produces a chart.js graph for a maximum of 30 results for each sensor (fig 23).

Sensor data for dates between 16/12/2019 and 15/12/2019 for each sensor.

Please note, the search is limited to return a maximum of 30 records.

Click on a sensor identifier below to remove it from the chart.

ALL sensor data for dates between 16/12/2019 and 15/12/2019 for each sensor.

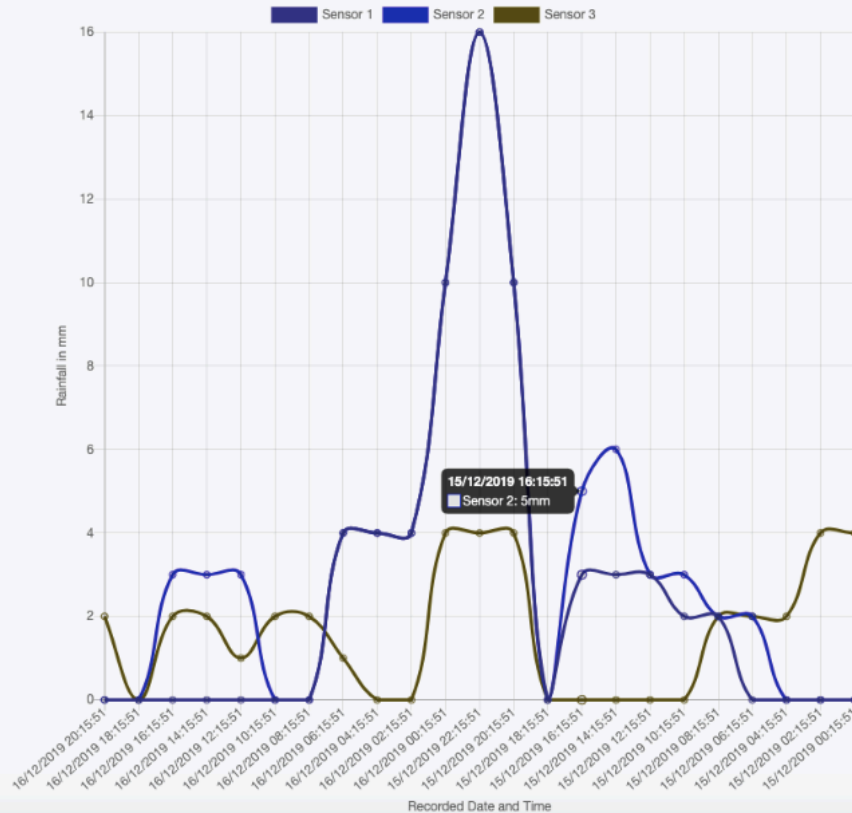


Figure 23. Search by date results for two given dates

The graph can be then be filtered for each sensor by clicking on the identifier as shown in fig 24.

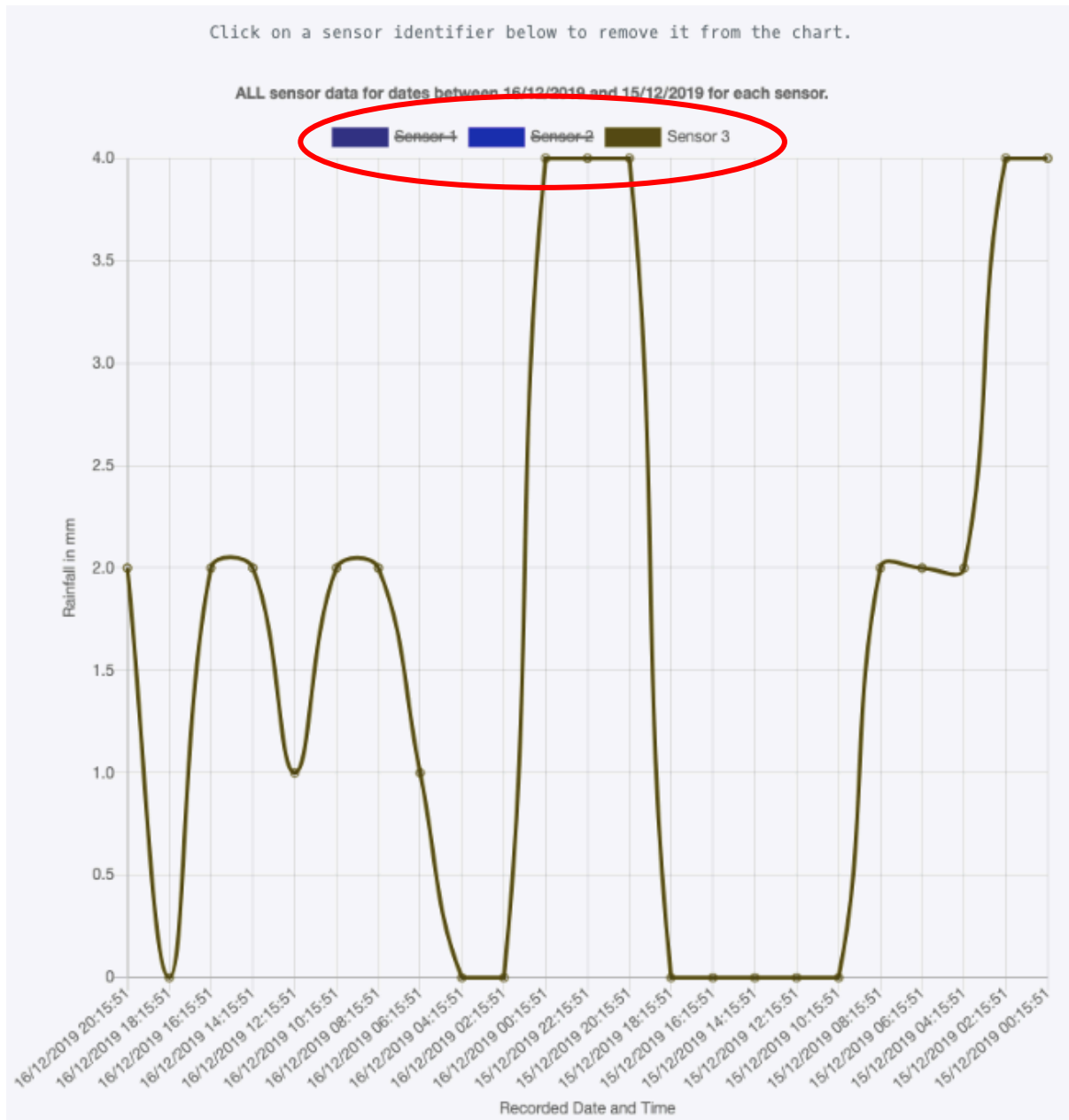


Figure 24. Graph result filtered to Sensor 3 only

Charts are built by injecting PHP into the JavaScript using both blade and php tags, as shown below in fig 25.



```

switch ($search_type) {
    case "sensor_id":
        $search_data = $request->validate([
            'sensor_id' => 'bail|required|integer|min:1|max:1000000'
        ]);
        $sensor_data = SensorData::where('sensor_id', $search_data['sensor_id'])->orderBy('datetime_recorded', 'DESC')->take(30)->get();
        return view( view: 'layouts/rainfall-charts' )->withData($sensor_data)->withType($validated_data['search_type']);
        break;

    case "date":
        /**
         * Validate input fields:
         * Date must be today's date or before (hence the use of date now plus one).
         */
        $search_data = $request->validate([
            'date_from' => 'bail|required|date|before:' . $date_now_plusone,
            'date_to' => 'bail|required|date|before:' . $date_now_plusone
        ]);
        if ($search_data) {
            //Add 1 day to the code so that sensor data is returned for that date.
            $date_to_plusone = date( format: "Y-m-d", strtotime( time: $search_data['date_to'] . ' +1 day' ));
            $search_array = array($search_data['date_from'], $date_to_plusone);
            $return_res = $this->build_return($search_type, $search_array);

            $date_from_format = date( format: "d/m/Y", strtotime($search_data['date_from']));
            $date_to_format = date( format: "d/m/Y", strtotime($search_data['date_to']));

            return view( view: 'layouts/rainfall-charts' )->withData($return_res[0])->withType($search_type)->withDates($return_res[1])->withSearch(array($date_
        } else {
            $request->session()->flash('error', 'Invalid Search Type.');
```

