|  |  |
| --- | --- |
| CLDV6211 | |
|  | |
| POE | ST10458380BRADLEY RWAFA |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | REFLECTIVE REPORTnitially the priority was creating the EventType property which would utilize predefined values through seeding to show the various event types and forward its contents from OnModelCreating by using the method HasData(). Defining a limited property for the Venue Availability was next and this was done by limiting character size to 20 to promote compliance to data integrity and input. The Venue Availability property was defined as IsAvailable(as astring) in the Venue model and was not mapped and its value is determined as Available or Unavailable by the existence of an event on the venue according to a booking. In memory LINQ filters are applied to the application from the Booking controller’s index method after retrieval of neighbouring properties from Event to EventType and Venue. The search feature is able to utilize BookingIDs and VenueIDs and EventIDs to search through the bookings. The EventType filter is able to filter booking results through a collection of EventTypes determined in the code using a simple dropdown menu so is the Venue filter but using a collection on VenueIDs instead. The date filter uses defined variables called startdate and enddate to define a range within it can filter results these are compared to a property in the Event model called EventDate to filter results. The venue availability filter is able to filter simply using two statuses called Available or Unavailable. There was an implementation of the search functionality in the events controller specifically in the index utilizing a selection of the aforementioned filter types to develop a system of drop-down lists per rule by comparing it to the data retrieved from Event and Venue and EventType models. The index method in my venue controller determines availability per individual venue then the collection of venue data is further evaluated and skimmed down to an availability status as Available or Unavailable. There was the creation of a PopulateAvailability method in the venue controller which acted as a helper method simplifying the tracking of availability and data integrity across the multiple places where venue availability is needed to be determined especially across the views. The model features employed in mvc were crucial by utilising ModelState errors by comparing VenueID and EventDate upon the creation of bookings we are able to prevent double bookings and prompt the user errors. When deleting any events or venues using the delete confirmed methods in the Event and Venue controllers we utilise the booking async method to ensure that no conflicting bookings are currently active in the system upon deletion to prevent deletion of currently utilised items by linking their functionality. To create a bookingId that will be unique there is a composite key created utilising the VenueID and EventID part of the process is seen in ApplicationDBContext through the means of an EF Core Index. The use cases are provided for every property identified in the scenario allowing them to create, read, update and delete according to software development standards. The views for the Create/Edit all utilise SelectList to ensure that they retrieve relevant current values from the database which is better suited to the realtime system. When the application prompts for any inputs according for the date values a calendar form of input appears to ensure the user is able to visually provide better accurate information this can be when providing for EventDate, BookingDate while utilising [DataType(DataType.DateTime)] as well {0:yyyy-MM-ddTHH:mm}. In my Booking controller in my index and the create method from my Event controller there is the use of temporary seeding values for the EventType and Venue tables this is done for preparatory and testing purposes.The Azure resources were created using the Azure Student Starter subscription and all resources were created in the CLDV resource group. An Azure server called eventeasemypoeserver was created and it hosts another azure resource called EventEaseDb which is an SQL database on the Free/Shared tier, 32 GB pricing tier which allows for procedurally easy escalation of application features on a small scale application when deploying consistent code.A web application called EventEasePOE was created and it is utilising .NET 9.0. We are able to deploy the code from visual studio to the web application through the built-in publishing feature offered by visual studio connecting Microsoft accounts which allows are to select resource groups and resources we wish to deploy to this promotes updates and patches to code.Azure Blob Storage was used to store all instances of images uploaded to the web application in the creation of venues and the imageurl is stored within the container of the Blob Storage. These images were able to be reproduced as pictures on the application after being stored in the container. The modular nature of MVC allowed for easy incorporation and deployment in successive order to the application with a focus on the .NET 9.0 optimized written application features.Monitoring migrations during the development of the application was one of the biggest challenges faced specifically configuring the IsAvailabile field which resulted in various changes in migration files in the migration folder before publishing. This assisted in learning the role of the migrations in the application development phase and how tracking or removing them affects changes to the database.LINQ queries were failing to recognise EventType and Venue and various foreign keys that were utilised in my filters resulting in them not working properly. Modifying the code to do the retrieval process earlier for the crucial traversal items using Include(...).ThenInclude(...) and then transferring all of the filtering process into C# in-memory LINQ only after the .ToListAsync() to