ST10445866 - Botshelo Koketso Sekwena

IIE MSA RUIMSIG

CLDV6211-Cloud Development 1A

POE Part 3

Table of Contents

[A.) Advanced Filtering 2](#_Toc201517094)

[B.) Azure Deployment Updates 2](#_Toc201517095)

[C.) Reflective Technical Report 3](#_Toc201517096)

[1.) System Feature List: 3](#_Toc201517097)

[1.2) Feature Explanation In context to the Event Ease System: 4](#_Toc201517098)

[2.) Component Discussion: 7](#_Toc201517099)

[2.1 Component Identification: 7](#_Toc201517100)

[2.2 Reasoning Behind Each Component Choice: 7](#_Toc201517101)

[2.3 Discussion On Alternative Components: 11](#_Toc201517102)

[3.) Reflection on project: 12](#_Toc201517103)

[3.1 Personal Perspective on the Project: 12](#_Toc201517104)

[3.2 Lessons learned during the project: 13](#_Toc201517105)

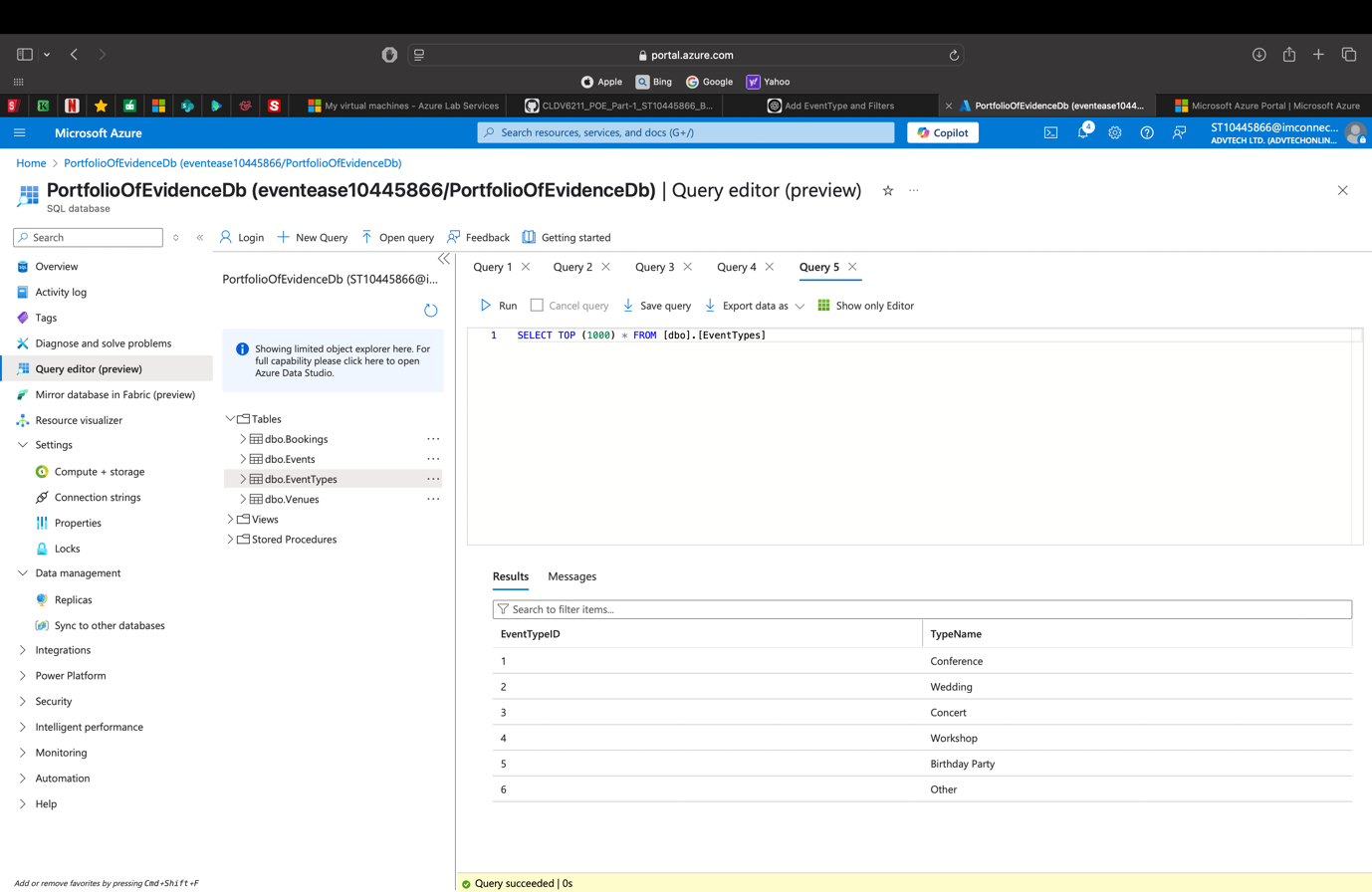
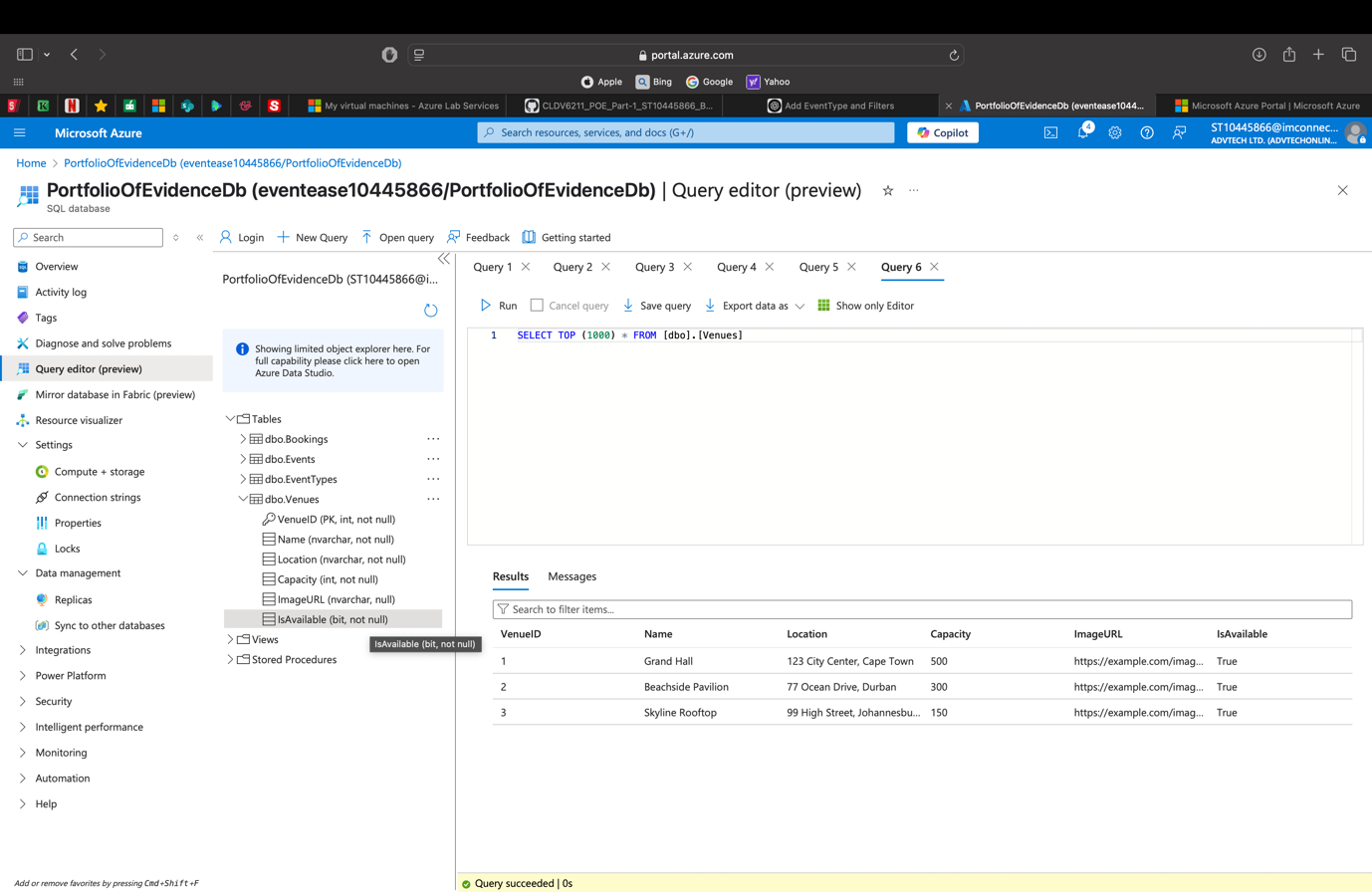
[3.3 My current understanding of Designing, Developing, and Architecting Cloud Based Application: 13](#_Toc201517106)