Figure 27. Switch statement for date inside search function of SearchSensorDataController

```

foreach ($sensor_id_list_res as $key) {
    $sensor_id_list[$i] = $key->sensor_id;//create an array of sensor_ids.
    $i++;//Increment loop variable.
}
$i = 0;//reset the loop variable $i to be used later.
foreach ($sensor_id_list as $item) {
    switch ($search) {
        case 'rainfall':
            //Get ALL results for recorded date for each sensor where the recorded rainfall was between the two search amounts
            //SELECT * FROM sensor_data where sensor_id = {sensor_id} AND rainfall_mm BETWEEN {amount_from} and {amount_to} ORDER BY datetime_recorded DESC LIMIT 30;
            //Limit all results to a maximum 30 records. A future version may implement a smaller amount and a 'more' button to release items in groups of 10 etc.
            $rainfall_amount_res[$i] = SensorData::where('sensor_id', $item)->whereBetween('rainfall_mm', [$search_params[0], $search_params[1]])->orderBy('datetime_recorded',
            break;
        case "date":
            //get ALL results for recorded data for each sensor between the two provided dates
            //SQL Equivalent:
            //SELECT * FROM sensor_data where sensor_id = {sensor_id} AND datetime_recorded BETWEEN {date_from} and {date_to} ORDER BY datetime_recorded DESC;
            //Limit all results to a maximum 30 records. A future version may implement a smaller amount and a 'more' button to release items in groups of 10 etc.
            $rainfall_amount_res[$i] = SensorData::where('sensor_id', $item)->whereBetween('datetime_recorded', [$search_params[0], $search_params[1]])->orderBy('datetime_recor
            break;
        case "wettest":
            //Get the top 3 recorded rainfall amounts for each sensor
            //SELECT * FROM sensor_data where sensor_id = {sensor_id} ORDER BY rainfall_mm DESC LIMIT 3;
            $rainfall_amount_res[$i] = SensorData::where('sensor_id', $item)->orderBy('rainfall_mm', 'DESC')->take(3)->get();
    }
    //check if there are any sensor data items for that sensor (a sensor could be newly installed and not have any recorded data).
    if (count($rainfall_amount_res[$i]) < 1) {
        continue;//if there are no data items, don't write it to the data item array. Skip this iteration of the foreach loop.
    } else {
        $rainfall_amount_array[$i] = $rainfall_amount_res[$i];
        foreach ($rainfall_amount_array[$i] as $sensor_data_item) {
            $datetime_array[$j] = $sensor_data_item->datetime_recorded;//build an array of dates which will later be made unique
            $j++;//increment loop variable
        }
    }
    $i++;
}
//create an array of unique dates (later used as the label of the chart's x axis)
$datetime_array = array_unique($datetime_array);

//create a return variable and return to the calling function.
$return_arr[0] = $rainfall_amount_array;
$return_arr[1] = $datetime_array;
} catch (Exception $e){
    session()->flash('error', 'Sorry, there was an issue processing your search. Please try again later.');
```

Figure 28. Main processing code in build\_return function of SearchSensorDataController

To improve this search, additional optional time fields could be added to search for a specific timeframe, or the date could be grouped by sum of rainfall for each day so that more data could be viewed in a simplified graph.

### 3.2.3 Search by Rainfall Amount (mm)

The rainfall amount search allows the user to enter two values as search parameters, the search form is validated to accept values between 0 and 600 (mm).

## Rainfall Data Search

Please Select a Search Type:

Search by Rainfall Amount

Amount From

Amount From...

Amount To

Amount To...

Note: To search for a single amount, please enter both from and to amounts as the required amount.

Search →

Figure 29. Search by rainfall amount

The result of the search produces a line graph for a maximum of 30 records per sensor (fig 30), which can be filtered per sensor. Table data is also displayed to the user (fig 31).

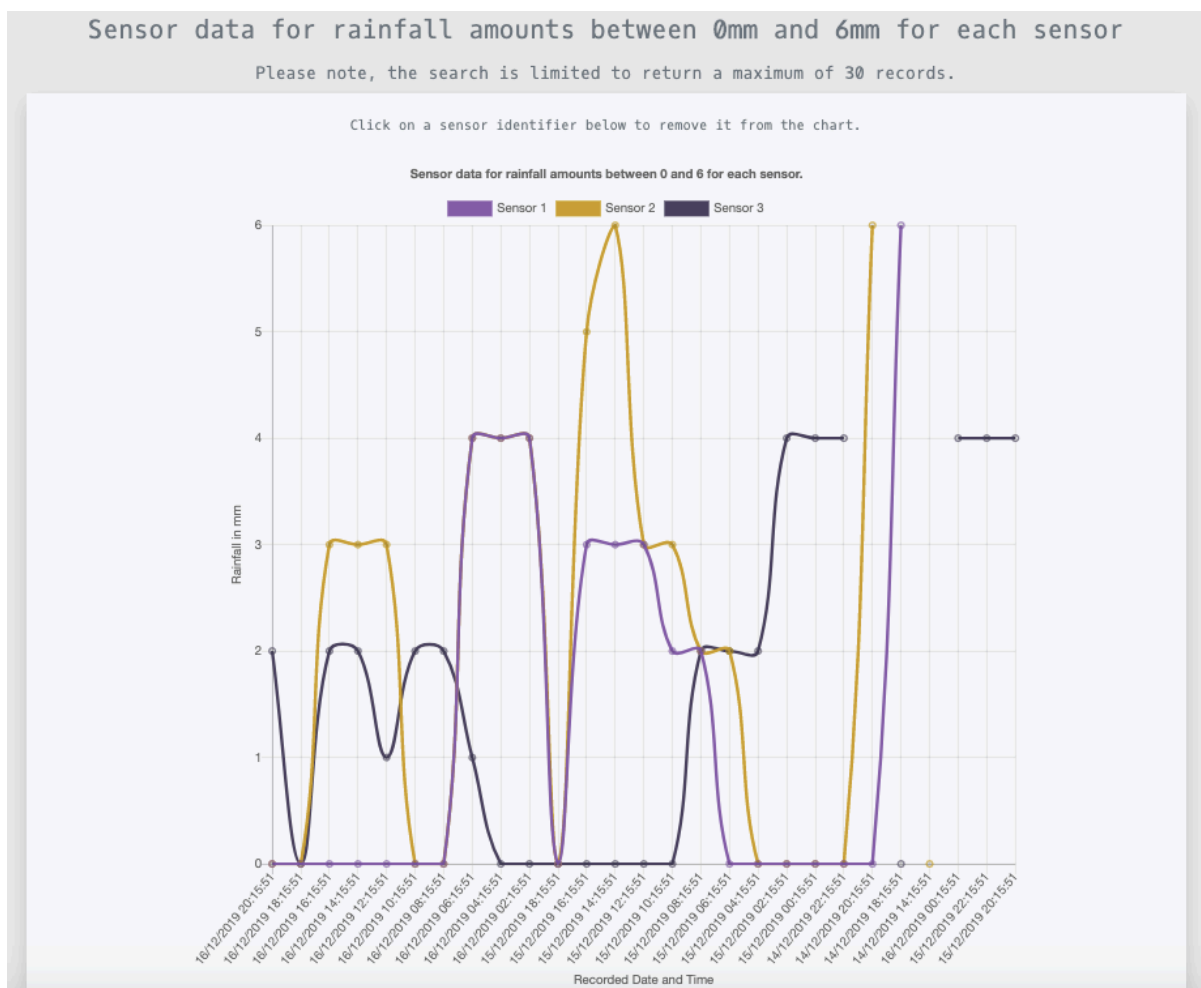


Figure 30. Search result for rainfall amounts between 0mm and 6mm

| Sensor 2        |                     |
|-----------------|---------------------|
| Rainfall Amount | Datetime Recorded   |
| 0               | 16/12/2019 20:15:51 |
| 0               | 16/12/2019 18:15:51 |
| 3               | 16/12/2019 16:15:51 |
| 3               | 16/12/2019 14:15:51 |
| 3               | 16/12/2019 12:15:51 |
| 0               | 16/12/2019 10:15:51 |
| 0               | 16/12/2019 08:15:51 |
| 4               | 16/12/2019 06:15:51 |
| 4               | 16/12/2019 04:15:51 |
| 4               | 16/12/2019 02:15:51 |
| 0               | 15/12/2019 18:15:51 |
| 5               | 15/12/2019 16:15:51 |
| 6               | 15/12/2019 14:15:51 |
| 3               | 15/12/2019 12:15:51 |
| 3               | 15/12/2019 10:15:51 |
| 3               | 15/12/2019 08:15:51 |
| 0               | 15/12/2019 06:15:51 |

Figure 31. Search result table for the above search for sensor 2

This search is especially useful when searching for extreme values, such as from 40 to 100 etc. This will show the areas in which extreme rainfall has previously been recorded as the sensor table headers provide a link to the sensor information include its location on a map.

### 3.2.4 Show Greatest Recorded Rainfall in a Day

The greatest rainfall recorded in a day search returns a single row result for each sensor - including the sum of rainfall for that day, and the date.

