Further Programming Assessment 2

Rafat Mahiuddin (s3897093)

Justification of Design Choice Report

How did you apply the MVC design pattern to build this application?

I applied the MVC pattern by assigning distinct, core responsibilities to models, views, and controllers and ensuring they interact in accordance to MVC guidelines. The model layer encapsulates data and data-related business logic through immutable Java records such as User, Event, Venue, and Booking. This immutability and encapsulation guarantees clear boundaries for data management, which fulfills the MVC rule that models should manage data without any concern for presentation.

The view layer is implemented using FXML files stored in resources/view/*. MVC describes that views should solely reflect a model's state without manipulating it. As such, my views only describe how model data should be presented (the UI layout) while strictly avoiding embedding any business logic.

MVC describes controllers as an intermediary layer between models and views, ensuring that neither takes on the responsibilities of the other. This is exactly how controllers have been used in my application, where user action is delegated from the view to the controller, the controller processes a model's state and decides on what data should be sent back to be rendered in the view.

How does your code adhere to SOLID design principles?

My program adheres to the **Single Responsibility Principle** via carefully scoped service classes. Each service handles a specific domain of functionality, such as the UserService managing user operations or the BookingService handling booking and commission calculations. This separation extends out to the controller layer, where every controller manages the concerns of a particular view only.

The **Open/Closed Principle** sees the extensive use of interfaces within my service layer, where each service is defined by an interface that specifies its contract, allowing for new, controller-level implementations to be added without modifying existing code.

Liskov Substitution can be demonstrated in my program via the use of controller abstractions. All controllers properly extend a base abstract controller, which maintains consistent behaviour (show errors, warnings, info, etc.) across all controllers.

Interface Segregation is employed via granular service-level interface definitions that prevents controllers from depending on methods they don't use. These interfaces are carefully scoped to specific domains, such as ICSVImporter for csv related operations and IBookingService for booking management.

Dependency Inversion is demonstrated via my ServiceProvider class, which acts as a central dependency management mechanism for all services within my service layer. ServiceProvider enables my controllers to depend on service abstractions rather than directly interfacing with concrete service implementations.

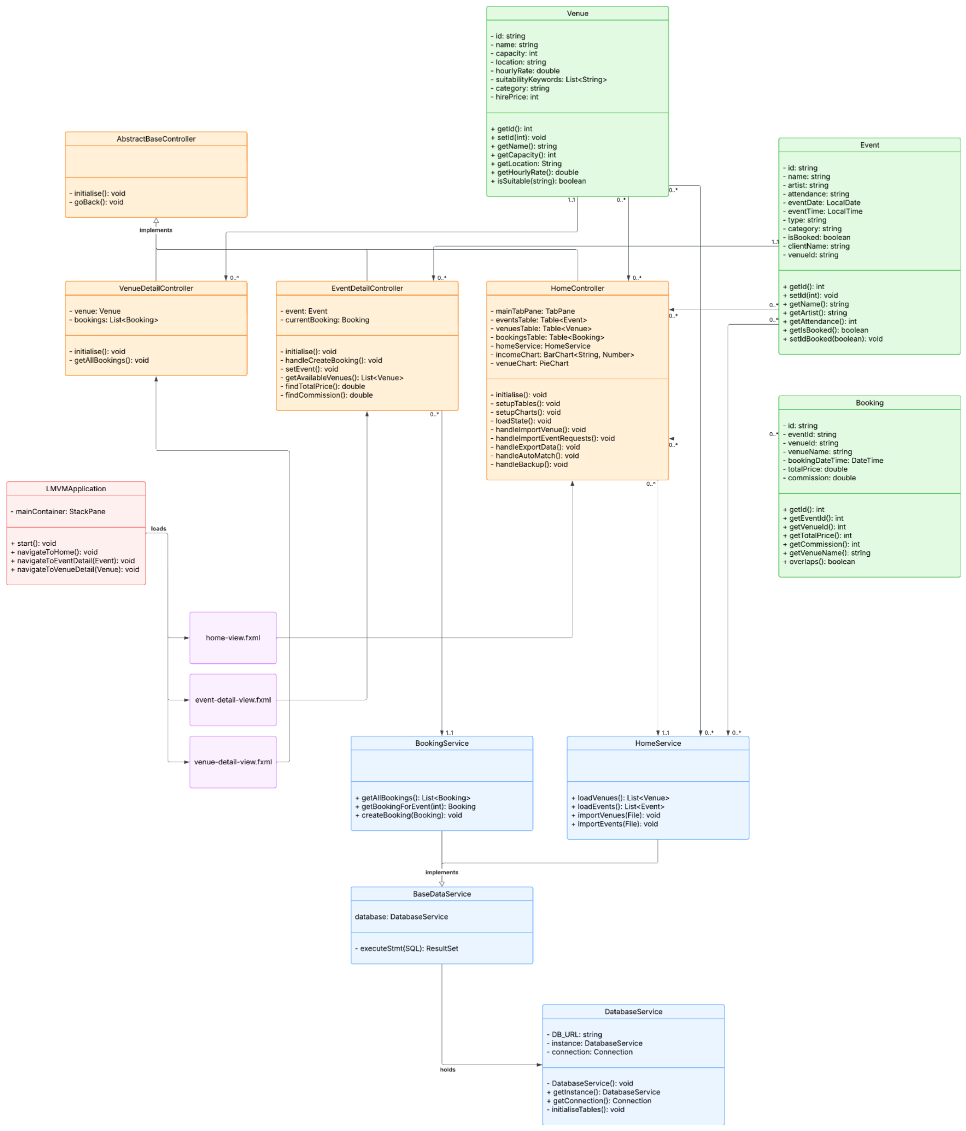
What other design patterns does your code follow? Why did you choose these design patterns?

