MBN Events Management Web Application: Design Report

*By Magdy Saleh, Niren Patel and Bindia Venugopal*

## Introduction

MBN Events is a web application that allows users to create and post their events, search and attend other events, as well as keep track of their activities. The users must register and log in to their account before they can access the facilities of the website. We used the Laravel 5 framework for web development to structure our web application which uses the model-view-controller principles. Please follow the link the find the YouTube video: <https://www.youtube.com/watch?v=s88ZflqGch0> The project can be found in this repository, under the branch **‘final code’**: <https://github.com/MagdyNirenBindia/LaravelWebPage/tree/finalCode>

## Web Architecture Design

By utilising Laravel’s intrinsic MVC design we were able to elegantly deal with user requests and data while separating it from the views. By creating specific routes for the links in the application and using different controllers to deal with the requests we are able to section the webpage and deal with similar requests easily.

Our application is mainly server side and the bulk of the functionality is dependent on the PHP server side implementation. However, in order for a sensible UX and to debug we used client side scripts in JavaScript and heavily depending on jQuery to implement this. These came in two main forms. Firstly, client side form validation is essential for our application to run and handle logical errors before they are entered into the server side.

We used validation features to design the forms such that users are prompted to enter dates only in the future when creating events and likewise restricting ticket sales from taking place after the date of the event. The webpage displays some form of error message when invalid dates are entered. We have also added a restriction such that a minimum of 5 words must be entered when writing the ‘description’ field of the ‘create events’ webpage. We used Laravel’s own login and register interfaces to make it easier and more effective to implement in our web application. The features include: automatic validation of email address to make sure it is in the correct format; validating a minimum length of 6 characters for password entries; making sure the password is the same twice in the registration page; and being able to reset password if the user is unable to login to their account. In this case, an email is sent directly to the users account for details on how to reset the password.

The second main role of the client side development was to dynamically create pages based on user specific information. This information was first queried by using Laravel’s eloquent methods and then printed in html tags that are hidden from the user. We used this method over AJAX.

We used JavaScript and jQuery to make our web page dynamic, for example, in our browse events page the events appear and disappear based on what the user types and/or filters in the search bar. This content changes without needing to refresh the page. Also, when an event is created in the ‘Create Events’ page, it will automatically update the results in the ‘Browse Events’ page to display the newly generated event. Likewise, our feedback page can dynamically update when new reviews are entered.

Database Design and SQL queries

FINAL3.png

The diagram above represents the relationship between the different tables in the database of our website. Here is a breakdown of all the relationships (‘Events Confirmed’ is similar to Orders Placed):

|  |  |  |
| --- | --- | --- |
| **Database Tables** | **Relationship** | **Description** |
| ‘Users’ and ‘Events’ | 1 mandatory to many optional | Users have the option to create many events, but it is mandatory for each event entry to have only one user that creates the event |
| ‘Users’ and ‘Feedback’ | 1 mandatory to many optional | Users have the option to write multiple feedback, but it is mandatory for each feedback entry to have only one user who writes the feedback |
| ‘Users’ and ‘Events Confirmed’ | 1 mandatory to many optional | Users have the option to attend many events, but it is mandatory for each events confirmed entry to have only one user |
| ‘Events’ and ‘Feedback’ | 1 mandatory to many optional | Events can have multiple entries in the Feedback table, but it is mandatory for each feedback value to have only one event |
| ‘Feedback’ to ‘Events Confirmed’ | 0 to 1 mandatory | For feedback to be submitted, the user must have a value in Events Confirmed, but the events confirmed table does not need to have any feedback |
| ‘’Events’ and ‘Events Confirmed’ | 1 mandatory to many optional | An event can have many entries in the events confirmed table, but it is mandatory for each events confirmed entry to have only one event |

The entity relationship diagram follows the same schema, and the explanation for how these tables can have relations with each other is due to the many occurrences of foreign and primary keys relations, allowing the database tables to be related to each other. Where Customer IDs (User IDs) and Event IDs appear in other tables apart from their own, they are called foreign keys. For example, in the Feedback Table, the entries which are Customer ID and Event ID are both examples of foreign keys that were primary keys in their respective parent tables. Because of this, it is now possible to see which user wrote what feedback, for what event. This was also used to prevent people from providing feedback to events they had not attended. Every table apart from the Event ID and Customer ID tables in the entity relationship diagram contain entries that each contain both an Event ID and a Customer ID. In the case of the Events table, it does contain the Creator ID as a foreign key, however, the Users table does not require the Event ID to appear as a foreign key in its table. Having a separate table to show every single record of someone attending an event, made it easier to separate the events users were attending from the ones they had created; which can be shown on the homepage.

In addition to the tables above, there are a few more tables that are required for the database schema, to allow the smooth functionality of the website, multi-device editing support and also for security purposes. These are the following four additional tables that do not display any data on the client-side:

Untitled Diagram.png

The migrations table was an important tool in making sure that we did not have to spend an excessive amount of time discussing the schema of the database contained on our respective local drives. It worked as a version control, in which people could stay up to date with the most current schema when downloading the latest version of the Laravel project, this meant there would be no errors when we tried to host the Laravel project on our own computers.

