## 

horizontal line

**Mono Apartments - Technical Documentation v1.0**

17 May 2023

—

Prepared for Modern Nomads Group Pty Ltd

Yudo Baskaro, Director of the company

Team OneTen

# Table of Contents

[**Table of Contents 2**](#_e696l5oxm9x2)

[**1. Introduction 3**](#_osqwez9q1s9n)

[1.1. Purpose 3](#_w315f8a8z9fq)

[1.2. System Skill Sets 3](#_ed61uif2pu)

[**2. Deployment to a New Server 4**](#_wxzc3zbw304q)

[2.1. Development Environment 4](#_mwoe0a5t2guj)

[2.2. Windows 4](#_iikf0xpai32o)

[2.3. MacOS 5](#_ay8izvi42xd)

[2.4. SQL Database and Schema 6](#_lgbdn0404omv)

[2.4.1. SQL Database Schema and Example Data 6](#_svpwt6544xqt)

[2.4.2. MySQL Database Structure 7](#_6av1mdocv1p)

[2.5. Cake PHP 8](#_ugjvvvf1y9th)

[2.6. GIT Repository 8](#_ecgee0j0xd4k)

[2.7. Launching Server 9](#_gurf9vxk8uyl)

[2.8. Backup and Recovery 10](#_vbl2hgsbdnpv)

[2.8.1. Implementing Server Backup 10](#_zd3tigludhq5)

[2.8.2. System Recovery 10](#_sq7q71kyg1io)

[2.9. Making code updates 11](#_lsi2fmkjzwar)

[**3. Authentication & Authorization 12**](#_p0uwedwo69z)

[3.1. Authentication 12](#_jt7o4x8pjhsi)

[3.2. Login Function in Authentication 12](#_a56qc0onkw56)

[3.3. Authorization 13](#_424rrupkcjk0)

[**4. Codebase structure 13**](#_v1xazr20z9a2)

[4.1. User Controller 13](#_j1aq7tej2pbn)

[4.2. Apartment Controller 14](#_fq3ec32x5jan)

[4.3. Email Controller 15](#_6qb5u3z2jwap)

[4.4. Inspection Controller 17](#_r4pr6e88ezk)

[4.5. Image Controller 18](#_5dgco7d1dbke)

[**5. System e-mail management 19**](#_euwjbdpa5qmy)

[**6. CKEditor 20**](#_n4kly8lczp2l)

[6.1. CKEditor functions list 20](#_1c9yp7i99mls)

[**7. Design styles 22**](#_eoy89k6flcdp)

[7.1. Theme 22](#_efscco16mgs5)

[7.2. Error page 22](#_91a65ya282n0)

# 

# Introduction

Mono Apartments is a real estate company located in Melbourne that offers a variety of rental properties in convenient locations throughout Melbourne. Their primary goal is to provide a simple and stress-free experience for clients looking to relocate or find a new home. The company employs local experts to help clients find the best possible housing arrangements based on their unique needs. Mono Apartments goes above and beyond by offering personalised services, such as advice on location, amenities, lease terms, and rental rates. Their commitment to excellent customer service and environmentally friendly building practices ensures a top-notch living experience for all tenants.

### Purpose

The technical document is a set of instructions and resources for the technical users at Mono Apartments on how to use a new system that helps manage property inspections for contractors. The document will detail all technical aspects of the system ranging from the system basics to the development environment and deployment. The user document will also include all the necessary internal/external resources and information about the system and provide troubleshooting tips in case any issues arise. The goal of the technical document is to help technical users at Mono Apartments deploy and allow Mono Admins to use the new system effectively and have a positive experience with it.

### System Skill Sets

There are skills required to continue development on the Mono Apartments inspections system, these skills include: Cake PHP, PHP, HTML, CSS, JavaScript, MySQL and MariaDB.

