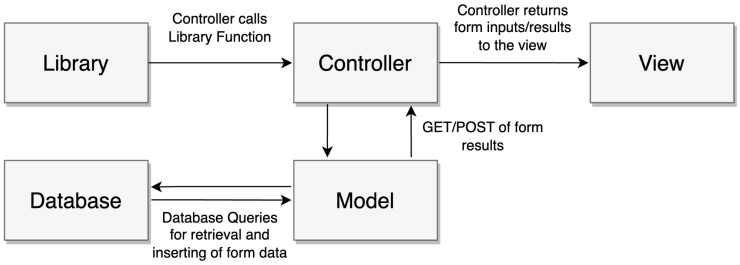
Model View Controller (MVC)



The Controller will communicate with the library to create functions such as form inputs with CSS styling, database manipulation and CRUD operations. The Model will act as the API to interact with the database. The CodeIgniter framework supports Active Record which natively uses MySQL and can help prevent Cross Site Scripting (XSS). If given the time and knowledge, we intend on using plugins like Doctrine or Eloquent. To demonstrate how programmers can use our library, 2 basic views will be created, the first view lists out different form types being created and previously created. The second view is an interactive form input with CSS styling. The number of views may change dynamically to illustrate and categorize the functionality provided in the library.

Function Calls

Our library aims to ease the process of form creation programmatically. Programmers should be able to call functions easily which aid the form building. HTTP, GET & POST will be used for retrieval and form data submission. The programmer will be able to perform basic CRUD operations on the forms they have created. Within the controller, they can easily create a variety of form inputs and labels. The examples are as follows:

- **Form Label Example**: label(string $name, string $textDisplay, string $additionalParams='')
    - Labels in HTML require a field name for the label being created and a text display to display the label to the user. Thus, 2 fields are required to create a label tag, but programmers can also include additional parameters that need to be in HTML syntax that will be appended to the HTML label tag if needed.
    - Function Usage Example: $form->label('name', 'Your name: '); or $form->label('name', 'Your name: ', ' class = "label" ');
    - Returns:
        - <label for="name">Your name: </label> or
        - <label for="name" class="label">Your name:</label>
- **Form Text Input Example**: text(string $name, string $inputId='', string $placeholder='', string $inputClass='', string $additionalParams='');
    - Text inputs in HTML only require a field name. The rest of the fields are for optional HTML data attributes which are frequently used by programmers. Additional parameters will also be accepted here.
    - Function Usage Example: $form->text('name');
    - Returns:
        - <input type="text" name="name" id="name">
- **Dropdown List Example**: select(string $name, array $inputArray, string $inputId='', string $inputClass='', string $default ='', string $additionalParams='');
    - Dropdown lists in HTML require a field name and an array of values to show for the form. The rest of the fields are optional HTML data attributes which are frequently used by programmers. Additional parameters will also be accepted here.
    - Function Usage Example: $form->select('name', $inputClass->'name');
    - Returns:
        - <select name="name" class="name">
            <option value="joshua">Joshua</option>
          </select>
    - Will support other types of inputs like checkboxes, radio buttons, etc.

  Additional function calls that our library will provide to speed up the development time and increase the scalability of our library.
- **Form Validation**
    - Data integrity, accuracy and consistency is of utmost importance to both the user and programmer. Programmers can use our library's simple form validation function call to perform basic sanitization on the data that the user has submitted and validate it according to the rules that the programmer has set.

    - $form->validate($string fields);
    - $dataToValidate = $form->validate('name|required|min[2]|max[32], phoneNumber|numeric, email');
        - returns a boolean value to our error handling function call on whether the user inputs are valid.
- **Pre-Built Forms**
    - Programmers can make use of this function to speed up the development process. Our library would offer a variety of common pre-built forms such as contact, login and feedback forms. Additionally, programmers can create custom forms by making use of our inbuilt form input function calls.
      For example:
        - $form->useCustomForm('contactUs'); # Pre-built Form
        - $form->useCustomForm('companyRecruitment'); # Custom Form
- **Error Handling**
    - Programmers should incorporate proper error handling practices to ensure the robustness and reliability of the code. Based on the results of the form validation function, errors can be displayed if a field did not pass the rules the programmer has defined which notifies users on the respective input fields that did not pass the rules of the form validation function call. For example, if a text input field called ‘name’ was not filled up by the user, it will return an error message to notify the user:
        - $form->in_errors('name', 'Please enter a valid name');
- **Debugging**
    - Programmers can make use of the print_r() or d() functions incorporated from PHP and CodeIgniter 4 to display information on the variables in layman terms. For example, $form->printr($var_a);

Library Architecture

Our library allows users to create various HTML input types, error handling for invalid inputs, security, debugging tools and ensuring integrity of data. The following properties that will be incorporated are:

- **Security**
    - Programmers may create forms which may retrieve sensitive information from the users. Hence, our library will conform to the OWASP top 10 framework to protect confidential form data from attacks such as SQL Injection or XSS.
- **Autoloading**
    - The implementation of autoloading with PSR-4 standards allows programmers to easily integrate our library into their projects, helping to load the necessary files automatically and simplifying the process of including classes.
- **Library Documentation**
    - Our library will include examples, usage scenarios and code snippets for the function calls that our library provides, allowing programmers to easily reference and understand how to make use of our library’s function calls.
- **Library Test Cases**
    - To shorten the debugging time for programmers, they can make use of our library’s test cases to ensure that the function calls are working as intended by producing the expected results.

Database Schema

We intend to implement a relational database (MySQL). The database schema implemented is designed to dynamically generate new forms without creating new entities or altering the database schema. PHP’s inbuilt data serialization capabilities are used to store form data in a TEXT format within the FormSubmissions table as seen in the diagram below. The diagram below is an example of our database schema in MySQL.