Project Report

Spring 2024

CSE-403L Database Management System Lab

Submitted by:

Ali Asghar(21PWCSE2059)

Shahzad Bangash(21PWCSE1980)

Suleman Shah(21PWCSE1983)

Class Section: **C**

Submitted to:

Engr. Sumayyea Salahudin

Date:

27th June 2024

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Hall Booking System in Laravel

Contents

Chapter 1

Introduction

1.1 Project Overview

This project involves integrating a Unity game with a MySQL database. A standard method for creating a game has always been a big challenge faced by game developers. In this project, we aim to create a standard game architecture method for creating a shooting game using Unity as Game Engine and MySQL as backend database for our game.

1.2 Keywords

MySQL, Laravel, Database, Operating System(OS), DBMS, Relational Schema, Conceptual Schema, Web App.

1.3 Objectives

- Make a standardized hall booking system.
- Create a MySQL database for our project.
- Connect our project with a MySQL database.
- Manage the relation betweeen our web application and database

1.4 Softwares/Frameworks Used

- MySQL.
- Laravel.
- Draw.IO.

1.5 Database Management System

A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database[2].

1.5.1 Types of DBMS

DBMS can be classified into two types: Relational Database Management System (RDBMS) and Non-Relational Database Management System (NoSQL or Non-SQL)

1.5.2 RDBMS

Data is organized in the form of tables and each table has a set of rows and columns. The data are related to each other through primary and foreign keys.

1.5.3 NoSQL

Data is organized in the form of key-value pairs, documents, graphs, or column-based. These are designed to handle large-scale, high-performance scenarios.

1.5.4 MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. It is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons [1].

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

1.5.5 Database System and DBMS Workflow

The general workflow of a database system is that queries are passed to a DBMS which interpret it. After succesful execution of a query, OS(Operating System) interacts with actual database and retrieves/add/delete data from database as shown in1.1. Below is the more refined workflow of a database system.

Refined Workflow is:

- User/Application $\rightarrow$ DBMS
- DBMS interprets and optimizes the query.

- DBMS requests the OS for data access.
- OS reads/writes data from/to disk.
- OS → DBMS
- DBMS processes the data and returns results to User/Application.

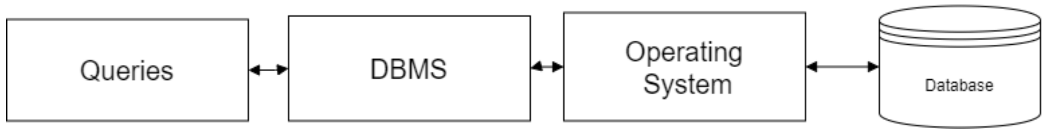


Figure 1.1: Database System Workflow

1.6 Laravel

Laravel is a web application framework with expressive, elegant syntax. A web framework provides a structure and starting point for creating your application, allowing you to focus on creating something amazing while we sweat the details. Laravel strives to provide an amazing developer experience while providing powerful features such as thorough dependency injection, an expressive database abstraction layer, queues and scheduled jobs, unit and integration testing, and more. Whether you are new to PHP web frameworks or have years of experience, Laravel is a framework that can grow with you. We'll help you take your first steps as a web developer or give you a boost as you take your expertise to the next level. We can't wait to see what you build[**?**].

1.6.1 Laravel Framework Architecture

Some of the salient features of laravel are:

- Bundles: Modular packaging system and dependencies.
- Routing: Best routing feature provide a hassle-free path taken out every complexity.

- ORM support: Relationship and mapping of the database due to the ORM support.
- Migrations: A way to version our database scripts in a much more efficient manner.n
- Queue Management: To make the process faster
- Dependency Injection: For ease in testing

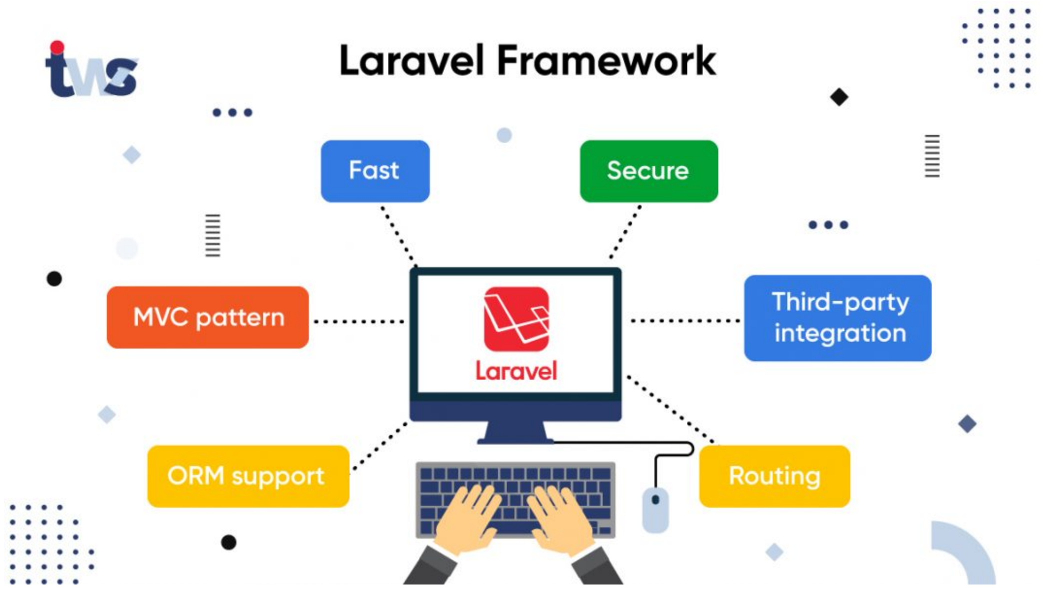


Figure 1.2: Laravel Framework Architecture [4]