Another design pattern used extensively throughout my codebase would be the factory (specifically provider) pattern. The ServiceProvider class handles and manages the initialisation of the numerous forms of services within my program. This allows for my controllers to depend on a single, unified interface to access any service related methods. Such an abstraction aids in modularising my codebase by enforcing consistent API and access boundaries, which further promotes code reuse and maintainability. Further patterns, such as singletons, were used in areas deemed appropriate, such as the DatabaseConnection. It is important for a program to handle database connections properly, as mismanagement can lead to resource leaks and expensive loading times. In most cases, a program should only hold a single connection to the database and this connection is best represented as a singleton. This is because clients do not expect to worry about opening/handling the physical connection to a database, every time they require a DatabaseConnection instance.

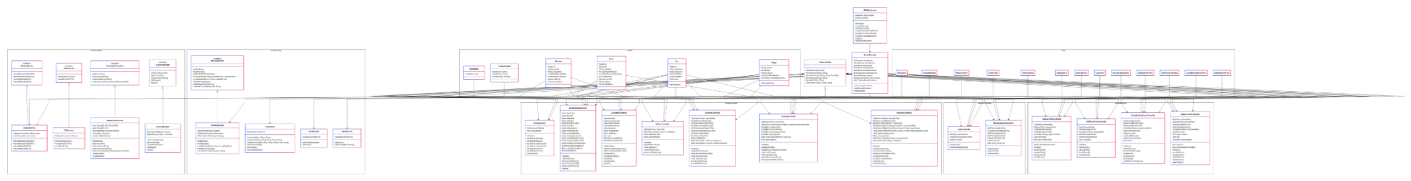
Class Diagram

Given the size of my program, the actual class diagram is very large. For this reason, I have attached two diagrams. The first is the shortened version of the full diagram which demonstrates my program flow, general class relationship and inheritance patterns. The second is the full diagram written in mermaid, it includes all classes and necessary fields and methods.

The diagram below is colour coded to demonstrate the MVC pattern. Yellow is controller, green is model, blue is service, purple are view and red is the root application. [High resolution link](#).

