Final CLDV6211 POE

ST10443218

Ushay Madhav

[Year]

Screen shots

Updated SSMS

A screenshot of a computer

AI-generated content may be incorrect.

Code from Visual Studio(Filtering by Date, Availability etc)

These are just snippets, cant include all because it wont be readable but my code has been submitted , so you may check all code for functionality

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Azure deployment

I wasn’t able to deploy , it is giving me errors of my subscription being restricted by policy and only allows creating App Service Plans with basic or shared SKUs, I tried both South East Asia and South Africa North as locations because those are the cheapest, therefore I am unable to produce a link ending in .net like my old one (<https://eventease20250406151141.azurewebsites.net>) this resource has been deleted by VC. I'm not certain, but it's possible that VC has implemented broader restrictions on resource creation, potentially as a cost-saving measure. I was also unable to deploy my database, which suggests that the limitations may apply across all resources.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# C. Reflective Technical Report

A comprehensive list of specific system features and requirements listed

* Venue Management: CRUD operations for venues with image upload.
* Event Management: CRUD functionality for events, linked to venues, with image support.
* Booking System: CRUD operations with double booking prevention logic.
* Validation and Error Handling: Validation messages for required fields, booking conflicts, and rules for image upload.
* Search and Filter: Bookings can be filtered by Booking ID or Event Name in EnhancedRead.cshtml.
* Blob Storage Integration: Images for venue and events are kept in Azure Blob Storage.
* Azure SQL Integration: Data is kept in an SQL database that is in the cloud via Azure SQL.
* Azure Web App Hosting: Hosting of applications is on an Azure Web App service.
* Views and Reporting: A BookingDetails SQL View rolls up and displays joined data in all tables.
* Image Validation: Maximum file size and allowed extension validation is enforced for uploading.

Brief explanation of how each feature/requirement works in the context of the system

**Azure Blob Storage for uploading photos**

How it works: AzureBlobHelper.cs processes the files people upload to the server when they post Create.cshtml or Update.cshtml to upload photos to Azure Blob Storage.

The uploaded file is processed by ASP.NET Core's IFormFile method.

To prevent conflicts, AzureBlobHelper creates a new filename and uploads the file using the Azure Storage SDK (BlobClient).

The database stores the returned Blob URL (e.g., in an ImageUrl field).

**ASP.NET Core MVC layout**

Models: Declare data structures with validation attributes, e.g., Event, Venue, and Booking.

Views: Display user interface and bind to model data using Razor files (e.g., Create.cshtml).

Controllers: Perform CRUD operations (BookingController) and HTTP requests.

Dependency Injection: Services like AzureBlobHelper are injected into controllers for loose coupling.

**SQL View (BookingDetails)**

Purpose: Simplifies querying joined data (e.g. shows booking information alongside event/venue information).

Implemented in SQL Server as a view that joins Bookings, Events, and Venues.

Mapped onto a read-only view model in ASP.NET Core (e.g. BookingDetailsViewModel).

Used within admin views to present aggregated data without repeated joins in code.

**Validation Logic**

Custom Attributes:

[FutureDate]: Ensures that event dates are not historic dates (applied to Event.Date).

[MaxFileSize(5 \* 1024 \* 1024)]: Restricts to 5MB (prevents large files).

Client Side Validation: ASP.NET Core includes jQuery Validation out of the box and automatically displays errors before submission.

**Prevention of Double Booking**

BookingController Logic:

Before booking, the action verifies from the database if the venue has been booked on the same date or not.

In event of conflict, it returns a validation error (e.g., "This venue is already booked on the selected date").

Database Level Safeguards: It could also be enforced by a unique constraint on VenueId + Date.

**Deployment in Azure Web App**

Scalability: Handles traffic amount automatically through VC policy pricing (e.g. Basic/Standard plans).

Integration: Seamlessly integrates with Azure SQL and Blob Storage using configured connection strings.