1.7 Draw.IO

draw.io, also known as Diagrams.net, is a customized tool for creating diagrams and charts. It offers existing automated layout options, or the user can design a custom layout for their preferences. The tool provides a wide range of shapes and hundreds of graphic components to let you create unique diagrams. Depending on your needs, draw.io can keep saved charts in the cloud or in network storage at a data center. It works both offline and on mobile devices. The service does not keep any of your data or claim ownership of your creations, ensuring that you are safe regardless of where or how you work[?].

1.7.1 Role of Draw.IO in this project

In this project, draw.io is used for making diagrams for conceptual and relational schema. Draw.io is a great tool and saves a lot of time when making a schema for a database.

Chapter 2

Logical Database Design

2.1 Entity Descriptions

- **Customer** A customer or organization that can book any number of halls and submit reviews.
- **Hall** A hall that can be booked by customer and must have an assigned owner and manager. It can be of two types, wedding hall or conference center.
- **Booking** A booking entity contains info about booking of hall by a customer.
- **Invoice** An invoice must be generated for each booking, detailing the payment information, the customer who booked, and the hall that was booked.
- **Review** Reviews may be submitted by customers, providing feedback on their experience with the booked hall.
- **Owner** An Owner of its specified hall.

2.2 Business Rules

1. A customer can book one or more halls for events. A customer may submit reviews for each hall they book, providing feedback on their experience.

2. Each hall can be a wedding hall, conference hall. Each hall must have exactly one assigned owner. A hall can be booked by multiple customers over time.

3. An invoice must be generated for each hall booking. The invoice has details about payment, time of booking, customer details.

4. Customers are encouraged to submit reviews after their event. Reviews should provide honest feedback about the customer's experience with the hall. Reviews are used to assess the quality of service and identify areas for improvement.

5. Owner is responsible for the overall property and condition of the hall. The owner may delegate day-to-day operational responsibilities to a manager but retains ultimate accountability for the hall

2.3 Conceptual Schema

The database conceptual schema includes entity relationship diragram (ERD) for customer, hall, manager and other related entities. The schema is designed to handle various relationships between these entities.

2.3.1 Conceptual Schema Sketch

Sketch of Relational Schema is given in figure 2.1.

2.4 Relational Schema

The following normalized relational schema is obtained from the conceptual schema from previous section.

2.4.1 Relational Schema Sketch

Sketch of Relational Schema is given in figure 2.2..