# Deployment to a New Server

### Development Environment

The Mono Apartments Auditing System was developed using the [PHPStorm](https://www.jetbrains.com/phpstorm/) IDE(Build 231.8770.68). The open-source web framework used was [CakePHP(v4.4.8)](https://cakephp.org/), with coding in [PHP(v8.2.6)](https://www.php.net/)

### Windows

**Windows Setup Steps:**

1. Download a web server solution stack to run an Apache HTTP Server and the MySQL database. For example, one is XAMPP 8.2.4 (64-bit). Download link: <https://www.apachefriends.org/download.html>.
2. Open the downloaded web server solution and follow the instructions of the download wizard to properly install PHP and MySQL onto your Windows PC.
3. Install PHP composer using the ‘Composer-Setup.exe’ download link and follow the setup wizard and the instructions found at <https://getcomposer.org/download/>.
4. Set up version control for Windows such as Git. Download link: <https://git-scm.com/download/win>.
5. To start your local web server with XAMPP, open the software, and start Apache and MySQL in the control panel.
6. Enter the Apache config on the panel and select the php.ini file, here enable the intl extension by searching for ‘intl’ then uncomment the line ‘extension=intl’ by removing the semi-colon.
7. To set up the MySQL database open the ‘localhost/phpmyadmin’ directory in a browser, here you need to create the project database and the user account with permissions to access it.
8. Then you need to edit the ‘config/app\_local’ file located in the project directory and add the database configuration that was set.
9. Then the project can be run from the local Windows device by going to the localhost/ directory in a browser.

### MacOS

**MacOS Setup Steps:**

1. Install Homebrew (if not already installed):

* Open Terminal (Applications > Utilities > Terminal).
* Paste the following command and press Enter:

| /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)" |
| --- |

1. Install PHP:

* In Terminal, run the following command to install PHP using Homebrew:

| brew install php |
| --- |

1. Install a web server (e.g., Apache or Nginx):

* For Apache, run the following command in Terminal:

| brew install httpd |
| --- |

* For PHPmyadmin, run the following command in Terminal:

| brew install mariadb phpmyadmin |
| --- |

1. Configure the web server:

* For Apache, open the host configuration file:

| sudo nano /opt/homebrew/etc/httpd/httpd.conf |
| --- |

* Inside the file:
* Uncomment the following code in configuration:

| LoadModule rewrite\_module lib/httpd/modules/mod\_rewrite.so |
| --- |

* Add the following code in configuration:

| LoadModule php\_module /opt/homebrew/opt/php/lib/httpd/modules/libphp.so  <FilesMatch \.php$>SetHandler application/x-httpd-php</FilesMatch> |
| --- |

* In the following code, the commented part is the original code that needs to be located and replaced:

| #AllowOverride None  AllowOverride All  #<IfModule dir\_module>DirectoryIndex index.html</IfModule>  <IfModule dir\_module>DirectoryIndex index.php index.html</IfModule> |
| --- |

1. Restart the Apache server

| brew services restart httpd |
| --- |

1. Install Composer:

* In the Terminal, run the following command to install Composer:

| brew install composer |
| --- |

### SQL Database and Schema

#### 2.4.1. SQL Database Schema and Example Data

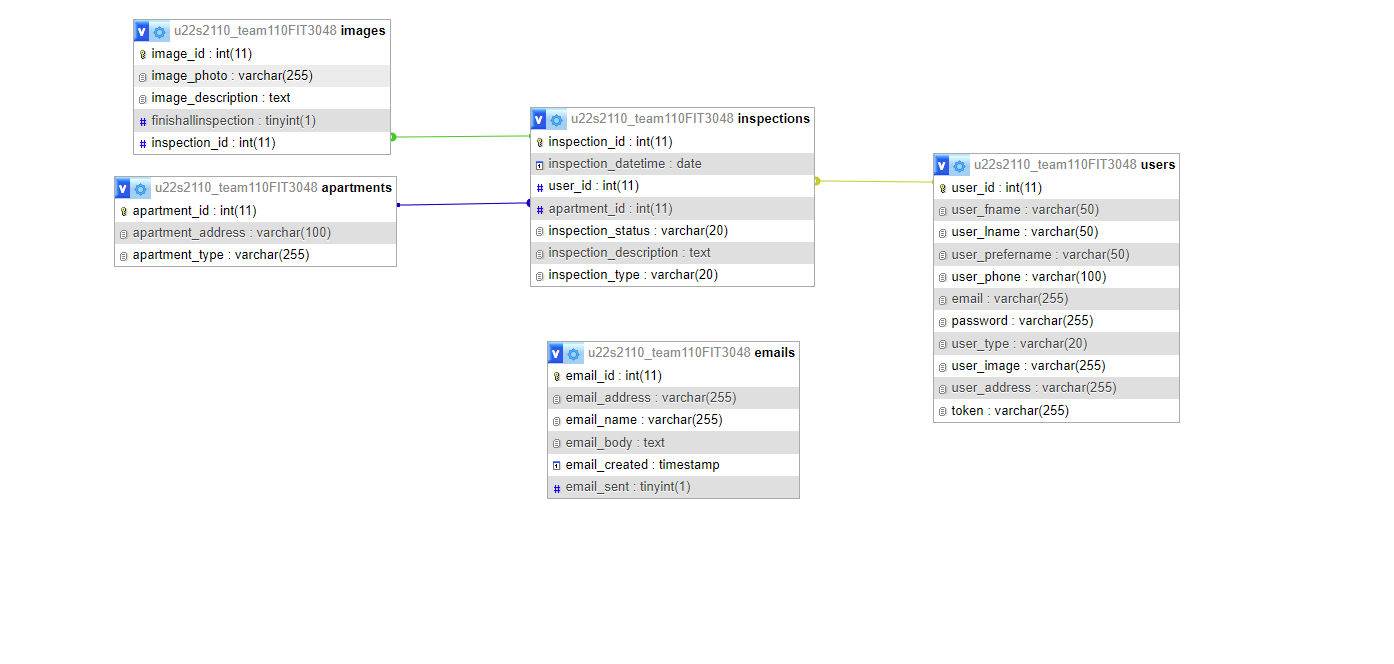
Within the GitHub project, you will find SQL files that contain the schema and example data for the system's database. To set up the database structure, start by importing the ‘Database.sql’ schema file into your database. This file defines the database structure and includes example data that demonstrates the system's functionality. Setting up the schema is necessary when configuring a local environment and when deploying the database to a new web server.

#### 2.4.2. MySQL Database Structure

The website relies on a MySQL database for its functionality. Within this project, there are 5 entity tables: inspections, images, users, apartments, and emails. All of these tables are interconnected except for the email table.

1. **Inspections**: This table serves as the main entity and has relationships with other tables. It has a one-to-many relationship with the images table, indicating that multiple images can be associated with each inspection.
2. **Images**: This table is a sub-table of the inspections table, representing a one-to-many relationship. Each inspection can have multiple images associated with it.
3. **Users**: The inspections table has a one-to-many relationship with the users' table. This means that multiple inspections can be linked to each user.
4. **Apartments**: Similar to the user's table, the inspection table also has a one-to-many relationship with the table of the apartment. Multiple inspections can be associated with each apartment.
5. **Emails**: The email table is independent and not directly connected to the other tables.

For a visual representation of the database structure and relationships, refer to the Entity Relationship Diagram (ERD) provided below.