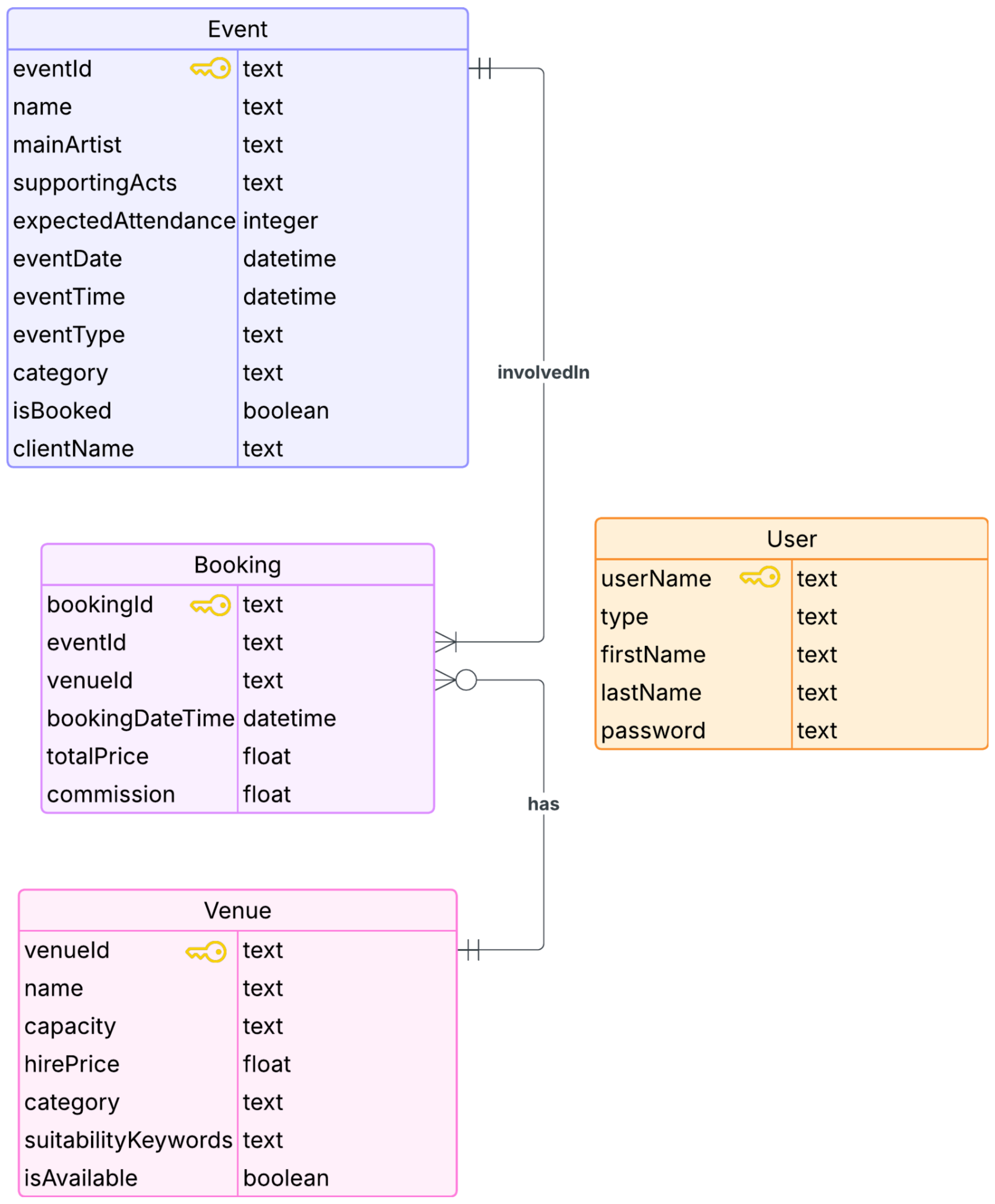


The full diagram is presented below. The source code to this diagram is under diagrams/class.mermaid in the repository. [High res link](#).