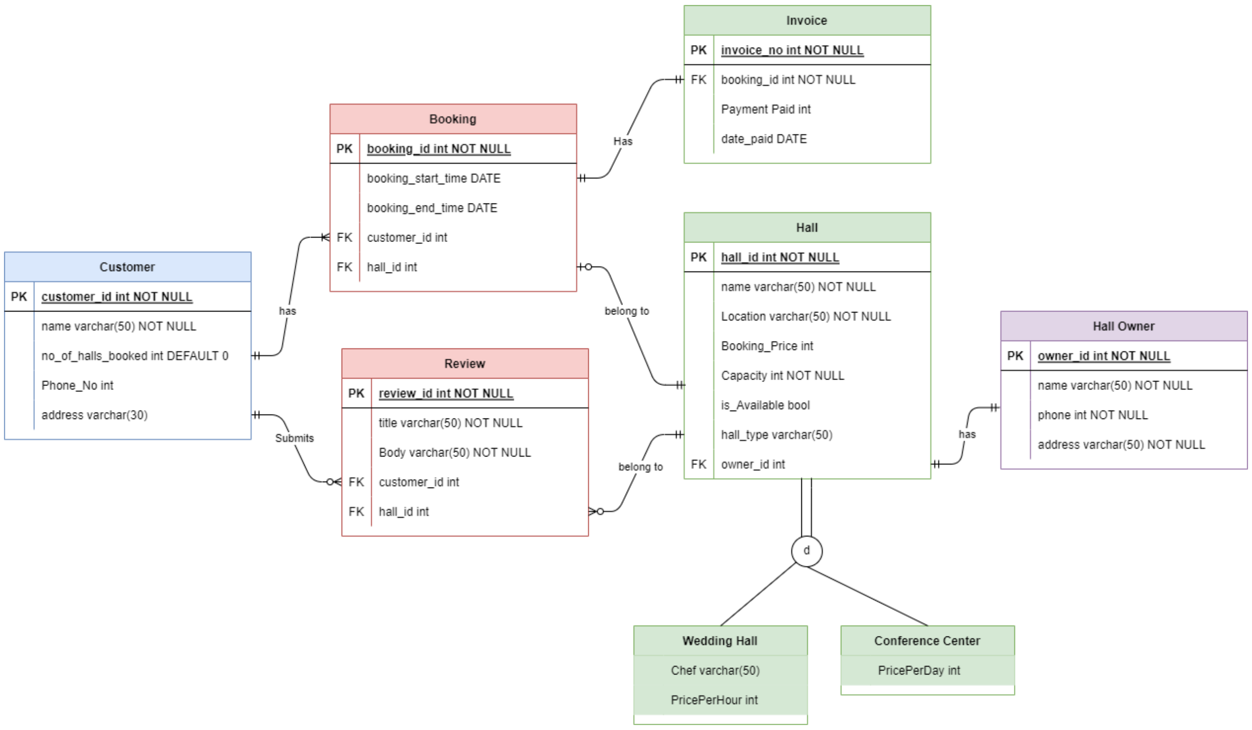


Figure 2.1: Database Conceptual Schema

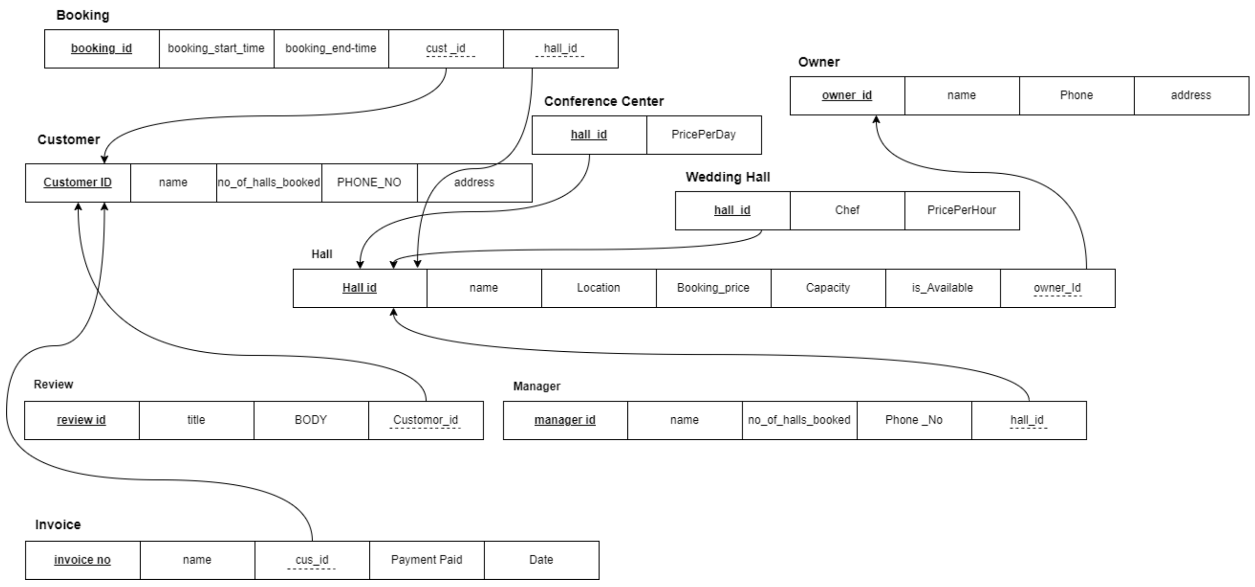


Figure 2.2: Database Relational Schema

Chapter 3

Physical Database Design

3.1 Physical Database Structure

After running successful SQL queries, following tables we created in the database. These tables are physically stored in the memory of the computer as shown in1.1.

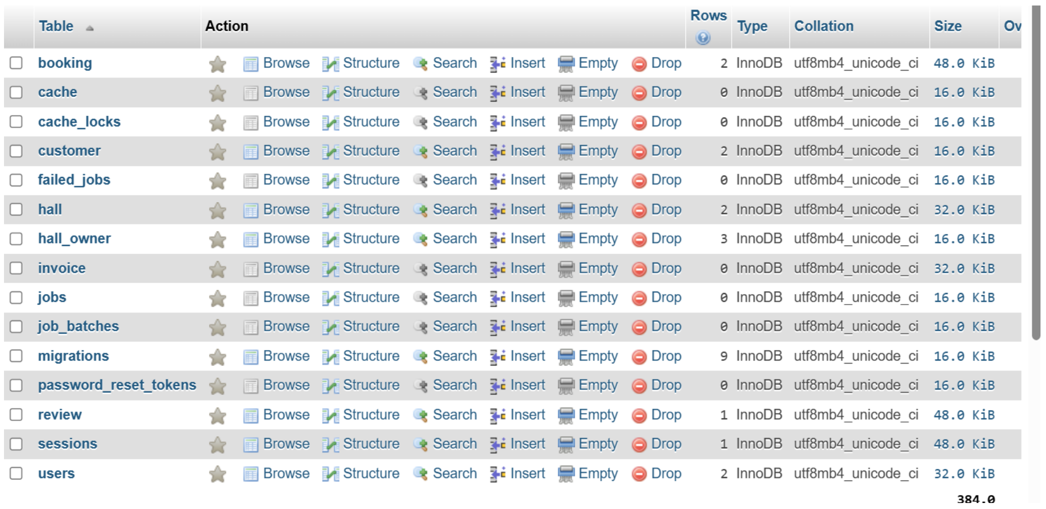


Table	Action	Rows	Type	Collation	Size	Ov
booking	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	48.0 KiB	
cache	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
cache_locks	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
customer	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
failed_jobs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
hall	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
hall_owner	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
invoice	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
jobs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
job_batches	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
migrations	Browse Structure Search Insert Empty Drop	9	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
password_reset_tokens	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	
review	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	48.0 KiB	
sessions	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	48.0 KiB	
users	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	32.0 KiB	
					384.0	

