**Reflective Technical Report for EventEase App**

The creation of the EventEase application proved to be an essential turning point in my development as a full-stack developer, particularly inside the Azure environment. EventEase is a venue and event booking platform based on ASP.NET Core MVC and developed with scalability, modularity, and cloud-first principles in mind. This reflective report covers the major milestones, choices, and lessons learned throughout the project. Azure Web App, SQL Database, and Azure Storage were critical components in shaping the app's architecture, deployment, and functionality. This paper assesses their contributions and how each service affected my design thinking, development workflow, and final implementation approach.

Integration of Azure Services

1. Azure Web app

The EventEase application was hosted in the cloud using Azure Web App. Azure Web App was chosen due to its simplicity of integration with Visual Studio, built-in continuous deployment support, and ability to manage scalability, performance, and availability automatically (Microsoft, n.d.)

. The deployment method entailed setting the CI/CD pipeline with Visual Studio's Publish functionality and assuring environment consistency between development and production. (Mikejo5000, n.d.)

This hosting option removed the requirement for manual server management and enabled secure and rapid deployment of the ASP.NET Core application. It mandated HTTPS as the default setting, requiring me to modify all routes and static resources to ensure compatibility with secure protocols (Microsoft, n.d.). A significant takeaway was grasping how web.config configurations and environment-specific appsettings.json files influence deployment.

2. Azure SQL database

Venue information, user reservations, and event categories were among the relational data in the application that was stored and managed using an Azure SQL Database. LINQ queries and code-first migrations made data interactions easier through integration with Entity Framework Core. (Microsoft, 2021)  
  
Performance is ensured by appropriate indexing and normalization, particularly when filtering and searching, as demonstrated by the use of Azure SQL Database. I discovered that foreign keys may be used to create associations between tables, such as connecting reservations to users and locations. Using seed data for startup and maintaining version control of the database schema were also key components in managing migrations between environments.

Security was a prominent area of emphasis; I learned about encrypted keys and secure access policies from Azure portal setups and managing connection strings using user secrets (Microsoft, 2024). The dependability and accessibility of Azure SQL also allowed me to concentrate less on database uptime and more on reasoning.

3. Azure Storage

Although complete deployment is still ongoing, Azure Blob Storage was incorporated at the planning phase to handle media assets, including promotional files, venue paperwork, and event photos. Secure access control, scalability, and high availability are all provided by Azure Blob Storage. (tamram, 2022)  
  
The goal is to free up server resources and allow users to efficiently upload and view files by offloading static file delivery to Blob Storage. As part of my preparation, I experimented with creating SAS tokens and utilizing the Azure.Storage.Blobs SDK to safely provide temporary access to particular files. This integration helped me plan out the backend's reactive file upload functionality and organize media-related models. (Conejo, 2023)

Understanding the blob lifecycle, managing blob containers, and putting content type checks and storage tiers into practice were among the main lessons learned. Despite this feature's continuous existence, architectural planning was greatly impacted by its expected impact.

List of systems features

An extensive range of administrative and user operations are supported by the EventEase program. The characteristics of the system and their functions are listed in detail below:

|  |  |
| --- | --- |
| Feature | Description |
| User Registration/Login | Enables safe account creation and login for users. Multi-role access control and future OAuth are supported by the system, even if fundamental identity scaffolding is still lacking. |
| Event Booking System | A central system that lets users choose locations, define the kind of event and the range of dates, and verify reservations. The SQL Database is updated with verified items when availability is verified. |
| Search and Filtering | Uses query logic to allow users to filter venues by location, date range, and event type. indexed queries and LINQ optimization. (Microsoft, 2021) |
| Venue Management | Admins or venue owners may add, edit, and remove venue records, such as capacity, price, and availability periods, using a back-end system. |
| Event Type Lookup Table | A standardized table that holds event types (e.g., conference, wedding). guarantees uniform filtering and classification choices for all reservations. |
| Date Range Selection | When booking, users choose precise start and finish dates. The backend logic enforces business rules and makes sure there are no overlapping reservations. |
| Availability Status | Every location has a scheduling matrix or flag that shows when it is available. This is constantly updated to reflect current availability following each reservation. |
| Admin Portal | Bookings, user roles, availability overriding, and the creation of reports on event trends and times of high demand are all done through the admin dashboard. |
| Azure Deployment | The app is linked with an Azure SQL backend and published to Azure Web App (Microsoft, n.d.). Secure HTTPS routing, remote monitoring, and cloud scalability are guaranteed. |
| Blob Upload (Planned) | Allows data and photographs linked to events and venues to be uploaded and retrieved using Azure Blob Storage (tamram, 2022)  makes media distribution and bandwidth more efficient. |