**Search Functionality**

Implementation:

EnhancedRead.cshtml accommodates a search form (e.g., for venue titles or dates).

The controller uses LINQ to narrow down BookingDetails results

var query = \_context.BookingDetails.AsQueryable();

if (!string.IsNullOrEmpty(searchTerm)) {

query = query.Where(b => b.VenueName.Contains(searchTerm));

}

Dynamic Queries: Can be extended to date ranges or dropdown select.

**Azure SQL Database**

Relationships:

Foreign keys like Booking.EventId link tables.

Entity Framework Core handles migrations (e.g., dotnet ef migrations add InitialCreate).

Performance: Indexes on frequently queried columns (e.g., VenueId) accelerate searches.

**Image Display in Views**

Rendering:

<img src="@Model.ImageUrl" /> in views this directly links to the Blob Storage URL.

Blob URLs are public (or use SAS tokens for private blobs).

Fallbacks: Placeholder images can be shown if ImageUrl is null.

**Alerts and Errors with TempData**

Example Flow:

If I try to delete a venue with active bookings, the action checks dependencies:

if (\_context.Bookings.Any(b => b.VenueId == id)) {

TempData["Error"] = "Cannot delete: Venue has active bookings.";

return RedirectToAction("Index");

}

UI Feedback: Views display TempData["Error"] as an alert.

D. Component Discussion

**Azure Web App (Hosting)**

Why Chosen?

Fully Managed: No server maintenance needed; auto-scaling.

Seamless Deployment: Direct integration with GitHub,Azure and Visual Studio.

Cost-Effective: Pay only for consumed resources (best for small-to-medium traffic) and this software is free and convenient for students.

Alternative Considered:

Azure Static Web Apps (Better for pure frontend applications with serverless APIs).

Azure Kubernetes Service (AKS) (Overkill for this project but helpful for microservices).

Why Not Chosen?

Static Web Apps do not provide backend functionality needed for this MVC system.

AKS introduces unnecessary complexity to a monolithic application. (DigitalOcean, 2025).

**Azure SQL Database (Data Storage)**

Why Chosen?

Relational Structure: Perfect for bookings, events, and venues (foreign key relationships).

Links with my web app on azure, any changes to DB immediately affects my web app functionality

Azure SQL Database provides built-in scalability and high availability without requiring user maintenance (QA, 2023).

High Availability: 99.99% SLA with automatic backups.

Security: Built-in encryption and firewall protection.

Alternative Considered:

Azure Cosmos DB (If the system needed NoSQL flexibility).

PostgreSQL on Azure (Open-source alternative with similar features).

Why Not Chosen?

Cosmos DB is more expensive and not needed for structured relational information.

PostgreSQL was not chosen due to familiarity with SQL Server.

**Azure Blob Storage (Image Uploads)**

Why Chosen?

Already used azure services for my web app and database might as well stick with azure and use their storage so all components of the project can be easily monitored on one service

Blob storage is more suitable for non-structured data like images, file storage is better for structured, file-system-based access (Storage, 2024).

Cost-Effective: More cost effective than image storage in SQL and locally on a device.

Scalability: Accommodates thousands of image uploads without loss of performance.

Security: SAS tokens grant temporary access to private images.

Alternative Considered:

Local SSD

Azure Files (If shared storage between VMs was needed).

Azure CDN (For fast global image delivery).

Why Not Chosen?

SSD- Takes a lot of space and SSD is expensive.

Azure Files is an overkill since the app is running in a single Web App instance.

CDN wasn't necessary since most of the users are local.

Development Technologies

**ASP.NET Core MVC (Framework)**

Why Chosen?

MVC Pattern: Clean separation of concerns (Models, Views, Controllers), if we chose a normal C Sharp application it wouldn’t be able include all the needs of the POE

ASP.NET Core is more lightweight and cross-platform compared to the older ASP.NET framework (Pehlivanov, 2022).

Built-in Security: CSRF protection, model validation, and auth support.