Figure 3.1: Database Physical Structure

3.2 SQL Queries for Tables Creation

Following queries were given to MySQL for tables creation.

3.2.1 Booking Table

```
CREATE TABLE booking(
  ‘id‘ bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  ‘customer_id‘ bigint UNSIGNED NOT NULL,
  ‘hall_id‘ bigint UNSIGNED NOT NULL,
  ‘booking_start_time‘ date NOT NULL,
  ‘booking_end_time‘ date NOT NULL,
  PRIMARY KEY (‘id‘),
  KEY ‘booking_customer_id_foreign‘ (‘customer_id‘),
  KEY ‘booking_hall_id_foreign‘ (‘hall_id‘)
);
```

3.2.2 Customer Table

```
CREATE TABLE customer(
  ‘id‘ bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  ‘name‘ varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘no_of_halls_booked‘ int UNSIGNED NOT NULL DEFAULT ’0’,
  ‘phone_No‘ varchar(13) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘address‘ varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (‘id‘)
);
```

3.2.3 Hall Table

```
CREATE TABLE hall (
  ‘id‘ bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  ‘name‘ varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘location‘ varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘booking_price‘ int UNSIGNED NOT NULL,
  ‘capacity‘ int UNSIGNED NOT NULL,
  ‘is_available‘ tinyint(1) NOT NULL DEFAULT ’1’,
  ‘hall_type‘ varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘owner_id‘ bigint UNSIGNED NOT NULL,
  ‘created_at‘ timestamp NULL DEFAULT NULL,
  ‘updated_at‘ timestamp NULL DEFAULT NULL,
  PRIMARY KEY (‘id‘),
  KEY ‘hall_owner_id_foreign‘ (‘owner_id‘)
);
```

3.2.4 Hall Owner Table

```
CREATE TABLE hall_owner(
  ‘id‘ bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  ‘name‘ varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘phone_No‘ varchar(13) COLLATE utf8mb4_unicode_ci NOT NULL,
  ‘address‘ varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (‘id‘)
);
```

3.2.5 Invoice Table

```
CREATE TABLE invoice(
  ‘id‘ bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  ‘booking_id‘ bigint UNSIGNED NOT NULL,
  ‘payment_paid‘ int UNSIGNED NOT NULL,
  ‘date_paid‘ date NOT NULL,
  PRIMARY KEY (‘id‘),
  KEY ‘invoice_booking_id_foreign‘ (‘booking_id‘)
);
```

3.2.6 Review Table

```
CREATE TABLE review(
  ‘id‘ bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  ‘customer_id‘ bigint UNSIGNED NOT NULL,
  ‘hall_id‘ bigint UNSIGNED NOT NULL,
  `title` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL,
  `body` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  PRIMARY KEY (`id`),
  KEY `review_customer_id_foreign` (`customer_id`),
  KEY `review_hall_id_foreign` (`hall_id`)
);
```

3.2.7 Booking Table

```
CREATE TABLE booking(
  `id` bigint UNSIGNED NOT NULL AUTO_INCREMENT,
  `customer_id` bigint UNSIGNED NOT NULL,
  `hall_id` bigint UNSIGNED NOT NULL,
  `booking_start_time` date NOT NULL,
  `booking_end_time` date NOT NULL,
  PRIMARY KEY (`id`),
  KEY `booking_customer_id_foreign` (`customer_id`),
  KEY `booking_hall_id_foreign` (`hall_id`)
);
```