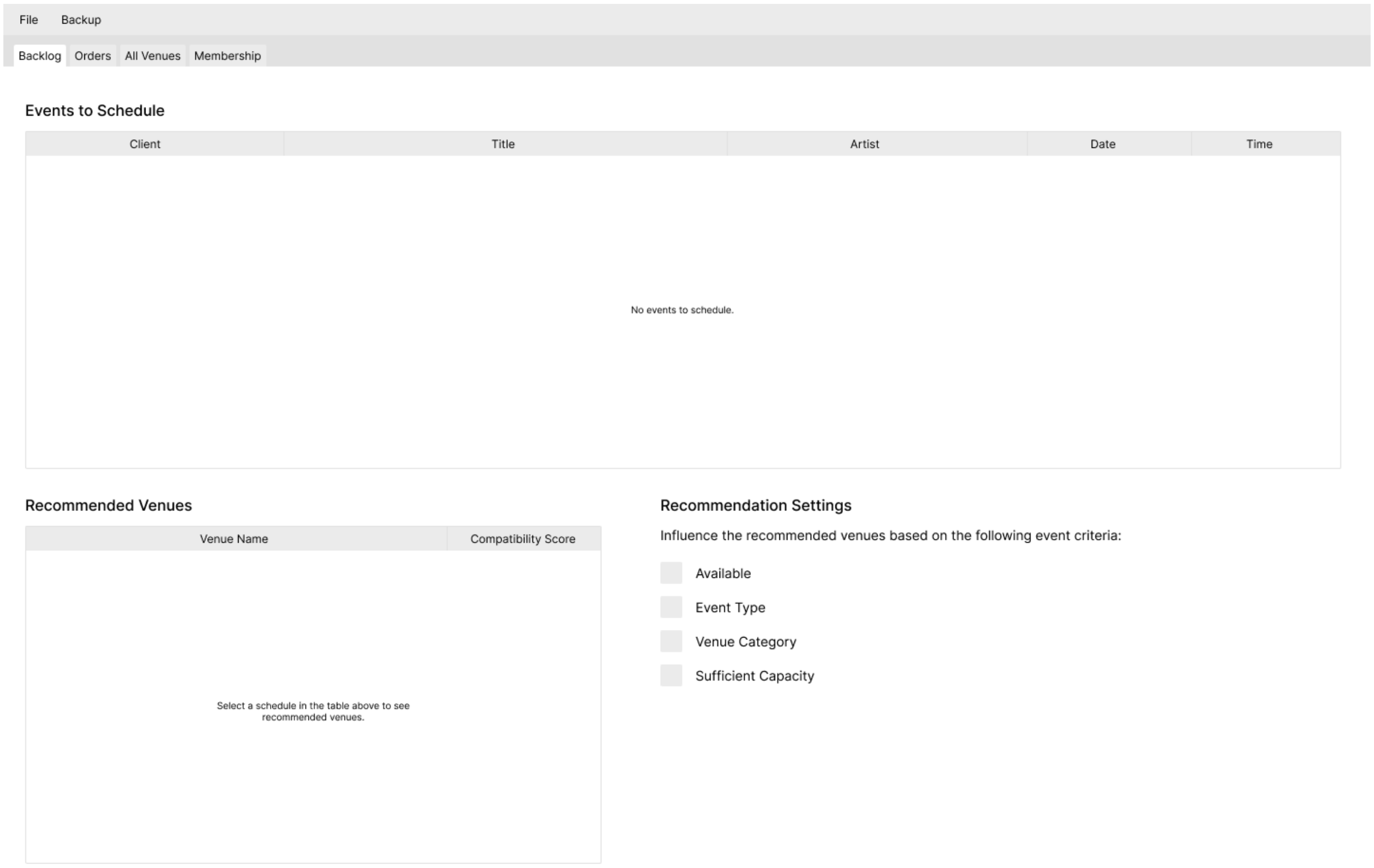
Database Diagram

Below is a database diagram that demonstrates my program’s data-relationships within my application’s database.



UI Design Files

The following are a collection of UI sketches (wireframes) representing the general layout of my program.



Events to Schedule

Client	Title	Artist	Date	Time

No events to schedule.

Recommended Venues

Venue Name	Compatibility Score

Select a schedule in the table above to see recommended venues.

Recommendation Settings

Influence the recommended venues based on the following event criteria:

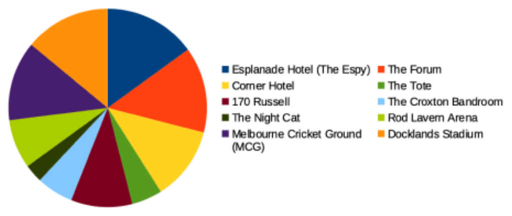
- ☐ Available
- ☐ Event Type
- ☐ Venue Category
- ☐ Sufficient Capacity