It is calculated using the following code and SQL statement for each sensor (fig 32).

```

/** Equivalent SQL for getting the Wettest DAY (sum of recorded rainfall)
 *
 * SELECT sensor_id, SUM(rainfall_mm) as rainfall_mm, DATE(datetime_recorded) as date_recorded FROM
 * sensor_data sd WHERE sensor_id = 1 GROUP BY DATE(datetime_recorded) ORDER BY SUM(rainfall_mm) LIMIT 1;
 *
 * Caveat - each sensor's 'wettest day' is individual and may not necessarily mean it is the wettest overall,
 * UNLESS it can be safely assumed that ALL data is recorded at the exact same intervals, eg every 2 hours.
 * Otherwise, the sum will differ.
 */

//declare and initialise variables
$sensor_id_list = array();
$sum_rainfall_res = array();
$sum_rainfall_array = array();
$i = 0;
//Get a list of ALL sensors to be used later.
$sensor_id_list_res = SensorInformation::select('sensor_id')->get();
foreach ($sensor_id_list_res as $key) {
    $sensor_id_list[$i] = $key->sensor_id; //create an array of sensor_ids.
    $i++; //increment loop variable.
}
$i = 0; //reset the loop variable $i to be used later.
foreach ($sensor_id_list as $sensor) {
    //use parameters to escape sql injection, however is not required as the result has come from the db.
    $sum_rainfall_res[$i] = \DB::select('SELECT sensor_id, SUM(rainfall_mm) as sum_rainfall_mm, DATE(datetime_recorded) as date_recorded FROM
    sensor_data sd WHERE sd.sensor_id=? GROUP BY DATE(sd.datetime_recorded) ORDER BY SUM(sd.rainfall_mm) LIMIT 1', [$sensor]);
    //If there are no results from the select
    if (count($sum_rainfall_res[$i]) < 1) {
        continue; //if there are no data items, don't write it to the data item array. Skip this iteration of the foreach loop.
    } else {
        //else assign the result to the rainfall array which will later be returned.
        $sum_rainfall_array[$i] = $sum_rainfall_res[$i][0];
    }
    $i++;
}
//Return the view with the requested data and the search type.
return view( view: 'layouts/rainfall-charts' )->withData($sum_rainfall_array)->withType($search_type);
break;

```

Figure 32. Greatest rainfall in a day code in SearchSensorDataController search function

The processed SQL result is returned to the rainfall-charts view as a 3D array, along with the original search type, which is used to show a different partial-view based on the condition, in this case, 'wettest\_search'.

In this view, the following bar chart and tables are created dynamically using PHP from the result of the search (fig 33 and 34).

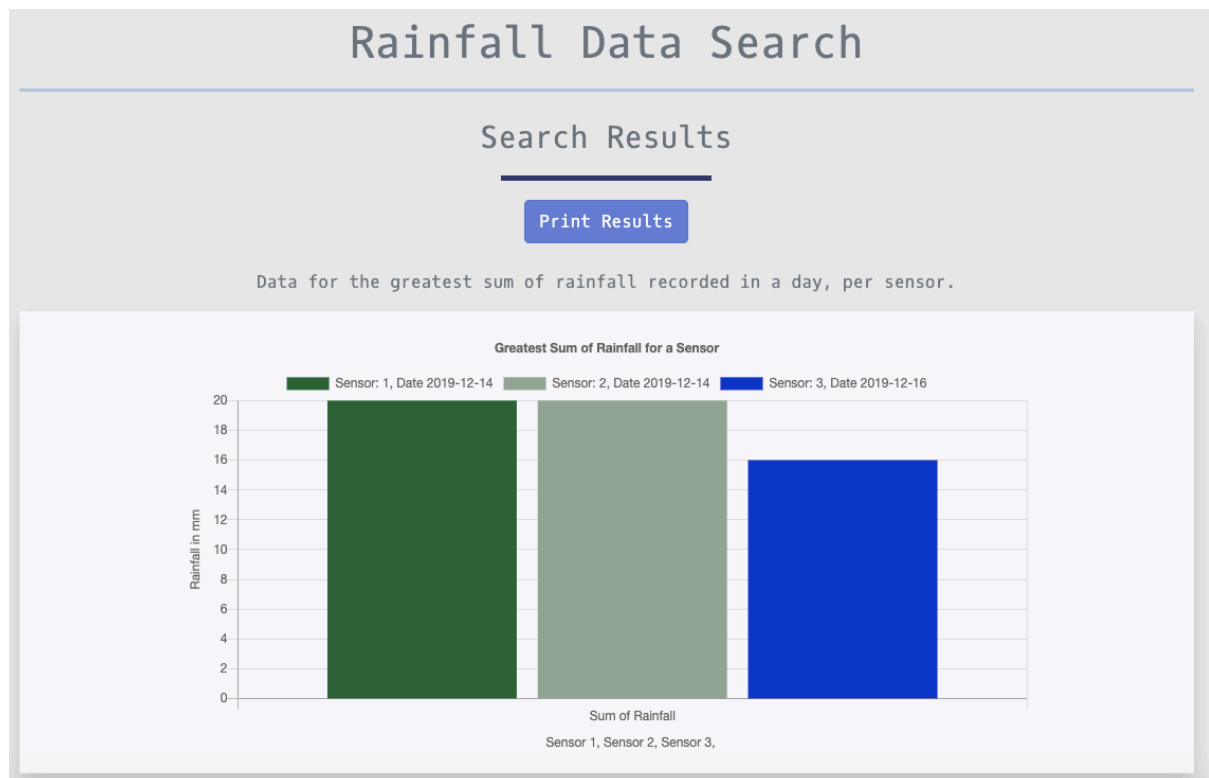


Figure 33. Result of Greatest Recorded Rainfall Search per sensor

### Greatest sum of rainfall recorded in a day, per sensor

Q Sensor 1

| Sensor ID | Sum of Rainfall Amount | Date Recorded |
|-----------|------------------------|---------------|
| 1         | 20                     | 2019-12-14    |

Q Sensor 2

| Sensor ID | Sum of Rainfall Amount | Date Recorded |
|-----------|------------------------|---------------|
| 2         | 20                     | 2019-12-14    |

Q Sensor 3

| Sensor ID | Sum of Rainfall Amount | Date Recorded |
|-----------|------------------------|---------------|
| 3         | 16                     | 2019-12-16    |

Figure 34. Table results from Greatest Recorded Rainfall Search

Figure 35. Build table data using PHP and the search result data

When a user logs in, they are directed to the dashboard which displays the most recent 5 results for each sensor.

In the following result (fig 36), all datetimes are identical, however logic has been implemented to account for differing datetimes, and so the x axis label would represent every unique date time. This applies for all rainfall sensor searches.

Annotated code documenting these technicalities can be found in the DashboardController index function (fig 37).