### 

### Cake PHP

CakePHP is a PHP framework that is open-source and offers a comprehensive set of tools for web application development. It stands out from other PHP frameworks due to its enhanced clarity, which aids developers in understanding it well. The following features highlight the advantages of CakePHP:

* Rapid Development: CakePHP automates the creation of common functions and parameter definitions, enabling efficient programming.
* Robust Programming Support: CakePHP provides professional backup websites, offering developers a vast resource for finding solutions and assistance. For more information please [click here](https://book.cakephp.org/4/en/index.html).
* Simplified Configuration: The self-generating nature of CakePHP simplifies comprehension and programming for developers. Furthermore, the use of functions in CakePHP is restricted to controllers, making it easier to locate and address bugs.
* Security: CakePHP incorporates built-in safety functions, equipping developers with multiple mechanisms for maintaining a secure environment. These include data validation, authentication, authorization, and more.

CakePHP's combination of rapid development, strong programming support, simplified configuration, and built-in security measures contributes to its appeal among developers seeking an efficient and reliable PHP framework.

### GIT Repository

Another part of this system is the GIT Repository, which is extremely important in allowing multiple people to develop the system simultaneously. GIT Repository is a distributed version control which means each user can have complete control of their copy and work independently without relying on a central server.

The following are commonly used git statements:

| git init  git pull --set-upstream origin master  git add .  git commit -m “initial commit”  git push |
| --- |

For more git help, please [click here](https://about.gitlab.com/handbook/).

With this link here:< [UGIE / UGIE-2022 / team110 / team110-app\_fit3048 · GitLab (monash.edu)](https://git.infotech.monash.edu/UGIE/ugie-2022/team110/team110-app_fit3048) > you can find a link to the new cloned Git repository that has been created for the Mono team to use going forward. It contains all the commits and comments created by the previous development team and also located in the README.md file is all relevant system information.

### Launching Server

To set up a new website using your projects control terminal, you need to clone the Git repository, the code needed to do this is as follows:

| git clone <git-repository-URL> |
| --- |

After this there should be a folder named after the Git repository project and this will be the website’s root directory folder.

This is the initial setup process after which every time you require to update the server the command to execute an update is as follows:

| git pull |
| --- |

After cloning the repository an install of the composer will need to be run for the project to ensure that it runs properly. The command for that is as follows:

| composer install |
| --- |

Next is the setup of the database, modifying the ‘config/app\_local.php’ to match the credentials of the database found in phpMyAdmin.

Finally setting up the domain, using the ‘Domains’ section of the control panel. Using this to create a subdomain or using the main domain. Then setting up the document root folder of the intended domain, as the project folder. In the URL box by navigating to your domain name the website should be online and ready to use.

### Backup and Recovery

#### 2.8.1. Implementing Server Backup

To ensure data integrity and facilitate system recovery, it is highly recommended to implement a reliable backup method for your server: [Example here](https://au.pcmag.com/web-hosting/86026/how-to-back-up-your-website). For this purpose, it is suggested to utilise cPanel's built-in backup functionalities available in the 'Backup' section. The cPanel offers various backup types, including:

1. Full Website Backup: This option allows you to create a comprehensive backup of your entire website including all files, databases, and configurations.
2. Home Directory Backup: With this backup type, you can selectively back up specific directories or files within your server's home directory.
3. SQL Backup: This option enables you to back up your databases, ensuring that critical data stored within them is protected.

Partial backups are particularly useful as they are smaller in size compared to full backups, resulting in shorter download times.

#### 2.8.2. System Recovery

In the event of a system issue or the need to revert to a more stable build, cPanel's 'Backup' section also provides recovery options. You can use these options to initiate a rollback and restore your system to a previous state. Additionally, this feature allows you to recover any potentially lost data.

By following these backup and recovery procedures, you can effectively safeguard your server's data and quickly restore it if necessary.

Please note that the specific steps for performing backups and recoveries may vary slightly depending on your cPanel configuration. Consult the cPanel documentation or contact your hosting provider for detailed instructions tailored to your setup.

