Coding Standard & Database Structure

Version	Created On
1.0	23/10/2022

Contents

1 Design

1.1 Website

As per the project requirement and design done by Designers.

1.2 Internal Apps

Only the below-mentioned CSS and JS have to be used (third-party design requires permission).

1.2.1 Designs to follow ffreedom internal apps

- new_apps:
    - https://dev.moneyadvice.in/admin_docs/new_apps_demo
- old_apps : existing design platform

2 Database

Here are some best practices for using MySQL:

- Use the correct data types for your columns. This will ensure that your data is stored efficiently and accurately.
- Use meaningful and descriptive names for your tables and columns. This will make it easier to write and understand your SQL queries.
- Use proper indexing to improve the performance of your queries. Indexes can help MySQL find and retrieve data quickly, especially when dealing with large tables.
- Use the appropriate storage engine for your tables. MySQL supports several storage engines, each with its own strengths and weaknesses. Choosing the right one can greatly improve the performance and reliability of your database.
- Normalize your data to avoid redundancy and ensure data integrity. Normalization involves organising your data into multiple related tables, which can help reduce duplication and make it easier to update and maintain your data.
- Use proper error handling and transaction management to ensure the reliability and integrity of your data. Transactions allow you to execute multiple SQL statements as a single unit, which can help prevent data inconsistencies in the case of an error or failure.
- Use prepared statements and parameterised queries to protect against SQL injection attacks. Prepared statements allow you to execute the same SQL

statement multiple times with different parameters, while protecting against malicious input.

- Appropriate default values must be given for each columns , if there is no default value , set the default value as NULL

2.1.Table Name

- should be in small letter
- separator is underscore (_)
- Database : Table Names
    - MDT- master data table
    - CDT- Common data table
    - SDT- static data table
    - TDT- transactional data table
    - LDT-log data table
    - AMDT- archived master data table
    - ACDT- archived Common data table
    - ATDT- archived transactional data table
    - ALDT-archived log data table

      All the tables should be categorized among the above listed table types and named accordingly. For eg. mdt_employees , ldt_user_punch

2.2.Column Name

- Should be in small letter
- Separator is underscore (_)
- Prefix with table abbreviator
- Keep same column name if any foreign key or relation exist with other tables
- Keep comment for all columns
- Data type and size should match with requirement
- Column name should not contain any reserved keywords.

3 Coding

3.1.General

- All the functions writing should contains date added added by/ modified by
  Eg: #### added_by:name added_on:date (edited_by name:date) #####
- Code should not be repeated while creating new application.
- Use common header and footer.
- All functions name should be readable english words(word_word) and should be related to exact functionality .
- Before developing anything, check existing pages to ensure that not already exist.
- Use Jquery standard for javascript.
- All ajax response preferred to be in json format with data status.
- All list pages should contains search filters and pagination(search should be in post).
- While updating Image replace if existing, instead of creating new image.
- Any customization required in existing template or platform should take proper permission from respective people.
- While using third party libraries,codes,or new language should be informed to everyone about the scope of that particular technology.
- Insert update should contains one method and one view page. don't create multiple functions/pages to achieve this.
- Don't use third party css in internal application use existing bootstrap template.
- Compress all images to improve performance.
- If you are creating any api - create it as template, so that it can be used in many places.
- Don't show any simple alert popup on update or insert action. show message.

3.2.Naming Convention

- Write all the functions related to a module in single controller and keep controller name same as module name. for eg. settings/add_page
- Controllers – begin with uppercase, words separated by underscore.
- All the actions related to a module should be kept in single controller. For eg. settings_actions/update_page
- Keep method name same as view page name inside the respective module.
- Models – begin with upper case and same as table name followed by module. For eg. Mdt_employees_model .php
- All the method names to be in lower case, words separated with under score.

- Views- each module will be having a folder named after the module. All pages related to a module are kept inside the respective folders. Folder names in lower case and page/view name in lower case as well, words separated with under score.

3.3.Security

Here are some key points to keep in mind:

- Don't put unnecessary echo,print_r,display query,exit.
- Use post method to submit form. Try not to use get and don't pass any variable in url.
  All file upload folders should contains index.php/htaccess.
- All Document upload paths should contains index.php and file names should be encrypted or non guessable format.
- Sessions should be handled carefully and it should not be compromise.
- All forms should contains all possible validations
- Data should be encrypted or masked.Consumer or associates Mobile number and email should not be displayed anywhere (script, display,ajax,popup etc).
- Don't give delete option anywhere - use status.