Azure Integration: Works well with Azure services and VC pays for azure so we have to use it

Alternative Considered:

Blazor (Server/WASM) (For a more interactive UI).

Node.js + Express (If JavaScript was the preferred choice).

Why Not Chosen?

Blazor is harder to learn for 2nd year students.

Node.js lacks inbuilt structure as that of ASP.NET Core.

**Entity Framework Core (Database Access)**

Why Chosen?

LINQ Support: Minimized SQL injection attacks through strongly-typed queries.

Migrations: Lightweight database schema migration.

Performance: Query performance optimized with lazy loading.

Entity Framework Core simplifies working with databases by allowing developers to write LINQ queries directly against their models, which saves time and reduces the chance of SQL errors (Srivastava, 2025).

Alternative Considered:

Dapper (Faster for read-heavy apps but needs manual SQL).

ADO.NET (Too low-level for this project).

Why Not Chosen?

Dapper has more boilerplate code.

ADO.NET is old for modern web applications.

**Visual Studio (IDE)**

Why Chosen?

Visual Studio is also used in our programming module so we will be familiar with the UI and coming from previous IDE like NetBeans is isn’t a major change of the interface.

Azure Integration: One-click deployment and debugging.

Productivity: IntelliSense, Git integration, and testing tools.

Alternative Considered:

Rider (JetBrains) (Cross-platform but not free).

Why Not Chosen?

Rider is fantastic but not within budget for this project.

System Key Features

* SQL View (BookingDetails)

Why Used?

Simplifies complex joins (Bookings + Events + Venues) into a queryable single object.

A SQL View acts as a virtual table that simplifies complex joins and can be used to expose only the necessary data to users or applications (Gupta, 2019).

Reduces redundant LINQ joins in controllers.

Alternative:

API-Level Joins (More flexible but more code).

* Double Booking Prevention

Implementation:

Validate in BookingController prior to saving.

Database level unique constraint (VenueId + Date) as backup

Alternative:

Database Trigger (More involved to update).

* Blob Storage for Images

Why Not Database Storage?

Other resources on azure have been created to work hand in hand with this project.

Performance: Databases are more optimized to process structured data.

Cost: SQL storage is expensive for binary files.

* Search Functionality (LINQ)

Why LINQ?

Strongly-typed queries reduce the risk of SQL injection.

Compile-time checks make it more trustworthy.

TempData for Error Handling

Why Used?

Convenient way of passing transient messages (e.g., "Cannot delete: Venue has bookings").

Alternative:

Toastr.js (For richer client-side notifications).

E. Reflective Technical Report

Self Reflective Technical

Part 1 was the hardest part of the project by far. I had to start from scratch and it took me the most time. I had to learn how to use Visual Studio and SQL Server Management Studio (SSMS), and then how to connect both of those to Azure all while trying to get the basics of the system working.

This project gave me real hands-on experience with ASP.NET Core and Azure services. Throughout, I faced many challenges:

* I was struggling to create connection strings correctly. It was annoying when Visual Studio would not retrieve the data from SSMS, or when the data was not appearing on the web page as I wanted it to.
* While implementing Part 2, some of my functionalities I had implemented in Part 1 got destroyed. I had to debug everything and make sure that the new feature did not interfere with what I had already done.
* I ran into trouble when I was unable to repopulate tables with the same name — I had to learn how to update existing tables but keep my data across all pieces of the POE.
* Installing Azure Blob Storage was in itself a challenge, mainly ensuring that uploaded images on the web page were actually being stored correctly in Azure.
* Validation writing that worked without interrupting the user experience was also something I had to test thoroughly.
* I also had to download many Nuget Packages to access certain functionality and make sure my code can execute without errors
* I encountered errors when dealing with SQL Views and foreign keys, which took me some time to know and fix fully.
* My Azure resources were deleted at the beginning of part 3, which caused my database and Visual Studio to get disconnected and not communicate with one another. I had to redo most of the things I did in Part 1, which was irritating.