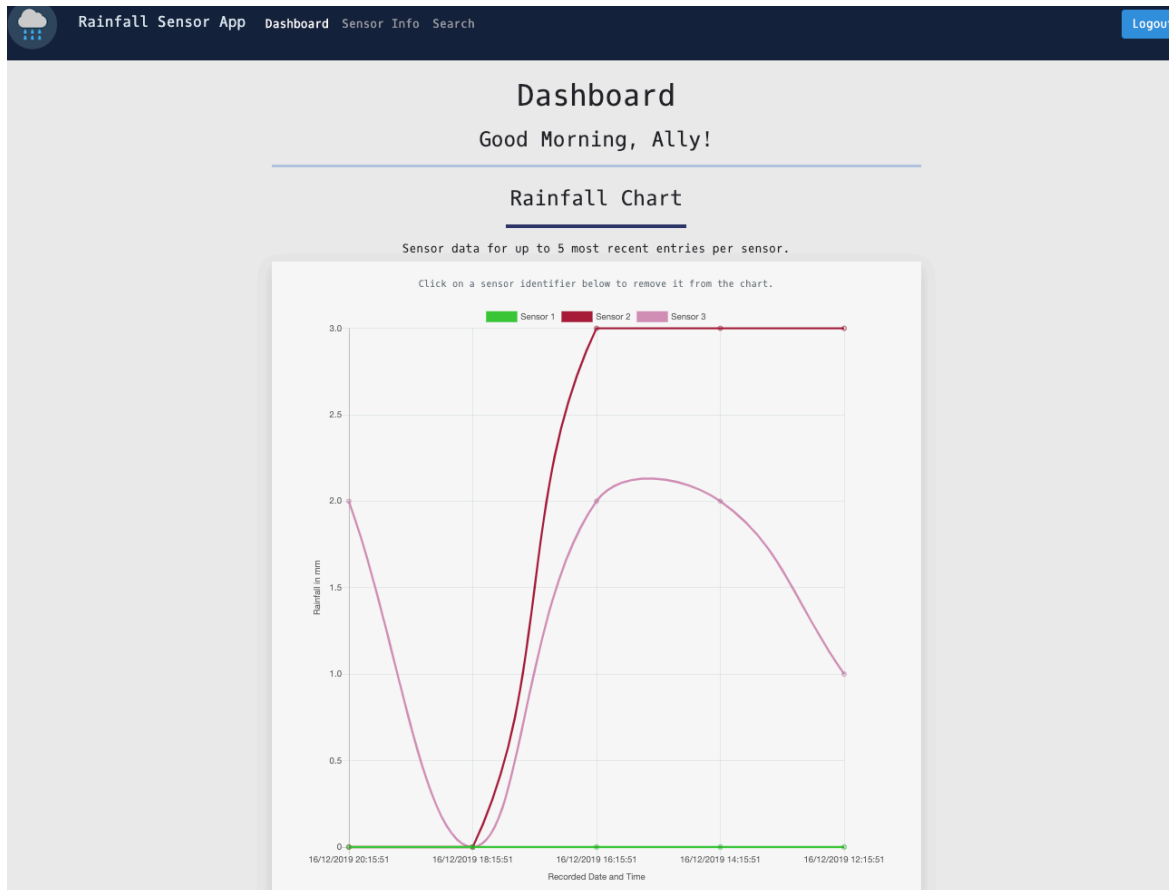


Figure 36. User dashboard, most recent 5 results for all sensors

```

foreach($sensor_id_list_res as $key){
    $sensor_id_list[$i] = $key->sensor_id;
    $i++;
}

$i = 0; //reset loop variable

/**
 * It has been noted that in the provided sensor data, the recorded datetime is the same across all sensors.
 * However, in an attempt to produce a more scalable solution, it must be assumed that the sensors record data at varying times.
 * Therefore, an array of dates must be produced which will later be used as an array of labels for the x axis of the combined chart
 *
 * The dates will be grouped to make an array of unique dates.
 */

foreach($sensor_id_list as $item) {
    /** Equivalent raw SQL for the following eloquent query:
     * SELECT * FROM sensor_data WHERE sensor_id = {$item} ORDER BY datetime_recorded DESC LIMIT 5; */
    $data_item_array = SensorData::where('sensor_id', $item)->orderBy('datetime_recorded', 'DESC')->take(5)->get();
    //check if there are any sensor data items for that sensor (a sensor could be newly installed and not have any recorded data).
    if(count($data_item_array) < 1){
        continue; //if there are no data items, don't write it to the sensor data item array. Skip this iteration of the foreach loop.
    } else {
        $sensor_data[$i] = $data_item_array; //if not empty, write it to the array that will be passed to the view
        //loop through each data item and grab the datetime_recorded for that item. Write it to an array.
        foreach ($sensor_data[$i] as $sensor_data_item) {
            $datetime_array[$j] = $sensor_data_item->datetime_recorded; //build an array of dates which will later be made unique

            //due to the fact that the sensor data datetimes are all the same in the given data, insert a datetime (sorted) to test the
            if($j == 2){ //comment out this if block unless for testing
                $j++;
                $dateobj = DateTime::createFromFormat('Y-m-d H:i:s', "2019-12-16 15:01:33");
                $datetime_array[$j] = Carbon::parse(date_format($dateobj, 'Y-m-d H:i:s'));
            }
            $j++; //increment loop variable
        }
        $i++;
    }
}

//group datetime array into unique datetimes
$datetime_array = array_unique($datetime_array);
//return the view with two separate arrays
//1 - the array of unique dates/times,
//2 - the full 3D array of sensors and their 5 most recent recorded data items (rainfall amounts)

$greeting = $this->build_greeting();

return view( view: 'layouts/dashboard' )->withData($sensor_data)->withDates($datetime_array)->withGreeting($greeting);

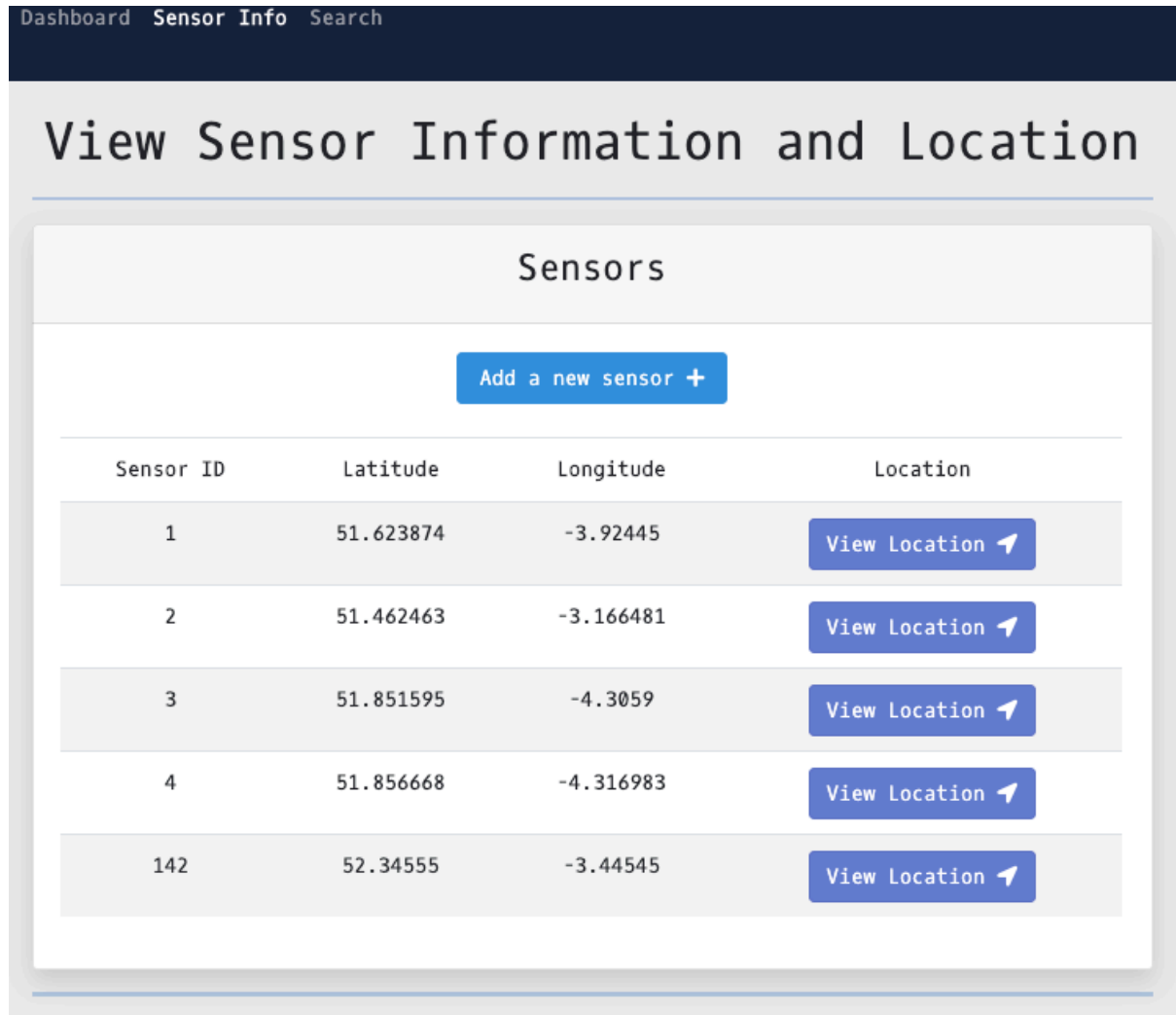
```

Figure 37. Snippet of annotated code from the DashboardController.

### 3.2.6 All Sensor Information

Further to the rainfall searches, sensor information has also been implemented.

The following table (fig 38) shows information on ALL sensors in the database. In the future, pagination should be implemented for an improved user experience.



| Sensor ID | Latitude  | Longitude | Location                        |
|-----------|-----------|-----------|---------------------------------|
| 1         | 51.623874 | -3.92445  | <a href="#">View Location ↗</a> |
| 2         | 51.462463 | -3.166481 | <a href="#">View Location ↗</a> |
| 3         | 51.851595 | -4.3059   | <a href="#">View Location ↗</a> |
| 4         | 51.856668 | -4.316983 | <a href="#">View Location ↗</a> |
| 142       | 52.34555  | -3.44545  | <a href="#">View Location ↗</a> |

Figure 38. Sensor info for all sensors. First 4 created using DB seeder.

### 3.2.7 Search for individual sensor

The individual sensor data can be accessed by clicking the table row from the previous search (fig 38) or from buttons in the table headers in rainfall searches.

The search uses URL query strings, where the request is routed to the SensorInformation controller search function, see Fig 39 below.

```
/**
 * Search is accessed via the GET action on the route: sensor-info/search?sensor={id}
 *
 * It returns the information for the given sensor, if exists.
 */
public function search(Request $request){
    //If the search parameter does exist.
    if($request->has( key: 'sensor')) {
        $sensor = $request->input( key: 'sensor');
        if($request->input( key: 'sensor') != ""){//check it isn't empty
            //check the sensor exists
            $sensorinfo = SensorInformation::where('sensor_id', $sensor)->get();

            if(count($sensorinfo)<1){//check that the query result is not null.
                $sensorinfo = null;
            }
            //set the last recorded event for that sensor
            //RAW SQL EQUIVALENT:
            //SELECT * from sensors WHERE sensor_id = '$sensor' ORDER BY datetime_recorded DESC LIMIT 1;
            $last_recorded_event = SensorData::where('sensor_id', $sensor)->orderBy('datetime_recorded','DESC')->limit(1)->get();

            if(count($last_recorded_event)<1){//check that the query result is not null.
                $last_recorded_event = null;
            }else{
                //always returns as 3d array so take first element (as we have it limited to 1 anyway)
                $last_recorded_event = $last_recorded_event[0]->datetime_recorded;
            }
            //return the data to the sensor-result view. Will still be under the same route.
            return view( view: 'layouts/sensor-result')->withData($sensorinfo[0])->withDate($last_recorded_event);
        }else{
            session()->flash('error','Unable to get sensor information for this sensor.');
```

Figure 39. Search for sensor using URL query string

The google maps API was also implemented (fig 40 and 41) by including their API script and unique API Key, and then dynamically building the location using php and the sensor's longitude and latitude.

rainfall-sensor-app.test/sensor-info/search?sensor=1

App Dashboard Sensor Info Search

## View Sensor Information and Location

### Sensor Information for Sensor 1

| Latitude  | Longitude | Date Installed | Last Recorded Event |
|-----------|-----------|----------------|---------------------|
| 51.623874 | -3.92445  | 05/10/2019     | 16/12/2019 20:15:51 |