[Bibliography 15](#_Toc201517107)

# Advanced Filtering

\*Has been completed to the fullest of my knowledge in the project\*

# Azure Deployment Updates

* Web App URL: <https://st10445866-eventeaseapp-ewh6gqagddcnbzen.southafricanorth-01.azurewebsites.net>
* GitHub Link: <https://github.com/SekwenaBotshelo/CLDV6211-Portfolio-of-Evidence-Final-Submission-ST10445866.git>
* Screenshot of newly created Table shown in Query Editor:
* Screenshot of updated Field used from filtering in Venue Table Shown in Query Editor:

# Reflective Technical Report

It is to be noted that the Event Ease Web App was developed using the .NET Framework within Visual Studio, utilising the ASP.NET MVC architecture to structure the application.

## System Feature List:

* 1. System Feature List:

|  |  |
| --- | --- |
| Feature / Requirement | Type |
| Venue Management (CRUD) | Functional |
| Event Management (CRUD) | Functional |
| Booking Management (CRUD) | Functional |
| Prevent Double Booking | Functional and Validation |
| Display Consolidated Booking View | Functional/UI |
| Search & Filter Bookings (By Event Name, Date, etc.) | Functional/UX |
| Event Type Lookup Table | Database/ Functional |
| Venue Availability Toggle | Functional/Validation |
| Navigation via Home Page with Routing | Functional/UX |
| Database Integration using SQL Server | Technical/Back-end |
| MVC Pattern Implementation | Architectural |
| ADO.NET and EF6 Usage | Technical |
| Error Handling for Deletion and Invalid Entries | Non- Functional |
| User input Validation (e.g., Required Fields) | Non- Functional |
| Secure Data Storage in Cloud (Future integration) | Non- Functional/Future |
| Admin Access for Booking Specialties | Non- Functional/UX |

### Feature Explanation In context to the Event Ease System:

1. Venue Management (CRUD): This feature lets users add, edit, read, and remove venue records. A venue's name, address, capacity, availability, and image are all included. managed with Razor's built-in views in the VenuesController.
2. Event Management (CRUD): CRUD functions for events are comparable. Name, date, type (from a lookup table), and notes are all included in events. Foreign keys are used to link to locations and reservations.
3. Booking Management (CRUD): Booking experts are able to oversee client reservations. The location, event, start and end dates, and booking status are all noted for each reservation. incorporated into BookingsController.
4. Prevent Double Bookings: Verifies that a location cannot be reserved for dates that overlap by looking at current reservations before adding new ones. The controller logic or service layer handles validation.
5. Display Consolidated Booking View: This is a single view that uses a SQL View (vw\_BookingManagement) to combine data from Venues, Events, and Bookings. For convenience of UI presentation, a view model was used for display.
6. Search & Filter Bookings: Using arguments in the sp\_SearchBookings stored procedure, users can apply filters by Event Type, Date, and Venue Availability in addition to searching bookings by Booking ID or Event Name.
7. Event Type Lookup Table: By keeping reusable event types in a different database called EventsTypes and connecting it to the Events table using a foreign key called EventTypeID, the Event Type Lookup database normalises the event type data.
8. Availability of the Venue Switch: The Venues table has a boolean column called IsAvailable that can be changed to show if a venue is currently available for reservations. utilised in search functionality as a filter.
9. Getting Around using the Home Page: Links to About, Contact, Events, Venues, and Bookings are included on the easy-to-use Home page. implemented using routing in the \_Layout.cshtml and Views/Home/Index.cshtml files.
10. Database Integration with SQL Server: An Azure-hosted SQL Server database is linked to the application. Venues, Events, Bookings, and EventTypes are among the tables. accessible with EF6 or ADO.NET.
11. Implementation of the MVC Pattern: The application divides concerns into three categories: views for user interface, controllers for functionality, and models for data using the Model-View-Controller architecture.
12. Usage of ADO.NET with EF6: EF6 manages fundamental ORM operations like querying and saving entity data, whereas ADO.NET is utilised for raw SQL commands and stored procedures (such as for search).
13. Error Handling for Deletion and Conflicts: Keeps locations or events mentioned in ongoing reservations from being deleted. ModelState is used to display error messages in a gentle manner.AddModelError.
14. User input validation uses both controller-side logic and data annotations, such as [needed], to make sure all needed fields are filled out before submitting the form.
15. Future Scope for Secure Cloud Data Storage: aims to grow the application by using Azure App Service to host it and protecting data in Azure SQL with the right firewall and role-level permissions.
16. Booking Specialists' Admin Access: All CRUD functions and booking administration tools are made for booking specialists who serve as system administrators. Role-based access control (RBAC) may be incorporated into future updates.