3.4.PHP

To follow proper PHP coding standards, you should follow the guidelines set out in the PHP Framework Interop Group (FIG) PSR-1 and PSR-2 documents. These documents outline a set of recommended coding standards and best practices for PHP code, including how to format your code and how to name your variables and functions.
Here are some key points to keep in mind:

- Use 4 spaces for indentation, not tabs
- Use meaningful and descriptive names for your variables and functions
- Use camelCase for variable and function names, and UPPERCASE_WITH_UNDERSCORES for constant names
- Always use <?php and ?> to delimit PHP code, and never leave any whitespace before or after the PHP tags
- Use <?= for short echo statements, but make sure to avoid using this syntax when you are outputting complex expressions or unsafe data
- Put each statement on its own line, and end each statement with a semicolon
- Use braces to denote blocks of code, even when they are not strictly required
- Use the elseif keyword instead of else if
- It's also a good idea to use a code linter, such as PHP_CodeSniffer, to automatically check your code for compliance with the PSR standards.
- Ref. Link : https://www.php-fig.org/psr/psr-2/

3.5.Codeigniter

Here are some best practices for using CodeIgniter:

- Use the MVC (Model-View-Controller) architecture. CodeIgniter is based on the MVC architecture, which separates the application logic from the presentation layer. This makes it easier to develop, maintain, and scale your web applications.
- Use the built-in libraries and helpers. CodeIgniter comes with a number of built-in libraries and helpers that provide common functions and features. Using these can save you time and effort, and help you avoid writing repetitive code.
- Use the built-in security features. CodeIgniter provides a number of security features, such as input filtering and XSS prevention, to help protect your web applications from common security vulnerabilities.
- Use the built-in caching mechanism. CodeIgniter has a built-in caching mechanism that allows you to store the output of your controllers in cache files. This can improve the performance of your web applications, especially when dealing with large or complex queries.
- Use the built-in error handling and logging features. CodeIgniter provides a number of error handling and logging features that can help you troubleshoot and debug your web applications.All modules contains 2 main controllers.
- Use Codeigniter Controller core class(MY_Controller) to validate session and authentication
- eg: TEST Module
  Hrm and Test_actions :
    - Test : All the view pages using $this->load->view('main',$data); should be in this controller
    - Test_actions: All other view pages and CRUD related actions should be in this action.
    - You can use new controller based on your requirement
      eg:Test_ajax (all related controllers name should be prefix with module name).

- Model should not be loaded in view page.
- Don't write html view part in controller.
- All Database related SQL queries should be in model.(should not be there in helper,controller,or view). Get all query result inside the module and return to the controller and then load the view.
- Use codeigniter active records to build queries (https://www.codeigniter.com/userguide2/database/active_record.html). For complex queries use $this->db->query($sql);
- Controller function name,view page name,model function name preferred to be as same.
- Select only required fields instead of selecting all fields.

- Use config values everywhere, instead of writing as hardcoded.(url,values,file upload and display paths).
- All ajax pages should contains 3rd parameter as TRUE to avoid html render. eg:$this->load->view('test_view.php',$data,true);
- Use existing custom templates, Dropdown,Pagination,Sms,Email,Status change, etc.
- Load models in currently required functions instead of loading everything in constructor function.
- All models name should be same as table name and ending with _model.first letter should be in capital letter for all classes, models and controllers.
- Get all query result inside the module and return to the controller and then load the view.

3.6.API

Here are some best practices for designing and using APIs:

- Use clear and descriptive names for your API endpoints and parameters. This will make it easier for developers to understand and use your API.
- Use standard HTTP methods (e.g. GET, POST, PUT, DELETE) for your API endpoints. This will make your API consistent with the RESTful architecture and other APIs.
- Use appropriate HTTP status codes to indicate the result of an API request. This will help developers understand the success or failure of a request and take appropriate action.
- Use pagination for long lists of data. This will help prevent your API from returning too much data at once, which can overload the server and the client.
- Use filtering and sorting options for long lists of data. This will allow developers to retrieve only the data they need and in the order they prefer.
- Use versioning for your API. This will allow you to make changes to your API without breaking existing applications that use it.
- Use authentication and authorisation to protect your API and its data. This will prevent unauthorised access to your API and its resources.