### Making code updates

When any changes need to be made to the codebase of the system, follow the steps below:

1. Open the system folder in PHPStorm: Launch PHPStorm and navigate to the folder containing your system's codebase. This can be done by selecting ‘Open’ from the File menu and choosing the appropriate directory.
2. Make the necessary code changes: Edit the relevant files in PHPStorm to implement the required updates. Ensure that you thoroughly test the system after making these changes to ensure optimal functionality.
3. Push changes to the Git repository: It is considered best practice to version control your code using Git and push the updated changes to a repository. PHPStorm simplifies this process by providing built-in Git tools.
   * Locate the Git section: The Git section can be found near the top of the toolbar in PHPStorm.
   * Commit your changes: Before pushing the changes, it is recommended to commit your changes. This allows you to create a logical checkpoint in your version history. Use the Git tools in PHPStorm to stage the modified files, add a descriptive commit message, and create a commit.
   * Push to the Git repository: Once you have committed your changes, you can push them to the Git repository. The push operation uploads your local changes to the remote repository, ensuring that your updates are safely stored and shared with other team members.

By following these steps, you can efficiently make code updates, maintain version control, and collaborate effectively with your team using PHPStorm's integrated Git tools.

Note: Familiarity with Git concepts and workflows is essential for using these Git tools effectively. If you are new to Git, it is recommended to learn the basics of Git version control. Various online resources and tutorials are available to help you get started with Git.

# Authentication **& Authorization**

### 3.1. Authentication

In this system, CakePHP's Authentication for authentication purposes is used. Each part of the system has its separate Controller. The Authentication is loaded within the initialize() function of the controllers. The beforeFilter() function ensures session management by checking if a user is logged in and if the logged-in user still exists before each action is executed.

During the current stage, the admin and contractor roles follow a unified login and authentication process. Both roles are treated as common users lacking differentiation in terms of functionality and access rights. By leveraging the capabilities of CakePHP's authentication, role-based access control can be used, allowing us to tailor the user experience specifically for admins and contractors. Within the UsersController the login actions for both the Admin and Contractor portals are shared. However, they are distinguished based on the user's 'role', determined by the 'user\_type' field in their identity. This distinction enables us to redirect users based on their specific viewing needs as either admins or contractors.

### 3.2. Login Function in Authentication

The login() method is called inside the UsersController when an admin submits the login form. Login details are stored in the user's database using the password (varchar(255)) and email (varchar(255)) attributes.

To ensure secure password storage, CakePHP's DefaultPasswordHasher is used. The password hashing process is handled by the \_setPassword() function located in the Entity/User.php file. This function hashes the password before storing it in the database, providing an additional layer of security for users' passwords.

The template for the admin login page can be found in the templates folder under Users/login.php. Furthermore, the user's role determines the authorization level for the pages that are accessible within the admin dashboard. This ensures proper separation of permissions between admins and contractors.

### 3.3. Authorization

In this system, authorization plays a vital role in maintaining data integrity, protecting sensitive information, and preventing unauthorised access to restricted functionalities.

**Authorization policy**: An authorization policy is a PHP class that encapsulates the rules and logic for determining whether a user is authorised to perform a specific action on a resource. These policies provide a centralised and reusable way to define access control rules across the application.

To define an authorization policy, a policy class is created that extends the Authorization\Policy\Policy class provided by CakePHP. Within the policy class, they define methods that correspond to different actions on a resource, such as view(), create(), update(), or delete(). This can be customised to the logic to determine whether a user is authorised to perform the specified action on the resource.

CakePHP's authorization component automatically applies the corresponding policy method based on the requested action and resource, making the process of enforcing access control seamless and straightforward.

