**INTRODUCTION**

Event Ease is a Microsoft Azure-hosted ASP.NET Core MVC application for booking venues. This report delineates the application's iterative development over three phases, focusing on architectural choices, incorporation of Azure services, and primary insights into cloud-centric app development.

**Application Feature Overview**

Part 1 Foundation:

ERD Diagrams

Features- CRUD operations for Venues/Events/Bookings

Placeholder image URLs

Azure App Service deployment

Part 2 Enhancement:

Azure Blob Storage for images

Double-booking validation

Consolidated booking view with search

Part 3 Advanced:

Event type classification (Conference/Wedding/Concert)

Filtering by date range/availability

Cognitive Search integration

**Azure Services Implementation**

| Components: |  |  |  |
| --- | --- | --- | --- |
| Web hosting |  |  |  |
| Database  Storage  Search |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Azure services:   
App service

SQL database

Blob storage

Cognitive search

Purpose:

Zero-downtime deployment via GitHub Actions

Transactional integrity for bookings

Cost-effective image storage ($0.02/GB)

Semantic event search

**Technology Stack Analysis**

ASP.NET Core MVC:

* Razor views for dynamic rendering ui interchangeably display users information from a system.
* Using Tag Helpers, a user can validate and check errors in their inputs to the given forms.

Entity Framework Core:

* Using the Code First approach, one can design databases in Entity Framework and perform electronic migrations.
* A complex database on an application can also be executed and captured using LINQ Queries.

Azure SDK:

* Files and documents relevant to any organization can easily be stored by use of Azure.Storage.Blobs
* Cognitive sea was used for azure based searches along with the other many uses it has been put to

**Reflection Development Journey**

**Design Principles**

Normalization vs. Performance:

The original 3NF design led to excessive JOIN queries. Implemented denormalized BookingDetails view to enhance search optimization.

**Development Challenges**

Uploading Images:

UI freezes were avoided through asynchronous blob uploads.

Validation logic:

Users were prevented from making errors with a double booking check test.

**Deployment Lessons**

Upgrading databases remotely:

Achieved zero-downtime updates using EF Core migrations and ApplyMigration during application startup.

Security concerns relating to data:

After the initial risk of exposure, connection strings were securely kept in Azure Key Vault.

**Conclusion**

Azure's capability to manage scale web applications was shown via the EventEase project.

Important takeaways:

* Infrastructure overhead is minimized with serverless frameworks
* Validation in cloud-native environments is limited by dependency on distributed transaction validation
* Cost performance ratios can be optimized with tiered storage solutions

**Referencing**

**H. Thompson, "Cloud Database Design Patterns," *IEEE Transactions on Cloud Computing*, vol. 12, no. 3, pp. 45-59, 2024.**

**[Accessed: 20 June 2025]**

**C. Richardson, *Microservices Patterns*. Manning Publications, 2020.**

**[Accessed: 20 June 2025]**

**"ASP.NET Core Performance Best Practices," *Microsoft Developer Network*, 2025. [Online]. Available:** [**https://learn.microsoft.com**](https://learn.microsoft.com)

**[Accessed: 20 June 2025]**

**Github Link:** https://github.com/Modies20/EventsEase.git