Chapter 4

Making Web Application in Laravel -Phase 1

4.1 Introduction

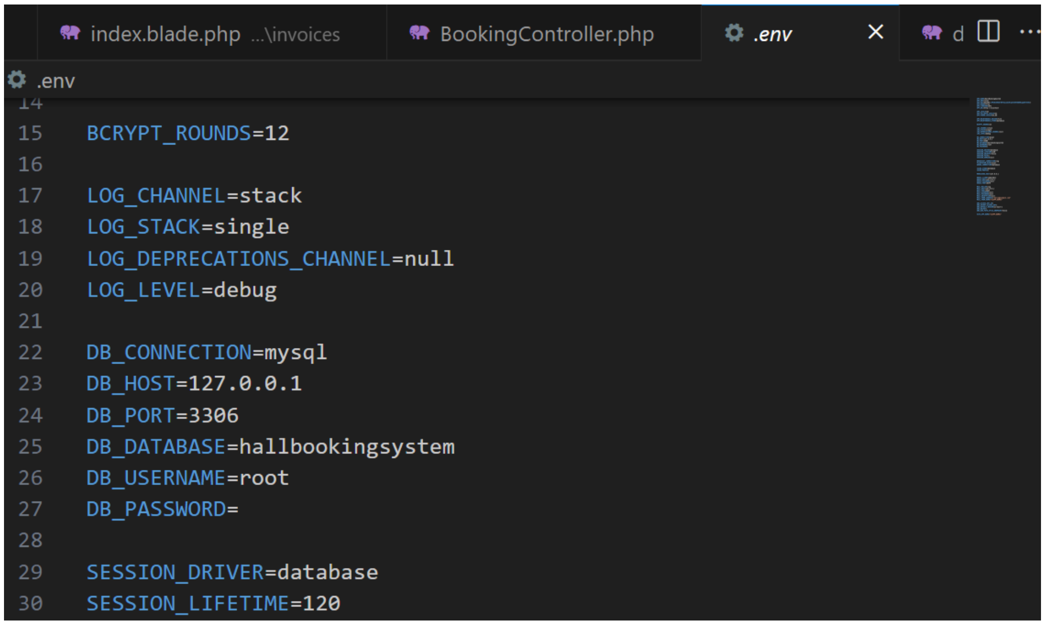
Making a Web App in Laravel involves writing php scripts for server side scripting, connecting with the database and then displaying the data through the frontend to the user. Laravel framework follows the MVC (Model-View-Controller) architecture.

4.2 Database Connection Code - MySQL Connection

We can connect database with Laravel App using .env file. After making some changes in .env file, we were able to establish connection with the database as shown in figure 4.1.

4.3 Running Migrations

After successfully connecting database with Laravel app, next step is to run migrations and make the desired tables inside the database.



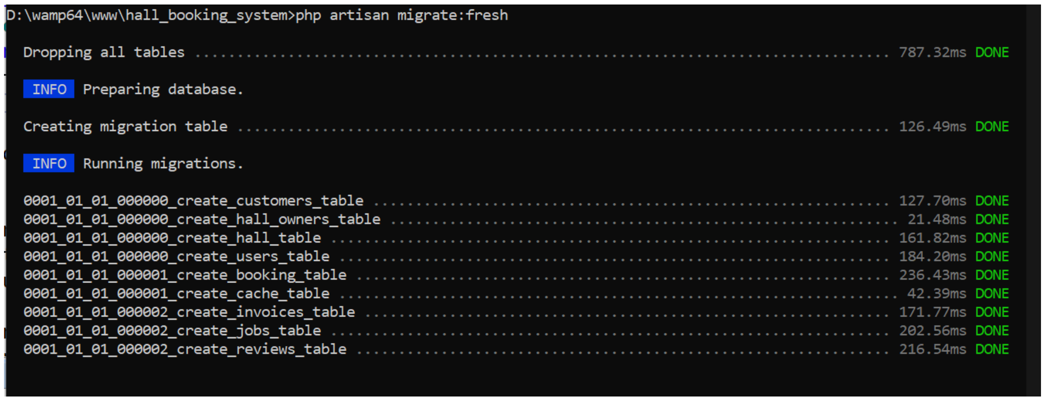
```
BCRYPT_ROUNDS=12

LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=hallbookingsystem
DB_USERNAME=root
DB_PASSWORD=

SESSION_DRIVER=database
SESSION_LIFETIME=120
```

Figure 4.1: .env File Contents



```
D:\wamp64\www\hall_booking_system>php artisan migrate:fresh

  Dropping all tables ...................................................................... 787.32ms DONE

  INFO  Preparing database.

  Creating migration table ................................................................. 126.49ms DONE

  INFO  Running migrations.

  0001_01_01_000000_create_customers_table ................................................. 127.70ms DONE
  0001_01_01_000000_create_hall_owners_table ................................................ 21.48ms DONE
  0001_01_01_000000_create_hall_table ...................................................... 161.82ms DONE
  0001_01_01_000000_create_users_table ..................................................... 184.20ms DONE
  0001_01_01_000001_create_booking_table ................................................... 236.43ms DONE
  0001_01_01_000001_create_cache_table ...................................................... 42.39ms DONE
  0001_01_01_000002_create_invoices_table .................................................. 171.77ms DONE
  0001_01_01_000002_create_jobs_table ...................................................... 202.56ms DONE
  0001_01_01_000002_create_reviews_table ................................................... 216.54ms DONE
```

Figure 4.2: Migration in Laravel

Chapter 5

Making Web Application in Laravel -Phase 2

5.1 Introduction

In this phase, we write model, view and controller php classes for each entity type. MVC is explained briefly in following next section.

5.2 MVC Architecture

MVC architecture is fundamental to Laravel, providing a structured approach to organizing your codebase as shown in figure 5.1. In this project, I have created Model and Controller class for each entity in the conceptual schema.

5.3 Code of Some Model Classes

Below are some of the codes of model classes. We have avoided namespaces in here just for the convenvience.