# **Codebase structure**

### 4.1. User Controller

* The controller contains several functions for handling different user-related operations: index, view, add, edit, delete, beforeFilter, login, logout, forgot password and reset password.
* The index method retrieves a list of users from the database and paginates the results to display a manageable subset at a time.
* The view method fetches the specific details of a user based on the provided $id parameter and renders the corresponding view.
* The add method facilitates the addition of a new user. It expects a POST request with form data, validates the input, and saves the user to the database if the validation passes.
* The edit method handles the updating of an existing user. It retrieves the user based on the provided $id, validates the submitted form data, and saves the changes if the validation is successful.
* The delete method allows the deletion of a user. It requires a POST request and deletes the user with the given $id.
* The beforeFilter method acts as a callback executed before any action method. It performs necessary setup tasks, such as configuring the login action to not require authentication and determining the layout based on the user's role.
* The login method manages user authentication. It utilises the Authentication component to validate user credentials and redirects them to a specified URL upon successful login.
* The logout method logs out the currently authenticated user by invoking the logout method of the Authentication component.
* The forgot password method handles the logic for generating a password reset token, sending it to the user's email, and displaying appropriate messages.
* The reset password method allows users to reset their passwords using a valid reset token. It verifies the token's validity and updates the user's password if the submitted form data is valid.
* The reset password method allows users to reset their passwords using a valid reset token. It verifies the token's validity and updates the user's password if the submitted form data is valid.

### 4.2. Apartment Controller

* beforeFilter() method: This method is invoked before any operation is executed. It configures the login action to bypass authentication, preventing infinite redirect loop issues. It also checks the user's role and sets the view's layout accordingly, distinguishing between administrators and contractors.
* index() method: Responsible for displaying a list of apartments. It initialises a new empty apartment entity, authorises it, and retrieves a paginated list of apartments from the database. The list is then passed to the view for rendering.
* view($id) method: Displays the details of a specific apartment. It fetches the apartment with the provided $id from the database, authorises it, and passes it to the view for rendering.
* add() method: Handles the creation of new apartments. It creates a new empty apartment entity, authorises it, and if the HTTP request is a POST request, it populates the entity with the request data, saves it to the database, and redirects to the index page. If the save operation fails, an error message is displayed.
* edit($id) method: Manages the editing of existing apartments. It retrieves the apartment with the specified $id from the database and authorises it. If the HTTP request is a PATCH, POST, or PUT request, it updates the apartment with the provided data, saves the changes to the database, and redirects to the edited apartment's view page. In case of a failed save operation, an error message is displayed.
* delete($id) method: Deals with the deletion of apartments. Only POST and DELETE requests are allowed. It retrieves the apartment with the given $id from the database, authorises it, deletes it from the database, and redirects to the index page. If the deletion process fails, an error message is displayed.

### 4.3. Email Controller

* index(): This method is responsible for fetching the list of emails and rendering the corresponding view. It paginates the emails and passes them to the view.
* view($id): This method retrieves a specific email by its ID and renders the corresponding view. The skipAuthorization() method indicates to bypass the authorization check for this operation.
* add(): This method handles the creation of new emails. It initialises a new email entity, authorises the email, and retrieves a list of users with the "admin" user type. If the request is a POST request, it saves the email entity, sends the email using the Mailer component, and redirects to the "Inspections" index action.
* contractorask($id): This method is used to email inspection-related questions. It retrieves information related to users, checks, and apartments. Similar to the add() method, it saves the email entity and sends the email using the Mailer component.
* adminadd(): This method is similar to the add() method, but is intended to be created by an administrator. It saves the email entity, sends the email using the Mailer component, and redirects to the "Inspections" indexing action.
* mark($id): This method marks the email as sent by setting the email\_sent field to true. If the email has already been marked as sent, an error message will be displayed.
* delete($id): This method deletes the email by its id.
* beforeFilter($event): This method executes before any controller actions. It loads the necessary models: "User", "Check" and "Apartment".
* send email to all contractors(): This method sends an email notification to all contractors. It retrieves users with a "Contractor" user type and sends each user an email.
* send email to admin(): This method sends an email notification to all administrators. It retrieves users with an "admin" user type and sends each user an email.
* edit(): This method sends an email notification to the user (specified by user id) when the admin edits the check.
* inspected(): This method sends an email notification to all administrators when a property is inspected.
* reject(): This method is used to send rejection mail for inspection. It retrieves information related to users, inspections and apartments and sends emails accordingly.