## Component Discussion:

### Component Identification:

|  |  |
| --- | --- |
| Component | Technology Used |
| Web Framework | ASP.NET Framework (MVC Pattern) |
| IDE | Visual Studio |
| Language | C# |
| ORM/Data Access Layer | Entity Framework 6 (EF6) and ADO.NET |
| Database | SQL Server (Hosted on Azure) |
| Application Architecture | Model-View-Controller (MVC) |
| Views/UI Layer | Razor Pages (.cshtml) |
| Data Validation | Data Annotations and Server-Side Validation |
| Stored Procedure and Views | SQL Server T-SQL |
| Hosting (in progress/future) | Azure App Service |
| Version Control | GitHub |

### Reasoning Behind Each Component Choice:

1. ASP.NET Framework(MVC Pattern):

Due to compatibility issues and Visual Studio tooling preferences, the system was developed using the full.NET Framework version of ASP.NET Core Web Application (MVC), even though the original plan called for using MVC. Because of the well-organised separation of responsibilities provided by ASP.NET MVC (Model, View, Controller), business logic, data, and display can be managed effectively.  
Reasons for selection:

* Sturdy and mature structure
* Complete compatibility with EF6 and Visual Studio
* Knowledge and the accessibility of resources
* Ideal for online applications in education and business

1. C# Programming Language:

Utilised for data processing, modelling, and controller logic across the application.

Reasons for selection:

* Object-oriented and heavily typed
* Complete assistance within the.NET environment
* Outstanding Visual Studio tooling

1. Entity Framework 6 (EF6):

EF6 was utilised for common database functions like saving and accessing information about events, venues, and reservations. It supports LINQ queries and offers an interface over raw SQL.

Reasons for selection:

* Streamlines CRUD processes
* Reduces boilerplate and enables quick development
* Strongly supported by the.NET Framework

1. ADO.NET:

When combined with EF6, ADO.NET is particularly useful for accessing views (vw\_BookingManagement) and running stored procedures (sp\_SearchBookings), providing greater control over sophisticated SQL logic and performance.

Reasons for selection:

* Permits running SQL queries in their raw form.
* Perfect for searches with high performance
* Enhances EF6 when certain queries are required.

1. SQL Server (Azure-hosted):

Utilised as the relational database system's backend for managing and storing information about events, venues, and reservations.

Reasons for selection:

* Smooth interaction with ado.net and.net
* Supports SQL views and stored procedures.
* Azure-hosted for cloud readiness and scalability

1. Razor Pages (.cshtml):

Used to produce dynamic web pages (such as venue lists and booking forms) for user involvement. Razor syntax seamlessly combines html and c#.

Reasons for selection:

* Simple and uncluttered syntax
* Integrating the MVC model is simple.
* Completely compatible with Visual Studio

1. Stored Procedures and Views:

Effective filtering and aggregated data presentation were made possible by the use of the custom stored procedure sp\_SearchBookings and view vw\_BookingManagement.

Reasons for selection:

* Improves the efficiency of search operations
* Minimises the app's logic duplication
* Makes the controller and view code simpler.

1. Azure App Service (Planned Deployment):

The system is presently being built locally, but for scalability and real-world application, it is meant to be deployed to Azure.

Reasons for selection:

* Cloud-based, safe, and scalable
* Integrates with SQL Azure with ease
* Perfect for deployments in small businesses and education

1. GitHub:

Used for collaboration, version control, and change tracking.

Reasons for selection:

* Source control industry standard
* Allows for cooperation and history tracking.
* Visual studio integration is simple.