File Backup

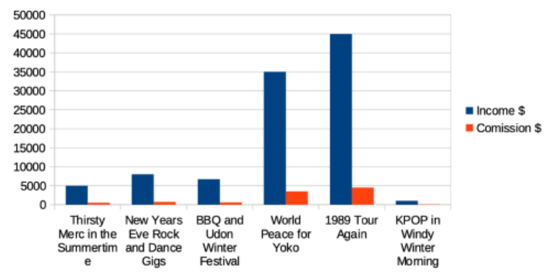
Backlog Orders All Venues Membership

Summary

Venue Utilisation



Income-Commission / Order



Total Commission / Client

Client	Total Commission

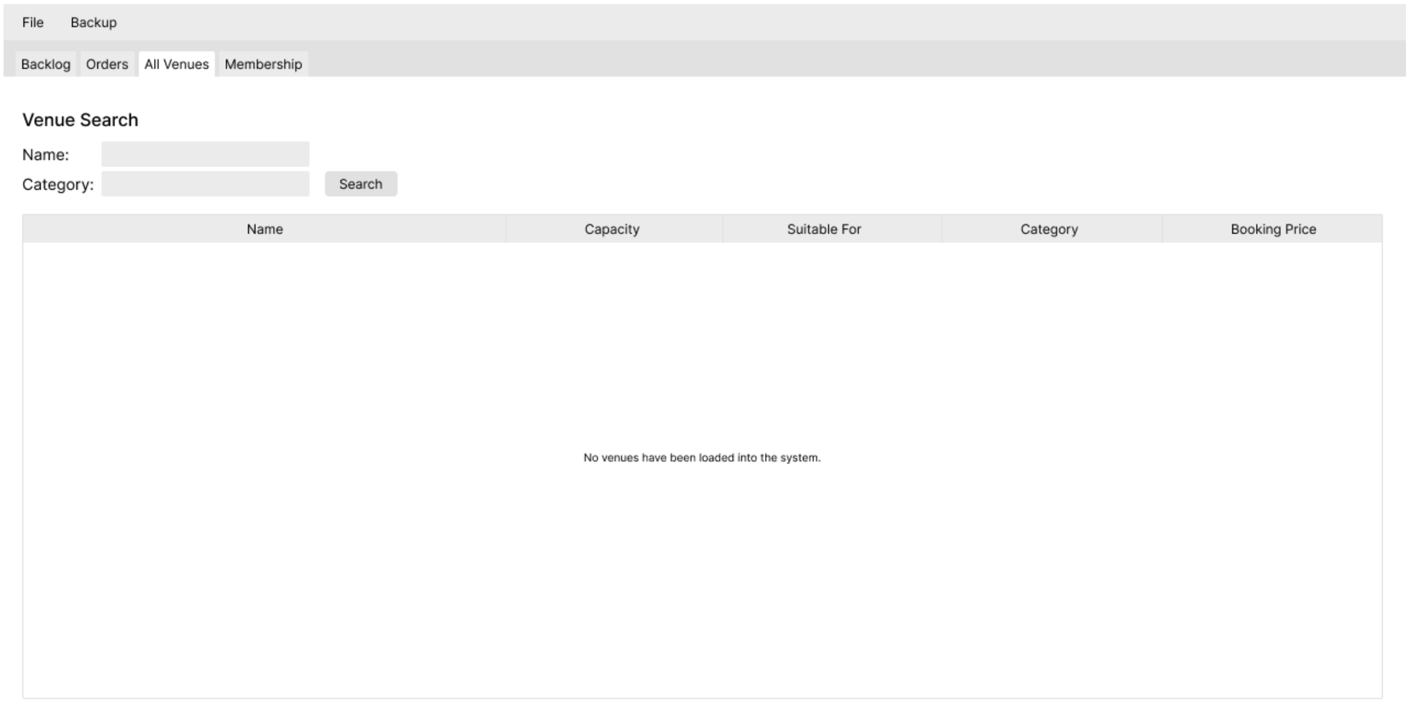
Load in clients or add more bookings to view commission summaries

Total Commission: $xxx,xxx

Order Details

Client	Event Title	Venue Name	Date	Commission

No orders to date.