### 4.4. Inspection Controller

* beforeFilter: This method is executed before any actions in the controller. It checks the user's role and sets the appropriate layout for the view based on the role. If the user is an administrator, the default layout is used, otherwise, the contractor\_default layout is used.
* index: This method retrieves the inspection list from the Inspections table and related data from the Users, Apartments, and Images tables. Then pass the check to the view.
* view: This method retrieves details for a specific check based on the provided ID. It also retrieves associated data from the Users, Apartments, and Images tables. Checks and related data are passed to the view.
* add: This method is used to add a new check. It creates a new empty Inspection entity and authorises it. If the request is a POST request, this method will attempt to save the inspection data. If successful, it redirects to the "sendemailtoallcontractor" action of the "email" controller. Otherwise, it displays an error message.
* edit: This method is used to edit an existing inspection. It retrieves checks based on the provided ID and authorises them. If the request is a PATCH, POST, or PUT request, the method attempts to update the check data. If successful, it displays a success message and redirects to the "Edit" action of the "Email" controller (if the user is an admin), otherwise redirects to the "View" action of the current controller. If the update fails, an error message will be displayed.
* delete: This method is used to delete the check. It retrieves the check based on the provided ID, authorises it and attempts to delete it. If successful, a success message will be displayed. Otherwise, an error message will be displayed. The method then redirects to the "index" action of the current controller.
* accept: This method is used for acceptance checks. It retrieves the check based on the provided ID, skips the authorization, and updates the check status to "Accepted". If the update is successful, a success message is displayed and the method redirects to the "sendemail admin" action of the "Email" controller. If the update fails, an error message will be displayed.
* complete: This method is used to mark the check as checked. It retrieves the check based on the provided ID, skips authorization, and updates the check status to "checked". If the update is successful, a success message is displayed and the method redirects to the Checked action of the Email controller. If the update fails, an error message will be displayed.
* reject: This method is used to reject the check. It retrieves the check based on the provided ID, authorises it, and updates the check status to "rejected". If the update is successful, a success message is displayed and the method redirects to the "Reject" action of the "Email" controller, passing the user ID and check ID as query parameters. If the update fails, an error message will be displayed.
* The finish method is responsible for marking an inspection as finished. It receives an optional parameter $id which represents the ID of the inspection to be finished. The method begins by retrieving the inspection entity from the database using the provided ID. The authorised method is then called to check if the current user is authorised to perform the finish action on the inspection.

### 4.5. Image Controller

* beforeFilter(): This method is executed before any controller actions are called. It checks the user's role (administrator or contractor) based on who the user is and sets the layout accordingly.
* index(): This method retrieves a list of images and their associated checks. It authorises the user to access the image and sets the retrieved data to the view.
* view($id): This method displays a specific image and its associated inspection based on the $id provided. It authorises the user to access the image and sets the retrieved data to the view.
* add(): This method handles the addition of new images. It authorises users to add images and handle submitted form data. It moves the uploaded image file to the target path and saves the image record. If successful, it redirects to the indexing operation. Also, if the form data contains a certain condition (finish all inspections), it redirects to the send mail admin action in EmailsController.
* edit($id): This method handles the editing of an existing image. It authorises users to edit images and process submitted form data. If the image record is successfully updated, it redirects to the indexing operation.
* delete($id): This method handles the deletion of the image. It authorised users to delete images and delete the corresponding records. If successful, it redirects to the indexing operation.

# **System e-mail management**

System email management on cPanel is a set of features for managing system-level email settings and configurations on Mono’s configured server. It provides a variety of tools and options for managing server settings, security, restrictions, and logging of sent and received emails.

The following is common functionality system Email management used:

* Email Filtering: this method can help users to filter spam and mail.
* Authentication: Used to configure the server to determine user permissions through email authentication settings.
* MX Entry: Allows you to specify the priority and configuration of the mail server. You can add, edit, and delete MX records to ensure that emails are properly routed to the server.