[Edit sensor information](#)

### Location of Sensor 1 on Map



[Back to all sensors](#)

Figure 40. Sensor information result for sensor 1, note URL string and google maps

```

<div class="bg-light p-3 mt-2 mb-4 shadow">
  <h3 class="mt-4">Location of Sensor {{ $data->sensor_id }} on Map</h3>
  <hr class="subheading-bold-hr-blue"/>
  <div class="map mx-auto" id="map"></div>
  <!-- build the map using the sensor's longitude and latitude -->
  <script>
    var map;

    function initMap() {

      var latlng = new google.maps.LatLng({{ $data->location_latitude }},{{ $data->location_longitude }});

      map = new google.maps.Map(document.getElementById('map'), {
        center: {
          lat: {{ $data->location_latitude }},
          lng: {{ $data->location_longitude }},
          zoom: 15
        }
      });
      var marker = new google.maps.Marker({
        position: latlng,
        map: map,
        title: 'Sensor #1.'
      });
    }

    initMap();
  </script>
</div>

<hr class="bold-hr"/>
<div class="text-center">
  <a class="btn btn-info text-white mb-4 mt-2" href="/sensor-info"><i class="fas fa-undo"></i> Back to all sensors</a>
</div>
</div>

```

Figure 41. Building the Google Map using php and search data

To improve this search in the future, a town field could be added to the database using a migration, and then added to the table result. A field could be added to search for sensors within a given town, using wildcards.

## 3.3 Other Functionality

### 3.3.1 Create and Update Sensors

The update form (edit-sensor.blade.php fig 43) allows the user to edit sensor information. The form is Posted to the same URL, where it is routed by the web.php route (fig 42) to the update function in the controller (fig 44).

```
//Ensure that users are both authenticated AND have verified their email for
// all of the following routes.
Route::group( ['middleware' => 'verified','auth' ], function()
{
    //Route get requests for /home to the show function in the DashboardController (same as redirect to dashboard)
    Route::get('/home', 'DashboardController@show')->name('dashboard');

    //Route any requests for /process-files to the index function in the ProcessTextFileController
    Route::any('/process-files', 'ProcessTextFileController@index')->name('process-files');

    //Return the view text-file for get requests to /text-file as there is no dynamic content
    Route::get('/text-file', function () {
        return view( view: 'layouts/text-file');
    });

    //Route get requests for /dashboard to the show function in the DashboardController
    Route::get('/dashboard', 'DashboardController@show')->name('dashboard');

    //Route get requests for /edit-sensor/{sensor_id} to the get function in the SensorInformationController
    Route::get('/edit-sensor/{sensor_id}', 'SensorInformationController@get');

    //Route post requests for /edit-sensor to the update function in the SensorInformationController. Name this update-sensor.
    Route::post('/edit-sensor', 'SensorInformationController@update')->name('update-sensor');

    //Return the create-sensor view for get requests to /new-sensor.
    Route::get('/new-sensor', function () {
        return view( view: 'layouts/create_sensor');
    });

    //Route post requests for /new-sensor to the create function in the SensorInformationController. Name this new-sensor.
    Route::post('/new-sensor', 'SensorInformationController@create')->name('new-sensor');

    //Route get requests for /sensor-info/{sensor_id} to the search function in the SensorInformationController.
    Route::get('/sensor-info/{sensor_id}', 'SensorInformationController@search');

    //Route get requests for /sensor-info to the show function in the SensorInformationController. Name this sensor-info.
    Route::get('/sensor-info', 'SensorInformationController@show')->name('sensor-info');

    //Return the search-form view for get requests to /search-form - no dynamic content that requires the controller.
    Route::get('/search-form', function () {
        return view( view: 'layouts/search-form');
    });

    //Route post requests for /search-form to the search function in the SearchSensorDataController.
    Route::post('search-form', 'SearchSensorDataController@search');
});
```

Figure 42. web.php routes file

rainfall-sensor-app.test/edit-sensor/id?sensor=1

r App Dashboard Sensor Info Search

## Edit Sensor Information

Update Sensor Information

Location Latitude

Location Longitude

Date Installed

Figure 43. Edit sensor information, form accessed via URL Query string and a GET request.

```
/**
 * update allows the user to edit the sensor data.
 * @param Request $request
 * @return \Illuminate\Contracts\View\Factory|\Illuminate\Http\RedirectResponse|\Illuminate\View\View
 */
protected function update(Request $request)
{
    try {
        $date_now = date('Y-m-d'); //today's date, used later for comparison of dates for field validation

        //validate the input data
        $validated_data = $request->validate([
            'sensor_id' => 'bail|required|min:0|integer',
            'location_latitude' => 'bail|required|min:-90|max:90|between:-90.000000,90.000000',
            'location_longitude' => 'bail|required|min:-180|max:180|between:-180.000000,180.000000',
            'date_installed' => 'bail|required|date|before:' . $date_now . '|after:2000-01-01'
        ]);

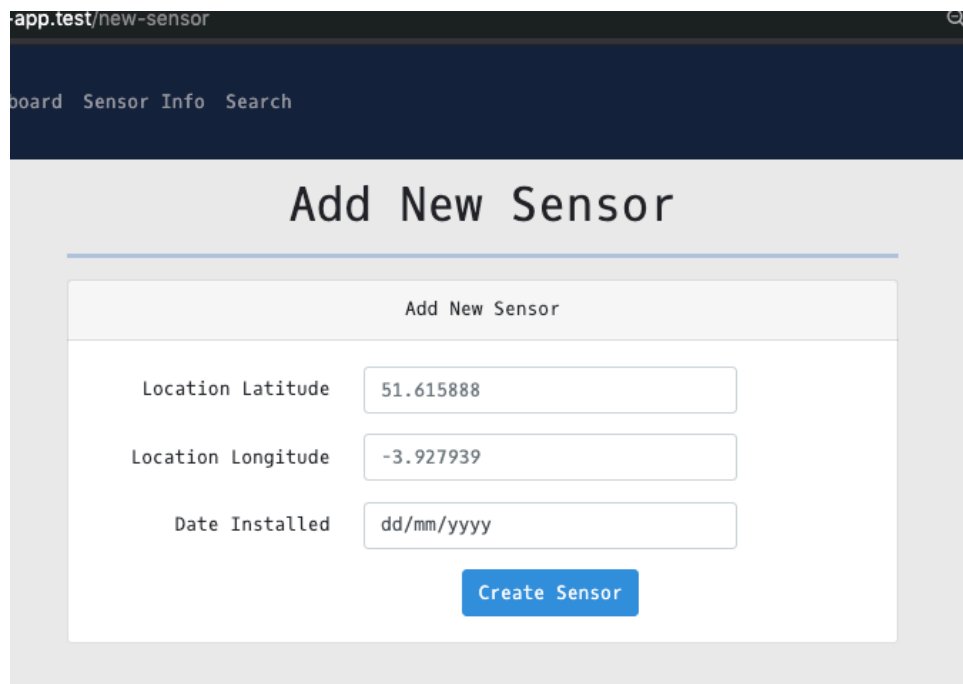
        //get the sensor id for the sensor to be updated and create a new eloquent instance of sensor
        $sensor = SensorInformation::find($validated_data['sensor_id']);
        //set the values to be the new updated values
        $sensor->location_latitude = $validated_data['location_latitude'];
        $sensor->location_longitude = $validated_data['location_longitude'];
        $sensor->date_installed = $validated_data['date_installed'];
        //save to the database
        $sensor->save();

        /**
         * RAW SQL Equivalent of update:
         *
         * UPDATE sensors SET location_latitude = $validated_data['location_latitude'],
         * location_longitude = $validated_data['location_longitude'],
         * date_installed = $validated_data['date_installed']
         * WHERE sensor_id = $sensor->sensor_id;
         */

        //show success message to use and route back to sensor information page (list all sensors)
        session()->flash('success', 'Success, Sensor information has been updated.');
```

Figure 44. Update sensor code in controller, note validation of fields and success/error messages set for user

New sensors can be created by Posting to the create function in the controller, and the form is validated similarly to others previously described.



The screenshot displays a web browser window with the address bar showing 'app.test/new-sensor'. The page has a dark blue header with navigation links: 'board', 'Sensor Info', and 'Search'. The main content area is titled 'Add New Sensor' in a large, bold font. Below the title is a form titled 'Add New Sensor' with three input fields: 'Location Latitude' with the value '51.615888', 'Location Longitude' with the value '-3.927939', and 'Date Installed' with the placeholder 'dd/mm/yyyy'. A blue 'Create Sensor' button is positioned at the bottom of the form.

| Add New Sensor                 |            |
|--------------------------------|------------|
| Location Latitude              | 51.615888  |
| Location Longitude             | -3.927939  |
| Date Installed                 | dd/mm/yyyy |
| <button>Create Sensor</button> |            |

Figure 45. Create new sensor form

## 4. Reflection

Overall, the web application produced has been a success and provides a good user experience and functionality whilst meeting the requirements of the application.