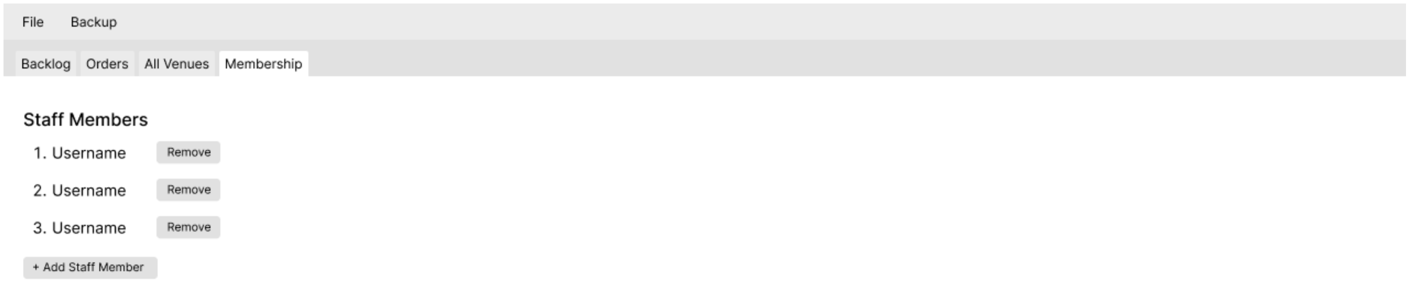
Venue Search

Name:

Category:

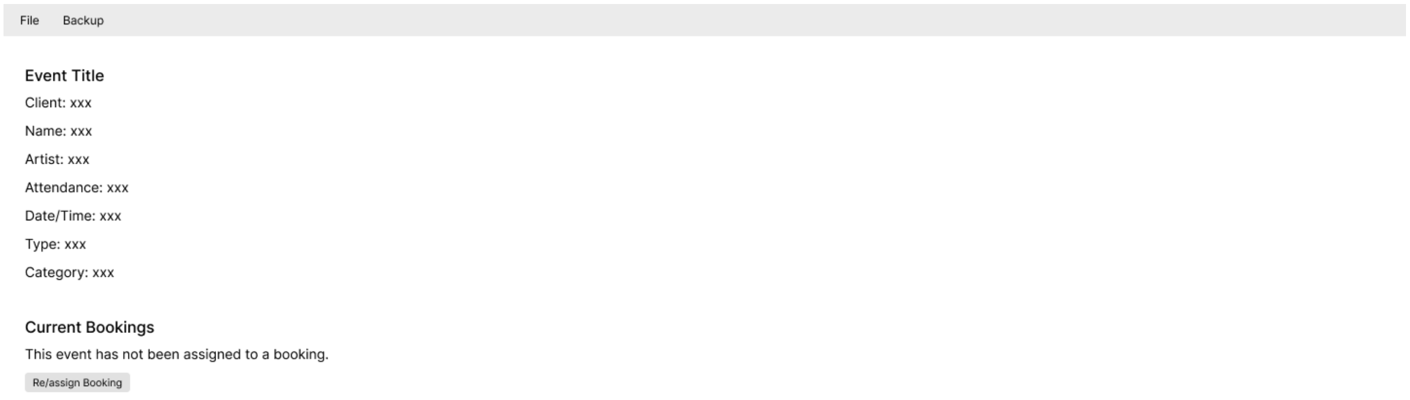
Name	Capacity	Suitable For	Category	Booking Price

No venues have been loaded into the system.



Staff Members

1. Username
2. Username
3. Username



File Backup

Event Title

Client: xxx

Name: xxx

Artist: xxx

Attendance: xxx

Date/Time: xxx

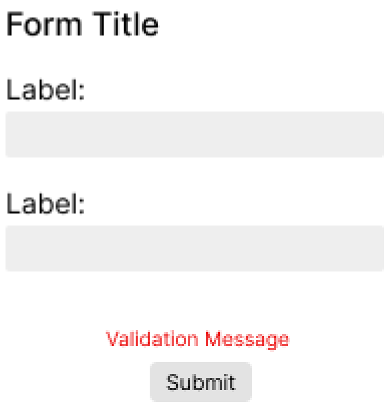
Type: xxx

Category: xxx

Current Bookings

This event has not been assigned to a booking.

Re/assign Booking



Form Title

Label:

Label:

Validation Message

Submit