The email account that has been used is:

| 'username' => 'team110@production.u22s2110.monash-ie.me',  'password' => 'team110team110',  'client' => null,  'tls' => true,  'url' => env('EMAIL\_TRANSPORT\_DEFAULT\_URL', null), |
| --- |

| 'InspectionEmail' =>[  'to' =>'1',  'from' =>'team110@production.u22s2110.monash-ie.me'  ] |
| --- |

# CKEditor

This system, utilises the capabilities of CKEditor 5 to offer users an intuitive and robust content editing experience. CKEditor 5 is a highly adaptable and versatile “What You See Is What You Get” (WYSIWYG) editor that empowers users to create and manage rich text content effortlessly.

One specific area where the system has implemented CKEditor 5 is in the inspection description feature. Contractors can leverage CKEditor 5 to modify their inspection descriptions, allowing them to effectively communicate the details of their inspections. With the editor's user-friendly interface, contractors can easily insert photos of the inspection and seamlessly create, edit, and format content as required.

To ensure the durability and accessibility of the content, the edited content is stored in an SQL database in HTML format. This format enables the preservation of the content's rich formatting, ensuring that the intended visual presentation is maintained.

A cloud-based solution has been utilised for handling image uploads. Each time a user opens CKEditor 5, the images associated with the content are dynamically loaded from the cloud. This approach minimises load times and allows users to conveniently access their images within the editor.

### 6.1. CKEditor functions list

1. Export PDF and Export Word: These buttons in the toolbar allow users to export the editor's content as PDF or Word documents.
2. Heading: This dropdown in the toolbar enables users to apply different heading styles to the selected text.
3. Bold, Italic, Strikethrough, Underline, Remove Format: These buttons in the toolbar provide basic text formatting options like bold, italic, strikethrough, underline, and removing formatting.
4. Bulleted List, Numbered List, Todo List: These buttons allow users to create bulleted lists, numbered lists, and to-do lists.
5. Outdent and Indent: These buttons help in adjusting the indentation level of text or list items.
6. Undo and Redo: These buttons allow users to undo or redo their actions.
7. Link: This button enables users to insert or edit hyperlinks in the text.
8. CKBox: This allows users to upload images and files with drag & drop, paste from a clipboard, and paste from Word.
9. Upload Image: This button allows users to upload images into the editor with CKBox. It is configured to use the CKEditor Cloud Services for image upload.
10. Block Quote: This button applies a block quote style to the selected text.
11. Insert Table: This button allows users to insert tables into the editor.
12. Media Embed: This button enables users to embed media content, such as videos or audio, into the editor.
13. HTML Embed: This button allows users to embed HTML content into the editor.
14. Font Size, Font Family, Font Color, Font Background Color, Highlight: These dropdowns in the toolbar provide options to change the font size, font family, font colour, font background colour, and highlight colour of the selected text.
15. Alignment: This dropdown in the toolbar offers options for aligning the text.
16. Horizontal Line, Page Break: These buttons insert a horizontal line or a page break at the current cursor position.

# **Design styles**

### 7.1. Theme

The entire theme is defined within the default.php and contractord\_default.php in templates/layout, these themes are set across both the contractor and admin side of the system. There are links to the bootstrap style and also interaction plugins set by the CSS and js files within the webroot directory.

### 7.2. Error page

The templates/Error layout handles the error page when an error has occurred in the system and handles 400 and 500 errors in the system, and there will be a link for the user to go back to the dashboard page. Modified using Bootstrap to match the theme of the system.

In summary, the design styles implemented in the inspection management system contribute to its usability and clarity. The consistent theme applied across the contractor and admin sides of the system creates a unified experience. The error page, tailored to match the system's theme, ensures contractors can quickly understand and recover from encountered errors. By focusing on these design aspects, the system offers an intuitive and visually pleasing environment for efficient inspection management.