### 4.1 Technologies Used

It is felt that the technologies used were suitable for the application, however, a faster/non-throttled database would be beneficial to improve response times. Another area of improvement would be the use of an automated script to process documents and write them to the database, as described in this report. This would allow the system to stay up to date and remain more scalable. Alternatively, implement the file processing asynchronously using AJAX, so that the page load is not delayed, and the user not left waiting.

The functionality to upload files without access to the S3 console was not implemented, and so an improvement would be to allow vendors to log into a sub-domain of the application with unique credentials and upload the files directly from the site to the S3 bucket.

The Laravel framework was a learning curve for the author however has now allowed her to develop new skills. In the future, the author would continue to use Laravel and its inbuilt methods due to their ease of use, security considerations and robustness.

### 4.2 Working Practices

Throughout the project good working practices have been used, from following the documentation to little and often commits to source control as recommended.

A future improvement would be to practice commenting more during the development process, so that code does not have to be re-visited. It would also be very preferable to have code reviews from peers on a push to the repository, so that any uncaught issues could be spotted by them.

### 4.3 Searches Implemented

With the main requirement of the application to make the visualisation of rainfall-sensor data more easily understandable, it was important that a range of searches be included to query that data.

It is felt that despite having four searches covering different areas, it would be beneficial to have implemented other searches, such as the sum of rainfall per month, grouped per sensor, and also the sum of rainfall per day for the last 7 days, per sensor, which would've been ideal for the dashboard data. A wider range of graphs such as a scatter graphs could also be implemented.

## 5. References

- [1] "Laravel - The PHP Framework For Web Artisans." [Online]. Available: <https://laravel.com/docs/5.8/installation>. [Accessed: 25-Feb-2020].
- [2] "ClearDB MySQL - Add-ons - Heroku Elements." [Online]. Available: <https://elements.heroku.com/addons/cleardb>. [Accessed: 25-Feb-2020].
- [3] "Eloquent: Laravel Framework." [Online]. Available: <https://laravel.com/docs/5.8/eloquent>. [Accessed: 25-Feb-2020].
- [4] "AWS SDK for PHP." [Online]. Available: <https://aws.amazon.com/sdk-for-php/>. [Accessed: 25-Feb-2020].

## 6. Appendix

### 6.1 *Appendix A*

#### **Laravel PHP Framework Benefits**

Laravel is an open-source PHP web application framework that simplifies many of the processes of building web apps, including authentication, sessions, routing and caching. It provides database integrations, migrations and seeders along with unit testing to build and test a robust application.

Laravel uses the blade templating engine to simplify views and include dynamic content using predefined blocks, whilst also allowing the use of plain php. All blade code is later compiled into plain php. It also incorporates Composer, which is a tool that includes all the dependencies and libraries required for the application and allows the easy installation of third-party libraries. All dependencies are noted in composer.json file which is placed in the source folder [1].

The Laravel framework provides numerous built in libraries and modules which helps in enhancement of the application. Every module is integrated with the Composer dependency manager which manages updates.

Laravel allows the user to define flexible routes, whilst allowing the application to scale and maintain its high performance.

Two of the most useful features are session management and user authentication. Laravel comes with flows including registration, forgot password and send password reminders, to allow the developer to focus on the core functionality of the application. In addition to this it also provides a mail class which aids in sending mail from the web application.

Finally, Laravel provides a simple, efficient and consistent approach to managing the configuration of different environments [2] through the use of config and .env files.

#### **Local Development Environment**

The application has been developed locally using Homestead, a pre-packaged Vagrant box that provides a development environment without requiring the user to install PHP, a web server, and any other server software on their local machine [3].

#### **Eloquent ORM and Query Builder**

Eloquent is Laravel's inbuilt query builder which provides ORM (Object Relational Mapping) and ActiveRecord implementation [4], which aids in the querying of data by using multiple chained methods. Each table has a corresponding Model which is used to interact with that table.

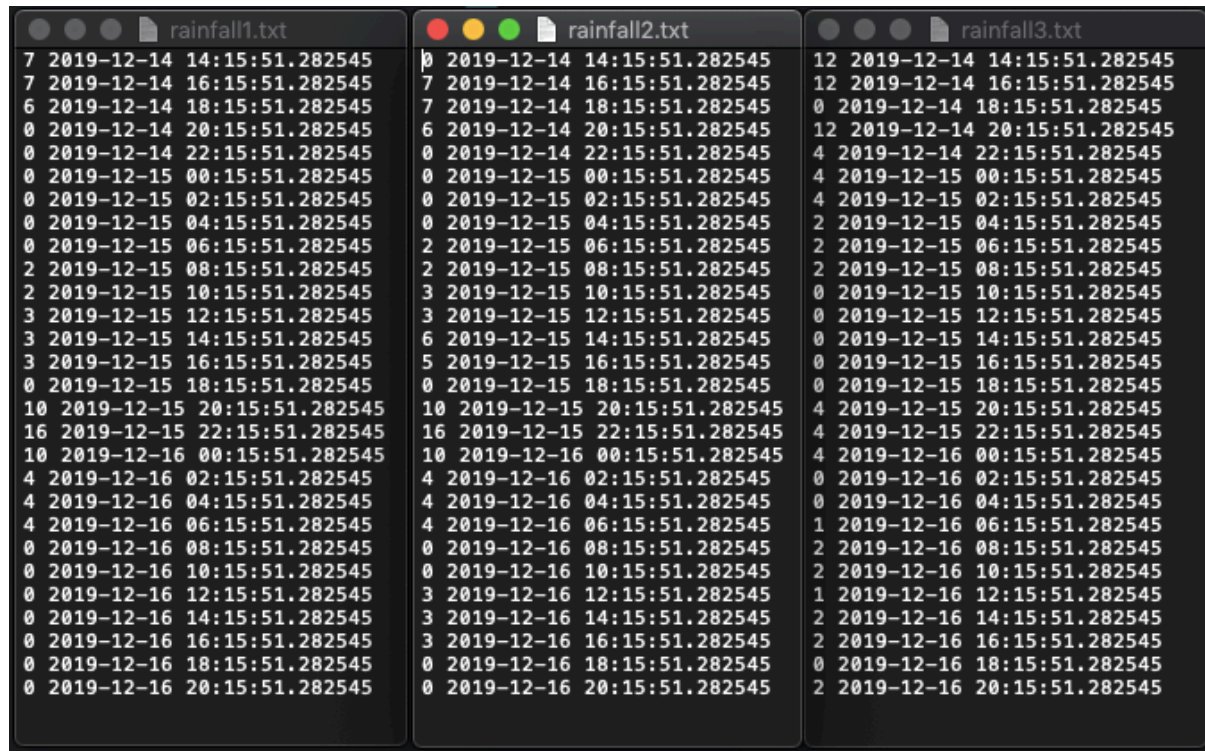
The query builder has inbuilt security protection against SQL injection, which can be used in place of raw SQL.

- [1]: "Laravel Overview." [Online]. Available: [https://www.tutorialspoint.com/laravel/laravel\\_overview.htm](https://www.tutorialspoint.com/laravel/laravel_overview.htm). [Accessed: 25-Feb-2020].
- [2]: "Laravel Introduction." [Online]. Available: <https://www.w3schools.in/laravel-tutorial/intro/>. [Accessed: 25-Feb-2020].
- [3]: "Laravel Homestead." [Online]. Available: <https://laravel.com/docs/5.8/homestead>. [Accessed: 25-Feb-2020].
- [4]: "Eloquent: Laravel Framework." [Online]. Available: <https://laravel.com/docs/5.8/eloquent>. [Accessed: 25-Feb-2020].

## 6.2 Appendix B

### Database Considerations and Design

The main requirement of the application was to store rainfall sensor data which would later be displayed the user in multiple formats, along with providing security measures and user accounts.