3.7.Redis

Here are some best practices for using Redis:

- Use the appropriate data type for your data. Redis supports several data types, including strings, lists, sets, and hashes. Choosing the right data type can help improve the performance and efficiency of your Redis database.
- Use pipeline and batch operations to reduce network overhead. Redis allows you to execute multiple commands in a single network round trip, using the pipeline or MULTI/EXEC commands. This can greatly improve the performance of your Redis applications, especially when dealing with large amounts of data.
- Use Redis transactions for atomic operations. Redis transactions allow you to execute multiple commands as a single atomic unit. This can help ensure the consistency and

integrity of your data, especially when multiple clients are updating the same data concurrently.

- Use Redis replication to improve availability and scalability. Redis supports master-slave replication, which allows you to have multiple copies of your data on different Redis servers. This can improve the availability and scalability of your Redis applications.
- Use Redis Sentinel to monitor and manage your Redis servers. Redis Sentinel is a distributed service that provides high availability and failover support for Redis. It can monitor your Redis servers and automatically failover to a slave in case the master goes down.

3.8.ElasticSearch

Here are some best practices for using Elasticsearch:

- Use appropriate data types for your fields. Elasticsearch supports several data types, such as text, keyword, and numeric, and choosing the right data type can improve the performance and accuracy of your search queries.
- Use proper mapping for your data. Elasticsearch allows you to define the mapping for your data, which specifies the data type and other properties of each field. Defining a proper mapping can help Elasticsearch understand your data better and improve the performance of your search queries.
- Use relevance scoring to determine the relevance of search results. Elasticsearch uses a relevance score to determine the relevance of each document to a search query. You can customize the relevance score by defining custom boosting rules and query clauses.
- Use filtering and aggregations to narrow down and summarize search results. Elasticsearch allows you to use filters and aggregations to narrow down and summarize the search results. This can help you provide more relevant and useful search results to your users.
- Use sharding and replication to improve the performance and availability of your search cluster. Elasticsearch allows you to divide your data into multiple shards and distribute them across multiple nodes in a cluster. This can improve the performance and availability of your search cluster, especially when dealing with large amounts of data.

3.9.MongoDB

Here are some best practices for using MongoDB:

- Use the appropriate data model for your data. MongoDB allows you to store data in documents, which are similar to JSON objects. Choosing the right data model, such as embedding or referencing, can help ensure the performance and scalability of your MongoDB database.
- Use indexes to improve the performance of your queries. MongoDB allows you to create indexes on your collections to improve the performance of your queries. Indexes

can help MongoDB find and retrieve data quickly, especially when dealing with large collections.

- Use the appropriate write concern for your application. MongoDB allows you to specify the level of acknowledgment you want for writes to the database. Choosing the right write concern can help ensure the performance, availability, and consistency of your data.
- Use sharding to distribute data across multiple servers. MongoDB allows you to distribute your data across multiple shards, which can improve the performance and scalability of your database.
- Use replica sets to improve the availability and durability of your data. MongoDB allows you to create replica sets, which are groups of MongoDB instances that maintain the same data set. Replica sets can improve the availability and durability of your data, especially in the case of server failures or network partitions.

3.10.Kafka

Here are some best practices for using Apache Kafka:

- Use appropriate partitions and replication factors. Kafka partitions allow you to distribute and parallelize data processing, while replication factors allow you to have multiple copies of your data for fault tolerance. Choosing the right values for these settings can help ensure the performance and availability of your Kafka cluster.
- Use unique and meaningful topic names. Kafka topics are the logical containers for your data, and using unique and meaningful names can help you organize and manage your data effectively.
- Use consumer groups and offsets to manage consumer state. Kafka consumer groups allow you to divide your consumers into logical groups, and offsets allow you to track the progress of each consumer within a group. Using consumer groups and offsets can help you manage the state of your consumers and ensure reliable data processing.
- Use Kafka Connect to integrate with other systems. Kafka Connect is a framework for integrating Kafka with other systems, such as databases, message queues, and file systems. Using Kafka Connect can help you move data between Kafka and other systems easily and reliably.
- Use Kafka Streams for stream processing. Kafka Streams is a library for building real-time stream processing applications on top of Kafka. Using Kafka Streams can help you process and analyze your data in real-time, without the need for separate stream processing systems.