5.3.1 Customer Model

```
<?php

class Customer extends Model
```

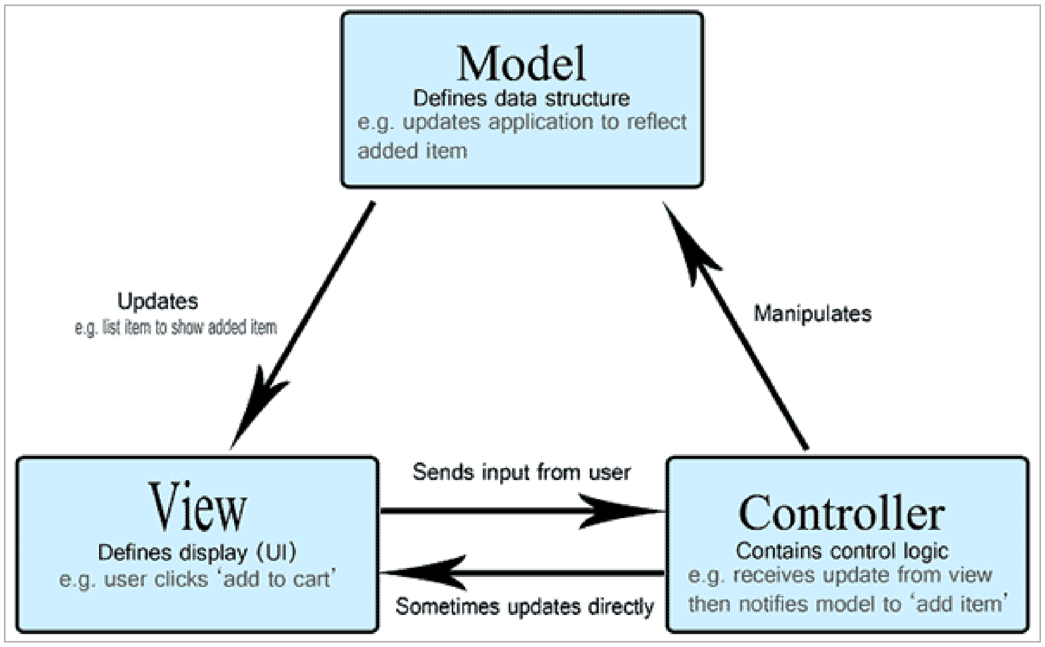

Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Figure 5.1: MVC Architecture

```
{
    use HasFactory;

    protected $table = 'customer';
    protected $primaryKey = 'id';
    public $timestamps = false;

    protected $fillable = [
        'name',
        'no_of_halls_booked',
        'phone_No',
        'address'
    ];


        public function reviews(){
            return $this->hasMany(Review::class);
        }

        public function bookings(){
            return $this->hasMany(Booking::class);
        }
}
```

5.3.2 Hall Model

```php
<?php

class Hall extends Model
{
    use HasFactory;

    protected $table = 'hall';
    protected $primaryKey = 'id';
    public $timestamps = false;

    protected $fillable = [
        'name',
        'location',
        'owner_id',
        'booking_price',
        'capacity',
        'hall_type' ,
    ];
    protected $casts = [
        'is_available' => 'boolean',
    ];
    public function booking(){
        return $this->hasOne(Booking::class);
    }
    public function reviews(){
        return $this->hasMany(Review::class);
    }

    public function hall_owner(){
        return $this->belongsTo(HallOwner::class);
    }
}
```

5.3.3 Booking Model

```
<?php
class Booking extends Model
{
    use HasFactory;

    protected $table = 'booking';
    protected $primaryKey = 'id';
    public $timestamps = false;

    protected $fillable = [
        'booking_start_time',
        'booking_end_time',
        'customer_id',
        'hall_id',
    ];

    public function customer(){
        return $this->belongsTo(Customer::class);
    }
    public function hall(){
        return $this->belongsTo(Hall::class);
    }

    public function invoice(){
        return $this->hasOne(Invoice::class);
    }
}
```

5.4 Code of Some Controller Classes

Below are some of the codes of controller classes. We have avoided namespaces in here just for the convenience.

5.4.1 Customer Controller

```php
<?php
class CustomerController extends Controller
{
    public function index()
    {
        $customers = Customer::all();
        return view('customers.index', compact('
           customers'));
    }

    public function create()
    {
        return view('customers.create');
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|max:50',
            'no_of_halls_booked' => 'nullable|
               integer',
            'phone_No' => 'nullable|integer',
            'address' => 'nullable|max:30',
        ]);

        Customer::create($request->all());
        return redirect()->route('customers.index'
           )->with('success', 'Customer created
           successfully.');
    }

    public function show(Customer $customer)
    {
        return view('customers.show', compact('customer'));
    }

    public function edit(Customer $customer)
    {
        return view('customers.edit', compact('customer'));
    }

    public function update(Request $request, Customer $customer)
    {
        $request->validate([
            'name' => 'required|max:50',
            'no_of_halls_booked' => 'nullable|integer',
            'phone_No' => 'nullable|integer',
            'address' => 'nullable|max:30',
        ]);

        $customer->update($request->all());
        return redirect()->route('customers.index')->with('success', 'Customer updated successfully.');
    }

    public function destroy(Customer $customer)
    {
        $customer->delete();
        return redirect()->route('customers.index')->with('success', 'Customer deleted successfully.');
    }
}
```