| rainfall1.txt                 | rainfall2.txt                 | rainfall3.txt                 |
|-------------------------------|-------------------------------|-------------------------------|
| 7 2019-12-14 14:15:51.282545  | 0 2019-12-14 14:15:51.282545  | 12 2019-12-14 14:15:51.282545 |
| 7 2019-12-14 16:15:51.282545  | 7 2019-12-14 16:15:51.282545  | 12 2019-12-14 16:15:51.282545 |
| 6 2019-12-14 18:15:51.282545  | 7 2019-12-14 18:15:51.282545  | 0 2019-12-14 18:15:51.282545  |
| 0 2019-12-14 20:15:51.282545  | 6 2019-12-14 20:15:51.282545  | 12 2019-12-14 20:15:51.282545 |
| 0 2019-12-14 22:15:51.282545  | 0 2019-12-14 22:15:51.282545  | 4 2019-12-14 22:15:51.282545  |
| 0 2019-12-15 00:15:51.282545  | 0 2019-12-15 00:15:51.282545  | 4 2019-12-15 00:15:51.282545  |
| 0 2019-12-15 02:15:51.282545  | 0 2019-12-15 02:15:51.282545  | 4 2019-12-15 02:15:51.282545  |
| 0 2019-12-15 04:15:51.282545  | 0 2019-12-15 04:15:51.282545  | 2 2019-12-15 04:15:51.282545  |
| 0 2019-12-15 06:15:51.282545  | 2 2019-12-15 06:15:51.282545  | 2 2019-12-15 06:15:51.282545  |
| 2 2019-12-15 08:15:51.282545  | 2 2019-12-15 08:15:51.282545  | 2 2019-12-15 08:15:51.282545  |
| 2 2019-12-15 10:15:51.282545  | 3 2019-12-15 10:15:51.282545  | 0 2019-12-15 10:15:51.282545  |
| 3 2019-12-15 12:15:51.282545  | 3 2019-12-15 12:15:51.282545  | 0 2019-12-15 12:15:51.282545  |
| 3 2019-12-15 14:15:51.282545  | 6 2019-12-15 14:15:51.282545  | 0 2019-12-15 14:15:51.282545  |
| 3 2019-12-15 16:15:51.282545  | 5 2019-12-15 16:15:51.282545  | 0 2019-12-15 16:15:51.282545  |
| 0 2019-12-15 18:15:51.282545  | 0 2019-12-15 18:15:51.282545  | 0 2019-12-15 18:15:51.282545  |
| 10 2019-12-15 20:15:51.282545 | 10 2019-12-15 20:15:51.282545 | 4 2019-12-15 20:15:51.282545  |
| 16 2019-12-15 22:15:51.282545 | 16 2019-12-15 22:15:51.282545 | 4 2019-12-15 22:15:51.282545  |
| 10 2019-12-16 00:15:51.282545 | 10 2019-12-16 00:15:51.282545 | 4 2019-12-16 00:15:51.282545  |
| 4 2019-12-16 02:15:51.282545  | 4 2019-12-16 02:15:51.282545  | 0 2019-12-16 02:15:51.282545  |
| 4 2019-12-16 04:15:51.282545  | 4 2019-12-16 04:15:51.282545  | 0 2019-12-16 04:15:51.282545  |
| 4 2019-12-16 06:15:51.282545  | 4 2019-12-16 06:15:51.282545  | 1 2019-12-16 06:15:51.282545  |
| 0 2019-12-16 08:15:51.282545  | 0 2019-12-16 08:15:51.282545  | 2 2019-12-16 08:15:51.282545  |
| 0 2019-12-16 10:15:51.282545  | 0 2019-12-16 10:15:51.282545  | 2 2019-12-16 10:15:51.282545  |
| 0 2019-12-16 12:15:51.282545  | 3 2019-12-16 12:15:51.282545  | 1 2019-12-16 12:15:51.282545  |
| 0 2019-12-16 14:15:51.282545  | 3 2019-12-16 14:15:51.282545  | 2 2019-12-16 14:15:51.282545  |
| 0 2019-12-16 16:15:51.282545  | 3 2019-12-16 16:15:51.282545  | 2 2019-12-16 16:15:51.282545  |
| 0 2019-12-16 18:15:51.282545  | 0 2019-12-16 18:15:51.282545  | 0 2019-12-16 18:15:51.282545  |
| 0 2019-12-16 20:15:51.282545  | 0 2019-12-16 20:15:51.282545  | 2 2019-12-16 20:15:51.282545  |

Figure 1. Provided data for sensors (rainfall1.txt, rainfall2.txt, rainfall3.txt)

After consideration of the provided rainfall1.txt, rainfall2.txt and rainfall3.txt files and their data format (shown above in figure 1), the following was noted:

1. Each row contains the following -
  - a. [{rainfall in mm} {date (Y-m-d) recorded} {time recorded to 6 fractional seconds}].
2. Each new row is a new data item.
3. The data in each row is split using a SPACE ' '.
4. The data in each file relates to a single sensor.

General assumptions made for this application:

1. The number in the text file name relates to the ID of the sensor (database ID).
2. The fractional seconds to 6 decimal places are not a critical aspect, as the data on average is 2 hours apart. This could be shortened in the database to the standard H:i:s format.
3. Despite rainfall1, rainfall2, and rainfall3 recording sensor data at the same times (and time intervals), this would not always be the case – for example due to sensor trigger delays from internet connection, or damage to a sensor.

4. To make the application more realistic to a production product, where multiple vendors may be in control of the sensors, the sensor data should be uploaded to a single centralised, easy to access location.
5. Users must be allowed to create their own user accounts through the use of a signup and will have access to their account after verifying their email address.
6. Only administrator users will have the rights to process the text files.
7. Only a single database will be required for both the development and staging environments.

As a result of the above, the following database design was created (figure 2).

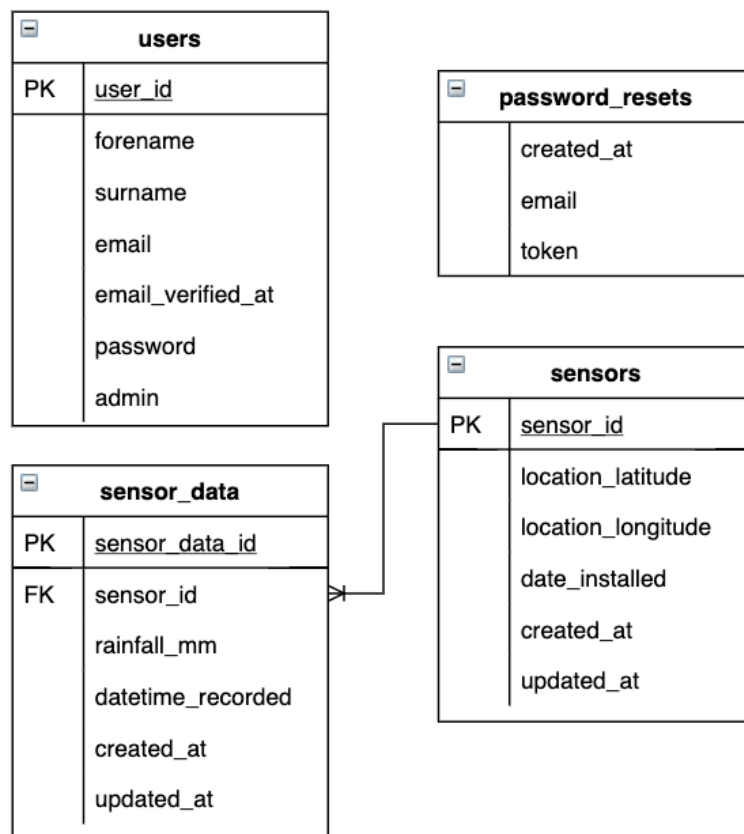


Figure 2. Entity-Relation Diagram of Database Structure

The database was defined and created using database migrations and the Eloquent query builder. Whilst investigating the database design a limitation of Laravel was noted – it is unable to store fractional seconds when using the Eloquent builder.

As a result, a standard datetime field was to be used. Other field lengths were considered, such as those stored in the sensors table, more specifically the longitude and latitude values. Latitudes range from -90 degrees to +90 degrees and therefore an appropriate data type/size is DECIMAL(10,8) whereas Longitudes range from -180 degrees to +180 degrees and so the data type/size is DECIMAL(11,8).

An example database migration for the creation of the sensor\_data table is shown below in figure 3.

```

public function up()
{
    /**
     * Raw SQL of create sensor_data table:
     * CREATE TABLE `sensor_data` (
     *   `sensor_data_id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
     *   `sensor_id` bigint(20) unsigned NOT NULL,
     *   `rainfall_mm` int(11) NOT NULL,
     *   `datetime_recorded` datetime NOT NULL,
     *   `created_at` timestamp NULL DEFAULT NULL,
     *   `updated_at` timestamp NULL DEFAULT NULL,
     *   PRIMARY KEY (`sensor_data_id`),
     *   KEY `sensor_data_sensor_id_foreign` (`sensor_id`),
     *   CONSTRAINT `sensor_data_sensor_id_foreign` FOREIGN KEY (`sensor_id`) REFERENCES `sensors` (`sensor_id`)
     *)
     */
    Schema::create( table: 'sensor_data', function (Blueprint $table) {
        $table->bigIncrements( column: 'sensor_data_id');
        $table->unsignedBigInteger( column: 'sensor_id');
        $table->integer( column: 'rainfall_mm');
        $table->dateTime( column: 'datetime_recorded');
        $table->timestamps();
        $table->foreign( columns: 'sensor_id')->references( columns: 'sensor_id')->on( table: 'sensors');
    });
}

```

Figure 3. CreateSensorTable migration to create the sensors table with annotated code and corresponding raw SQL equivalents.

During the development process the author had to recreate the database on clearDB, which was made much simpler through the use of migrations. A simple artisan command was run, 'php artisan migrate', which ran the migrations in time order to recreate the tables as shown in figure 4.