The EventEase Web App's present version makes use of tested and compatible.NET technologies designed for quick construction and scholarly demonstration. The choices were motivated by the.NET Framework's stability, familiarity, and interoperability with Visual Studio. The system is still extendable, though, and in later versions it might switch to ASP.NET Core MVC to benefit from cross-platform deployment, cloud-native features, and contemporary performance enhancements.

### Discussion On Alternative Components:

|  |  |  |
| --- | --- | --- |
| Component | Technology Used | Reasonings |
| Web Framework | ASP.NET Core MVC | Better performance, cross-platform compatibility, lightweight design, and modularity |
| ORM/Data Access | EF Core | A more recent EF version that is ASP.NET Core optimised |
| Database | SQLite/PostgreSQL | Cross-platform choices for open-source or lightweight databases |
| UI Layer | Blazor/React | For a richer client interface, SPA technologies could take the place of Razor Pages. |
| Stored Procedure | EF LINQ/Fluent API | May use stored procedures to accomplish the same logic in code. |
| Hosting | Dockerized deployment | For contemporary, microservices-based, containerised hosting on Azure or another platform |

## Reflection on project:

### Personal Perspective on the Project:

The approach appeared to be rather simple at the beginning of the project, and I didn't feel like I was learning anything new. However, I started to understand the intricacy and depth required to create a full-stack, cloud-capable application as I advanced into the initial implementation phases, especially those requiring data modelling, cloud integration, and the MVC architecture.

Understanding the interactions between all the parts of a cloud-based system, particularly with regard to database connectivity, booking validations, and integrating stored procedures within the ADO.NET and EF6 frameworks, was one of the most difficult things I had to do. Even at the end of the project, I ran into small problems that needed thorough troubleshooting, and there were moments when debugging errors or database connection problems severely hindered development.

In spite of these difficulties, the trip was instructive and fruitful. Every error or challenge turned into a teaching moment that enhanced my technical knowledge and sharpened my problem-solving skills. In spite of these difficulties, the trip was instructive and fruitful. Every error or challenge turned into a teaching moment that enhanced my technical knowledge and sharpened my problem-solving skills.

### Lessons learned during the project:

I learnt from this project that creating a cloud-based web application required knowledge of system architecture, data integrity, and deployment environments in addition to writing code. I obtained first-hand knowledge of:

* Using both Entity Framework and ADO.NET to link a SQL Server database to a.NET web application.
* To improve efficiency and streamline backend logic, stored procedures and views are created.
* creating a structure based on MVC that divides tasks and enhances maintainability.
* Using form inputs and CRUD processes to comprehend error handling, validation logic, and user experience.

Most significantly, I discovered that operating a system of this size requires regular version control and comprehensive documentation. I was able to monitor my work and modifications using GitHub, and having comprehensive documentation during implementation helped to avoid misunderstandings and mistakes.

### My current understanding of Designing, Developing, and Architecting Cloud Based Application:

This project has given me a solid fundamental grasp of the requirements for designing, developing, and architecting cloud-based apps. I now realise that:

* Design approaches for cloud-based systems must be secure, scalable, and adaptable.
* New factors like environment configuration, connection strings, and access controls are introduced when hosting a web application in the cloud (for example, with Azure App Service).
* Although Database-as-a-Service (DBaaS), such as Azure SQL, makes infrastructure deployment easier, it necessitates careful application integration.
* When creating cloud-based systems for future scalability and maintainability, the MVC pattern and separation of concerns are crucial.

This project has set a strong foundation, but I still have a lot to learn, particularly in areas like role-based access control, automated CI/CD pipelines, and sophisticated cloud security. With a more knowledgeable, realistic, and solution-focused approach, I am now more assured that I can handle comparable cloud projects in the future.

(Andrew Troelsen & Phil Japiske, 2022) (Kamil Mrzygłód, 2022) (OpenAI, 2025)

# Bibliography

Andrew Troelsen & Phil Japiske. (2022). *Pro # 10 with .NET 6 - Foundational Principles and Practices in Programming.* New York: Apress.

Kamil Mrzygłód. (2022). *Azure For Developers - Implement rich Azure Paas ecosystems using containers, serverless services, and storage solutions.* Birmingham: <packt>.

OpenAI. (2025, April 7). *OpenAI*. Retrieved from OpenAI: https://chatgpt.com