Architectural Decisions

The main goals when developing the program were scalability and modularity. Cloud-first design best practices informed a number of architectural choices:

**1.** Distinguishing Issues  
Using the Model-View-Controller (MVC) design, the application was neatly divided into:

* data structures and logic models,
* perspectives for displaying user interfaces,
* controllers to handle and coordinate requests.  
  This facilitates testing and improves scalability and maintainability. (Microsoft, 2021)

2. Cloud-Native Integration  
Among the Azure services that impacted the application's structure and data flow were:

* Regarding relational data, Azure SQL
* Storage of files in blobs (tamram, 2022)
* For pipelines for deployment, use GitHub Actions or Azure DevOps.

3. Security by Design

Early in the development process, security guidelines were integrated:

By default, HTTPS is used. (Microsoft, n.d.)

* Environment-specific and safe connection strings. (Microsoft, 2024)
* Future deployment of role-based access control (RBAC) is planned.

4. Durability in scalability.  
To guarantee that the app may grow without performance deterioration,  
Azure SQL has been chosen for scalable relational storage and Blob storage is designed to handle enormous amounts of unstructured data. (tamram, 2022)

5. The setting Awareness  
Configuration options were environment-specific:

* appsettings.Development.json for local/dev settings.
* Inclusive production configurations to bypass deployment conflicts and improve operational security. (Mikejo5000, n.d.)

6. Ensuring the future  
The system is built for simple expansion, the administrator dashboard and role management are scaffolded and analytics modules and other sophisticated features may be implemented with little reworking.

In conclusion, the EventEase solution not only improved my technical skills in.NET Core and Azure cloud services, but it also helped me progress my architectural thinking dramatically. Working through deployment, data integration, and cloud scalability taught me how to create solutions that are both user-centric and performance-focused. (Microsoft, n.d.)  
  
Azure's ecosystem offered a solid foundation that alleviated infrastructure concerns, enabling me to concentrate on creating valuable features. Each service—Web App, SQL Database, and Storage—played an important part in determining the app's performance, scalability, and growth preparation.

This experience gave comprehensive end-to-end exposure, from local creation to live deployment, and created the framework for future enhancements like as authentication, admin control, media storage, and sophisticated analytics. EventEase is more than simply a technical tool; it's a step towards understanding full-stack cloud development and creating effective, scalable apps.

References

Microsoft (n.d.). *Azure App Service documentation - Azure App Service*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/azure/app-service/>. [Accessed 23 Jun. 2025].

Microsoft (2021). *Overview of Entity Framework Core - EF Core*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/ef/core/>. [Accessed 23 Jun. 2025].

Microsoft (2024). *Azure encryption overview*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-overview>.

[Accessed 23 Jun. 2025].

Mikejo5000 (n.d.). *Publish an ASP.NET web app - Visual Studio (Windows)*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/visualstudio/deployment/quickstart-deploy-aspnet-web-app?view=vs-2022&tabs=azure>.

[Accessed 23 Jun. 2025].

Conejo, M. (2023). *Exploring the Power of Azure Lab Services API for Lab Management and Deployment*. [online] Medium. Available at: <https://medium.com/@marvinconejo/exploring-the-power-of-azure-lab-services-api-for-lab-management-and-deployment-1120b643a72f>

[Accessed 23 Jun. 2025].

tamram (2022). *About Blob (object) storage - Azure Storage*. [online] learn.microsoft.com. Available at: <https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview>.

[Accessed 23 Jun. 2025].