In production applications, database migrations are essential to provide a source control of changes to the database. Any alterations to the table structure should be completed through timestamped migrations so they can be applied to databases in other environments, such as altering a dev database and later making the change to the staging database.

```

SS-MBA:rainfall-sensor-app siobhan$ php artisan migrate
Migration table created successfully.
Migrating: 2020_02_04_102236_create_users_table
Migrated: 2020_02_04_102236_create_users_table (0.41 seconds)
Migrating: 2020_02_04_103522_create_password_resets_table
Migrated: 2020_02_04_103522_create_password_resets_table (0.37 seconds)
Migrating: 2020_02_04_105532_create_sensor_table
Migrated: 2020_02_04_105532_create_sensor_table (0.18 seconds)
Migrating: 2020_02_04_135340_create_sensor_data_table
Migrated: 2020_02_04_135340_create_sensor_data_table (0.38 seconds)
SS-MBA:rainfall-sensor-app siobhan$

```

Figure 4. CLI Output from running the database migrations

A database seeder was also created for the sensors table, so that test sensor data could be inserted in order to test the file processing. The code for this can be seen below in figure 5.

```

public function run()
{
    //I could later use a For Loop to iterate through random seeds,
    // however for this application I want to create 4 specific sensors.
    DB::table('sensors')->insert([
        [
            'location_latitude' => 51.61588800,
            'location_longitude' => -3.92793900,
            'date_installed' => '2019-11-01'
        ],
        [
            'location_latitude' => 51.46246300,
            'location_longitude' => -3.16648100,
            'date_installed' => '2019-11-22'
        ],
        [
            'location_latitude' => 51.85159500,
            'location_longitude' => -4.30590000,
            'date_installed' => '2019-12-01'
        ],
        [
            'location_latitude' => 51.85666800,
            'location_longitude' => -4.31698300,
            'date_installed' => '2020-02-02'
        ]
    ]);
}

```

Figure 5. SensorsTable Seeder code to create four specific sensors

### Database Table Definitions

The following screenshots show the table definitions from within MySQL Workbench.

Name: **password\_resets** Schema: heroku\_b8c95babb

| Column     | Datatype     | PK                       | NN                                  | UQ                       | BIN                      | UN                       | ZF                       | AI                       | G                        | Default / Expression |
|------------|--------------|--------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|----------------------|
| email      | VARCHAR(65)  | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |                      |
| token      | VARCHAR(191) | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |                      |
| created_at | TIMESTAMP    | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | NULL                 |

Figure 6. password\_resets table definition

password\_resets - Table x sensor\_data - Table x Query 6 x

Name: **sensor\_data** Schema: heroku\_b8c95ba

| Column          | Datatype   | PK                                  | NN                                  | UQ                       | BIN                      | UN                                  | ZF                       | AI                                  | G                        | Default / Expression |
|-----------------|------------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|----------------------|
| sensor_data_id  | BIGINT(20) | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |                      |
| sensor_id       | BIGINT(20) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| rainfall_mm     | INT(11)    | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| datetime_rec... | DATETIME   | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| created_at      | TIMESTAMP  | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| updated_at      | TIMESTAMP  | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |

Figure7. sensor\_data table definition

| <div> <div>sensor_data - Table</div> <div>sensors - Table</div> <div>Query 7</div> </div> |              |                                     |                                     |                          |                          |                                     |                          |                                     |                          |                      |
|-------------------------------------------------------------------------------------------|--------------|-------------------------------------|-------------------------------------|--------------------------|--------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|----------------------|
| Name: <input type="text" value="sensors"/>                                                |              | Schema: heroku_b8c95ba              |                                     |                          |                          |                                     |                          |                                     |                          |                      |
| Column                                                                                    | Datatype     | PK                                  | NN                                  | UQ                       | BIN                      | UN                                  | ZF                       | AI                                  | G                        | Default / Expression |
| sensor_id                                                                                 | BIGINT(20)   | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |                      |
| location_latit...                                                                         | DOUBLE(10,8) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| location_longi...                                                                         | DOUBLE(11,8) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| date_installed                                                                            | DATE         | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| created_at                                                                                | TIMESTAMP    | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| updated_at                                                                                | TIMESTAMP    | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| <click to edit>                                                                           |              | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |

Figure 7. sensors table definition

| <div> <div>sensors - Table</div> <div>users - Table</div> <div>Query 7</div> </div> |              |                                     |                                     |                                     |                          |                                     |                          |                                     |                          |                      |
|-------------------------------------------------------------------------------------|--------------|-------------------------------------|-------------------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|----------------------|
| Name: <input type="text" value="users"/>                                            |              | Schema: heroku_b8c95ba              |                                     |                                     |                          |                                     |                          |                                     |                          |                      |
| Column                                                                              | Datatype     | PK                                  | NN                                  | UQ                                  | BIN                      | UN                                  | ZF                       | AI                                  | G                        | Default / Expression |
| user_id                                                                             | BIGINT(20)   | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |                      |
| forename                                                                            | VARCHAR(191) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| surname                                                                             | VARCHAR(191) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| email                                                                               | VARCHAR(191) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| email_verified...                                                                   | TIMESTAMP    | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| password                                                                            | VARCHAR(191) | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |
| admin                                                                               | TINYINT(1)   | <input type="checkbox"/>            | <input checked="" type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | '0'                  |
| remember_to...                                                                      | VARCHAR(100) | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| created_at                                                                          | TIMESTAMP    | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| updated_at                                                                          | TIMESTAMP    | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | NULL                 |
| <click to edit>                                                                     |              | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> | <input type="checkbox"/>            | <input type="checkbox"/> |                      |

Figure 8. users table definition

Note, the length 191 was set for fields where a specific length was not set. The default value is read from the *AppServiceProvider.php* file, shown below.

```
Builder::defaultStringLength(191); // Update defaultStringLength
```

## 6.3 *Appendix C*

### **Heroku**

Heroku is a fully managed cloud platform-as-a-service (PaaS) that allows developers to increase productivity and focus on building applications and deploying code without maintaining servers or infrastructure, and to easily deploy, manage and scale modern apps [1].

Heroku allows for simple integration with GitHub, allowing for automatic deployments to the staging application based on a trigger event, such as a push or merge to a repository [2].

[1]: “About Heroku | Heroku.” [Online]. Available: <https://www.heroku.com/about>. [Accessed: 25-Feb-2020].

[2]: “GitHub Integration (Heroku GitHub Deploys) | Heroku Dev Center.” [Online]. Available: <https://devcenter.heroku.com/articles/github-integration>. [Accessed: 25-Feb-2020].

## 6.4 *Appendix F*

### **User Management, Authentication and Authorisation**

The inbuilt Laravel Auth methods have been used within the application to provide a secure and simple way for users to register, login and recover their account. This enabled the developer to focus on building the core functionality of the application without reinventing the wheel – and is able to provide a greater level of security thanks to its widespread usage and testing, where any security flaws are quickly fixed. Annotated code demonstrating how user authentication and session management could have been implemented in plain PHP has been included in the `app/Http/Controllers/Auth/LoginController` file.

Users may sign up for an account using a HTML form with a POST action to the `Auth/RegisterController`. Any user can register for an account and will gain access to the system once their email address has been verified.

Laravel uses it's inbuilt mail methods and a provided Gmail account to send the emails [1].

Users may also request a password reset email, where a unique signed url is sent to the user that directs them to the reset password view, and updates the database with the new password.

Laravel manages the sessions for users, carrying user data throughout the application using Session variables, and provides a logout function which destroys the current session when required [1].

[1]: “Laravel Authentication.” [Online]. Available:  
<https://laravel.com/docs/5.8/authentication>. [Accessed: 25-Feb-2020].

## 6.5 *Appendix D*

### **Chart.js and JpGraph**

Open-source Chart.js provides an attractive and simple way to build charts using JavaScript. It provides transitions, animations, mixed charts and filtering out of the box.

Chart.js provides 8 different base chart types, with the most common being pie, bar and line graph, each of them being animated and customisable [1]. Along with this, chart.js is responsive, and redraws charts when windows are resized.

JpGraph provides many more chart types than chart.js such as spider-plots, accumulated bar plots, grouped bar plots etc however are built using PHP [2]. An image is generated on the server in a specific size where it is returned to the client, meaning images cannot be resized upwards if quality is to be retained. As JpGraph provides static images, it lacks the user engagement compared to the JavaScript used by chart.js.

[1]: "Chart.js | Open source HTML5 Charts for your website." [Online]. Available: <https://www.chartjs.org/>. [Accessed: 25-Feb-2020].

[2]: "JpGraph - Most powerful PHP-driven charts." [Online]. Available: <https://jpgraph.net/>. [Accessed: 25-Feb-2020].

## 6.6 Appendix E

### Amazon Web Services Simple Storage Service (S3)

The AWS PHP SDK is used to access S3, with the SDK being provided access key credentials for an IAM user with access to the bucket contents through a .env file. The bucket contents are not publicly accessible [1].

AWS provides a free-tier for very low usage, along with 99.99% availability, aiding to the applications robustness. It encrypts data during transport and at rest to ensure the data remains secure.

S3 allows the use of presigned URLs to access and upload files without providing a user with security credentials or implementing access keys and programmatic access through the SDK.

rainfall-sensor-data-files

Overview Properties Permissions Management Access points

Q Type a prefix and press Enter to search. Press ESC to clear.

Upload Create folder Download Actions

US East (N. Virginia)

Viewing 1 to 2

| Name          | Last modified                    | Size    | Storage class |
|---------------|----------------------------------|---------|---------------|
| rainfall3.txt | Feb 28, 2020 2:20:58 PM GMT+0000 | 841.0 B | Standard      |
| rainfall7.txt | Feb 25, 2020 6:07:26 PM GMT+0000 | 29.0 B  | Standard      |

Viewing 1 to 2

Figure 1. S3 Management Console showing the bucket and 2 example files

[1]: “AWS SDK for PHP.” [Online]. Available: <https://aws.amazon.com/sdk-for-php/>. [Accessed: 25-Feb-2020].