5.4.2 Hall Controller

```php
<?php
class HallController extends Controller
{
    public function index()
    {
        $halls = Hall::all();
        return view('halls.index', ['halls' =>
           $halls]);
    }

    public function create()
    {
        $customers = Customer::all();
        $hall_owners = HallOwner::all();
        return view('halls.create', compact('
           customers','hall_owners'));
    }

    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required|max:50',
            'location' => 'nullable|string',
            'owner_id' => 'nullable|integer',
            'booking_price' => 'nullable|integer',
            'capacity' => 'nullable|integer',
            'is_available' => 'nullable|boolean',
            'hall_type' => 'nullable|string',
        ]);

        Hall::create($request->all());
        return redirect()->route('halls.index')->
           with('success', 'Hall created
           successfully.');
    }

    public function show(Hall $hall)
    {
        return view('halls.show', compact('hall'));
    }

    public function edit(Hall $hall)
    {

        $hall_owners = HallOwner::all();
        return view('halls.edit', compact('hall','hall_owners'));
    }

    public function update(Request $request, Hall $hall)
    {
        $request->validate([
            'name' => 'required|max:50',
            'location' => 'nullable|string',
            'owner_id' => 'nullable|integer',
            'booking_price' => 'nullable|integer',
            'capacity' => 'nullable|integer',
            'is_available' => 'nullable|boolean',
            'hall_type' => 'nullable|string',
        ]);

        $hall->update($request->all());
        return redirect()->route('halls.index')->with('success', 'Hall updated successfully.');
    }

    public function destroy(Hall $hall)
    {
        $hall->delete();
        return redirect()->route('halls.index')->with('success', 'Hall deleted
                successfully.');
        }
}
```

5.4.3 Booking Controller

```
<?php

class BookingController extends Controller
{
public function index()
{
    $bookings = Booking::with('invoice',)->get();
       // Adjust the number as needed for
       pagination

    return view('bookings.index', compact('
       bookings'));
}

    public function create()
    {
        $customers = Customer::all();
        $halls = Hall::all();
        return view('bookings.create', compact('
           customers','halls'));
    }

    public function store(Request $request)
    {
        $request->validate([
            'booking_start_time' => 'nullable|date
               ',
            'booking_end_time' => 'nullable|date|
               after:booking_start_time', //
               Ensure end time is after start time
            'hall_id' => 'required|exists:hall,id'
```

```php
            , // Ensure the hall exists
        ]);

        $booking = new Booking();
        $booking->booking_start_time = $request->booking_start_time;
        $booking->booking_end_time = $request->booking_end_time;
        $booking->customer_id = auth()->id(); // Set the authenticated user's ID
        $booking->hall_id = $request->hall_id;
        $booking->save();

        $hall = Hall::findOrFail($request->hall_id);

        $hall->is_available = false; // Set the boolean field to false (assuming you want to mark it as unavailable)
        //dd( $hall->is_available);
        $hall->save();

        // Create an invoice record
        $invoice = new Invoice();
        $invoice->date_paid = $booking->booking_start_time; // Assuming you have a 'booking_id' column in your 'invoices' table
        $invoice->booking_id = $booking->id; // Assuming you have a 'booking_id' column in your 'invoices' table
        $invoice->payment_paid = $hall->booking_price; // Replace with actual amount calculation logic
        $invoice->save();
        return redirect()->route('bookings.index')->with('success', 'Booking created successfully.');
```

```
    }


    public function show(Booking $booking)
    {
        return view('bookings.show', compact('booking'));
    }

    public function edit(Booking $booking)
    {
        return view('bookings.edit', compact('booking'));
    }

    public function update(Request $request, Booking $booking)
    {
        $request->validate([
            'booking_start_time' => 'nullable|date',
            'booking_end_time' => 'nullable|date',
            'customer_id' => 'nullable|integer',
            'hall_id'=> 'nullable|integer',
        ]);

        $booking->update($request->all());
        return redirect()->route('bookings.index')->with('success', 'Booking updated successfully.');
    }

    public function destroy(Hall $hall)
    {
        $hall->delete();
        return redirect()->route('bookings.index')->with('success', 'Booking deleted successfully.');
        }
}
```

5.5 View Code - Frontend Design

The frontend is written in blade files. These files display the data to the user of this app. Each controller has index, create, edit and show blades. It can be seen in the figure