The password resets table allows users to reset their password in the case they forget it, it is linked to the users table and ensures the security of the process as the database does not store any unencrypted passwords, which makes it easier for hackers to obtain passwords.

The jobs and failed\_jobs tables are used to keep a record of any automatic tasks that Laravel runs. In the case of our website, where automatic emails are sent regularly to remind users of upcoming events, it is able to record the success or failure of any tasks that were in queues. Queues ensure that if the user is using the website, the speed of the service is not affected by tasks running in the background on the servers, the job will then then run when the server is less busy. These tasks are scheduled in the kernel and are implemented by cron jobs. If there are no emails being sent to attendees of events, then it should appear in the failed jobs table. The administrator of the system, regularly viewing both tables, can then take the necessary procedures to rectify the issues.

In terms of queries used in the application, no raw SQL queries were used. Instead, Laravel’s Eloquent ORM language was used to query the database and obtain information. This was chosen because the improved readability of the eloquent language in addition to its easy integration with laravel. By doing this, only one database connection needed to be established and the database can be directly accessed from anywhere in the application from the php without having to reconnect. Furthermore, eloquent extends object like functions to database queries and allows for dealing with different rows of the database as objects.

**Listing of database queries by page:**

Login:

* Query the user table to check if user exists, and if they do, does the password match the password supplied, if it does send to home

Register:

* Take the information supplied in the form and insert it as a new row in the user table, under the condition that the email is unique

Home:

* Query the events\_confimred table to see if the user is attending any events, by checking their id in the CustomerID column, and then echoing all distinct results under the events attending section
* Query the event table to see if the user has created any events, by checking their id in the CreatorID column, and then echoing all distinct results under the events created section

Browse Events:

* Here we get all the rows from the event table, and echo them as a UI, and then filter the results by hiding them through CSS manipulation based on JavaScript commands

Create Event:

* Take the information supplied in the form and submit it as a new row into the event table, under the condition that the information satisfies the form validation restrictions

View Feedback:

* Echo all the rows from the feedback table into the html table

Give Feedback:

* Take the information supplied in the form and submit it as a new row into the feedback table, under the condition that the information satisfies the form validation restrictions

View Event:

* Query the event table to find the row of the event with the eventID that has been passed on through a post method from the previous page.
* Echo the corresponding fields of the row in the different html tags on the page

Participant List:

* Query the event table for the row of the event corresponding the eventID supplied from the previous page
* Query the events\_confirmed table for all distinct rows that have eventID equal to the eventID that was passed on through a post method from the previous page
  + By queering for distinct values the system automatically doesn’t echo any duplicate entries that might have occurred
* Query the user table for each userID received from the events\_confirmed table, to get a list of emails and names for the users attending the event.
* Get the size of the result and compare that to the number of tickets available from the event table query to get a ratio of the ticket sales progress

## Testing and Error Handling:

As mentioned above, we have included a failed jobs table which will report of any scheduled tasks, such as the automatic email updates, that fail to run when the website is placed on a server. These tasks can then be corrected through thorough testing on the command line terminal or by reviewing the cron job in the server, or the code that is running in the kernel. This error is generally unnoticeable for users when using the system, as these tasks run in the background, so no error will be displayed whilst the website is in use.

We used Laravel’s error reporting for pages that don’t exist, e.g. if a user types in a page name after the domain name that does not exist, the web browser will display a page notifying the user that “this page does not exist” rather than a general error, and users will realise they typed something incorrectly . Since there are only a few pages, it is unlikely a user will see this page, unless they manually type an incorrect link.

After encountering multiple errors when entering form data in earlier development stages, we decided to enter validation measures to notify users to retype data if they type something that is not in the correct format. However, this can become tedious for certain inputs such as date and time, where people may have different styles of inputting date. To Counter this, where possible, we used input forms where users can select options from a dropdown box, or from a date/time input that makes it simple for users to type times in a format that the database can accept.

In addition to this, we have eliminated the need to type in names and email addresses, when creating an event or clicking attending events, through the use of hidden input tokens that are automatically able to detect which user is currently logged in. This eliminates the chance of a user entering their personal data wrong and not being registered for events, meaning they do not receive any email updates.

Any minor errors, such as those involving CSS, have for the most part been eliminated, i.e. “failed to load CSS in the command line interface terminal”, as this can sometimes occur when an online reference to CSS is provided rather than having typed out styling codes manually. Therefore, for most of the pages we have decided to create the styling ourselves or copied CSS styling files into our project. Furthermore, keeping CSS files in separate files and not in-text with HTML, allowed us to detect problems easier and avoided confusion between developers in the testing stage.

As previously mentioned, we have used Laravel’s Eloquent ORM language to prevent database error issues. When a server hosts the website, only the details in one environment file must be changed to make sure that database is connected, rather than establishing connection on all the pages. Pages can often be overlooked and some details can often be missed, therefore this proves highly advantageous.

We have added multiple redirection pages after a form has been submitted to avoid confusion and reduce the chance of form resubmission from people hitting the back button, this has provided particularly useful in the scenario of password reset forms. Whilst we were in testing mode, we used the built in developer options on our respective web browsers to see if there were any possible discrepancies and any other minor errors occurring.