Despite all these challenges, I kept going by debugging my code, referring to documentation, and testing time and again until everything went as it should.

Lessons Learned

One of the best things I learned was how applications like Visual Studio work with databases. In a production environment, a cloud technician isn't always in charge of the entire application development we work with the cloud and database . But since I'm completing an IT degree, this project let me understand how everything maps out and gave a better sense of the field I will work in and points out my strengths and weaknesses.

At a certain stage, the buttons and layout had appeared on the webpage but were not connected to the logic in the controllers, clicking the buttons did nothing. I had to sit down and go through everything to make sure my views communicated with the controller actions properly.

I also learned that cloud storage like Azure Blob is much more effective in storing files than keeping them stored in SQL or even on local machines like a hard drive or SSD

This project made me learn to better plan and understand the role of the MVC pattern — how the model, view, and controller all work together to develop a complete application. I learned the practice of using validation attributes to make a user-friendlier and more solid forms. I also understood how important error handling and user feedback are in real-world systems, especially when something doesn't work as users expect.

Understanding of cloud-based architecture

Before this project, I hadn't realized the benefits of cloud compared to local environments. Now I have , especially scalability, security, and deployment ease. Azure does a lot of the heavy lifting in the background, so as a ‘developer’ it made it easier.

One aspect that I liked was having the ability to open my project from anywhere ,either at home or on campus. Azure's web app made it simple for my lecturers to glance through my project using a link, without having to download and run anything. It is the very same link that would be seen by end-users in real life, and the developer would use Visual Studio to write and update the application.

I had to create an Azure server before creating a database because where would the database be stored, the blob storge uses that same server and everything is kept in one place making cloud an efficient way to store data than it being scattered on a local drive.

Having a Platform-as-a-Service (PaaS) strategy like Azure App Service relieved the pain of directly managing the infrastructure. Everything - deployment, monitoring, and the like was readily available in one place.

I also learned that cloud apps need to separate storage and processing ( storing images in Blob and data in SQL). This improves performance and makes scaling easier because Visual Studio doesn’t store photos

Cloud services are also cost-effective in the sense that you only pay for what you use. I now understand why our resources were deleted after they were not being used anymore after we got our marks and keeping them active would simply eat into credits. It made me understand that use of the cloud has to be meaningful and efficient.

Finally, this project educated me on the cloud design modularity each service is single-tasked and all of them combined provide an entire, scalable system.

References

**Bibliography**

“8 Microsoft Azure App Service Alternatives | DigitalOcean”. 2025. Digitalocean.com. 2025 <https://www.digitalocean.com/resources/articles/azure-app-service-alternatives>.

Gupta, Rajendra. 2019. “SQL View - a Complete Introduction and Walk-Through”. SQL Shack - Articles about Database Auditing, Server Performance, Data Recovery, and More. July 1, 2019 <https://www.sqlshack.com/sql-view-a-complete-introduction-and-walk-through/>.

QA. 2023. “Azure SQL: Databases Overview”. Qa.com. QA. January 31, 2023 <https://www.qa.com/resources/blog/azure-sql-databases-overview/>.

Srivastava, Sukriti. 2025. “Database Access with Entity Framework Core in .NET: A Developer’s Guide”. Metadesign Solutions. January 30, 2025 <https://metadesignsolutions.com/database-access-with-entity-framework-core-in-net-a-developers-guide/> [accessed 23 June 2025].

Storage, Pure. 2024. “Blob Storage vs. File Storage”. Pure Storage Blog. January 22, 2024 <https://blog.purestorage.com/purely-educational/blob-storage-vs-file-storage/>.

veli-pehlivanov. 2022. “ASP.NET Core vs ASP.NET MVC: Which .NET Framework Is Better for You?” Resolute Software. September 15, 2022 <https://www.resolutesoftware.com/blog/asp-net-mvc-vs-asp-net-core/>.