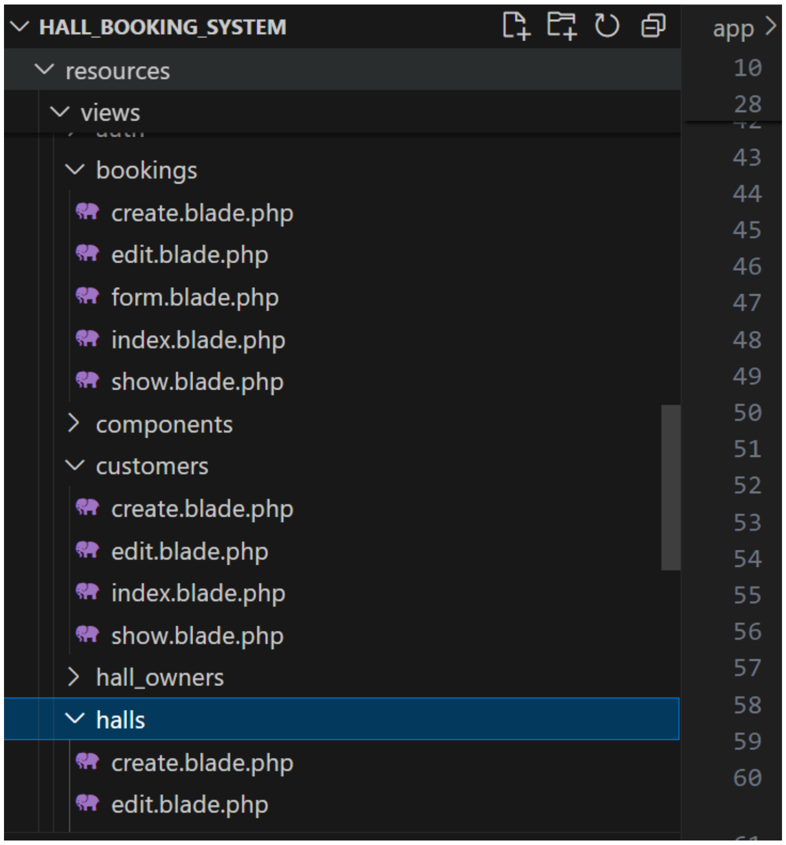


Figure 5.2: View/Blade Files for Model and its Controller

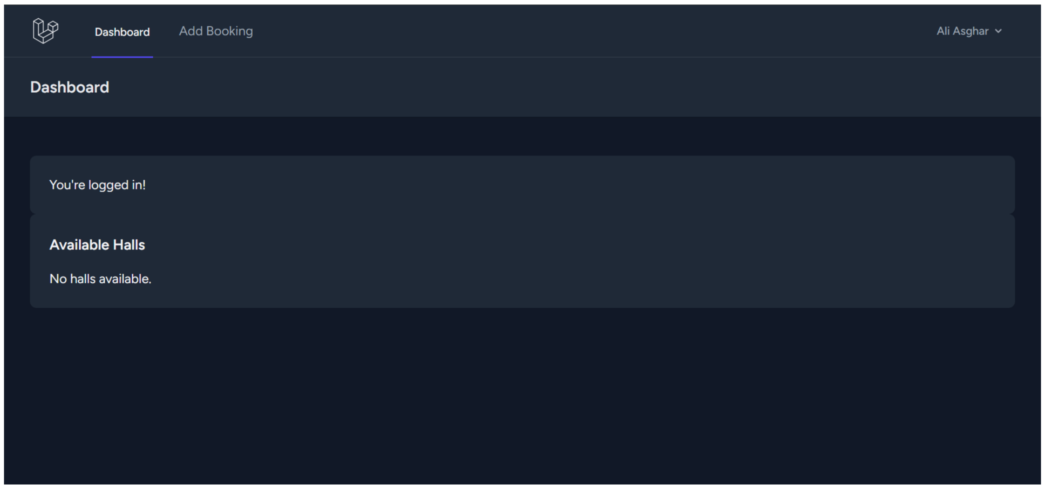
Chapter 6

Conclusion

This project demonstrates how to integrate a Laravel App with a MySQL database. The project provides some of the features of a hall booking system.

6.1 Final Output

Following are some of the snapshots of the output of this project.



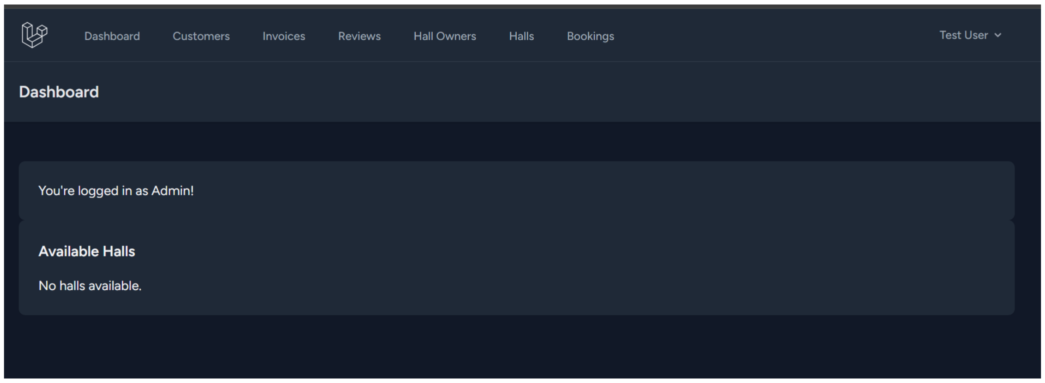
Dashboard

You're logged in!

Available Halls

No halls available.

Figure 6.1: Output Snapshot 1



Dashboard

You're logged in as Admin!

Available Halls

No halls available.

Figure 6.2: Output Snapshot 2 - Admin View

References

[1] https://www.tutorialspoint.com/mysql/mysql-introduction.htm

[2] https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1

[3] https://laravel.com/docs/11.x

[4] https://www.tekkiwebsolutions.com/blog/all-about-laravel-framework/