verify the existence of the loaded items.The availability of venue appeared differently on different pages such as the Booking and Venue each one would have its own. To combat the creation of the PopulateAvailability method was crucial to standardise the logic for the availability of status across the code whenever the data needed to be retrieved.The creation of the filters and features substantiated the use design tools and concepts such as ERDs and normalization allowing these incorporations to be made simpler. Outlining the domain diagram and according to these concepts as a precursor allowed the avoidance of unnecessary changes and remaining productive with a directive.The use of visual studio code in memory database and SQL options is a manner of development conductive of environmental errors. When making changes in a localised or temporary environment we are unable to understand the changes relative to the deployment environment it is best to replicate these changes (connection string etc) and adopt them as early as possible to avoid collusion and environmental errors in the production stage some of which were faced when creating this application.The integration of services used from Azure have given me a greater perspective on the (SQL, Blob Storage, etc) process of how they are able to connect to form bigger structures for automation processes in bigger companies which has made me more culpable to understand the development progression when working within a workflow that uses servers or is serverless in a future setting.Theory:The difference between CosmosDB and Relational Databases:Relational databases utilise normalized tables that follow strict ruling for their foreign keys and various data exchanges. Cosmos based on NoSQL is able to support various types of databases. Cosmos allows users to store key value pairs and JSON documents and more while offering the optionality to expand your data collection/file composition with minimal to no interruption while avoiding dramatically larger change procedures in comparison to relational databases. Cosmos DB was intentionally developed to be usable and accessible across third parties essentially decentralizing the need for a fixed data interpretation form as its able to clone and read data with instantaneous speeds across larger and different areas. Conventional databases or relational databases occasionally require the use of an intermediary to attempt to the reproduction process across these larger areas at such speeds.Decoupling in Cosmos DB is easier because of it is performed by at an automated pliable scale by means of throughput for reading and writing. In comparison traditional databases need individual directed monitoring to escalate computing   for their SQL databases which is conductive to many errors and possible interruptions.Factors to Consider for Logic Apps Handling Sensitive Data:Monitoring Connections and Authentications is critical by utilising the least privilege guideline for handling identities and utilising managed identities instead of combining them with connectors we can better handle sensitive data.Encryption of data is crucial to protecting data integrity understanding that cognitive applications will be bestowed encryption specifications and we must utilise user managed keys from Azure for the verification of our users in Azure services or for per connector we use.The recording of the data is very important to understand various applications retain important information which can be at times confidential or sensitive and in such settings, it is important that such applications are set to privately reroute their recorded data or delete the data in certain circumstances.Errors are able to produce critical contents of information on applications, so it is important taht they are managed in a manner that they rarely appear or do not show such information upon their occurrence. Attend to odd error occurrences always to determine causes early.Containing your network in a secure environment is crucial to data security when it comes to operating within an industry whereby the information is standardised and or sensitive then operate utilising protective environments such as Intergrated service environments .How Combining Event Grid with Other Services can Create Robust Workflows:The combination of Event Grid with Azure functions can best utilise event handlers that are able to only activate at necessitated times of operations essentially befitting the pay per use scheme which is a reduction on fees in comparison to large fee sums for usage or consistently active periods of activation.Event Grid with other functions allows for the better ability to skim through the vast amount of data travelling through the medium through the use of tags the combination is able to triangulate relevant data and accordingly forward it to its corresponding receiver in a near instantaneous automated manner. Screenshot:   Link for GIT and Website: Website : <https://eventeasepoe-bebgb8b8e7evhna6.canadacentral-01.azurewebsites.net/>  Github: https://github.com/RwafaBradley/CLDV6211POE | |  |
|  | REFERENCES | |  |
|  | **Nethikunta, N.** (2024) *SQL vs Cosmos database*. [Online]. Blog. Microsoft Q&A. Available at: <https://learn.microsoft.com/en-us/answers/questions/2084372/sql-vs-cosmos-database> (Accessed: 22 June 2025).    **freeCodeCamp.org** (2023) *Advanced C# – LINQ Tutorial*. [Online]. YouTube video, added by freeCodeCamp.org. Available at: <https://youtu.be/5l2qA3Pc83M> (Accessed: 22 June 2025).  **Microsoft** (2025) *Azure security baseline for Logic Apps*. [Online]. Microsoft Security Benchmark. Published 25 February 2025. Available at: <https://learn.microsoft.com/en-us/security/benchmark/azure/baselines/logic-apps-security-baseline> (Accessed: 21 June 2025)  **technetmagazine** (2023) *Master Event‑Driven Architectures with Azure Event Grid*. [Online]. Blog. TechNet Magazine. Available at: <https://www.technetmagazine.com/mastering-architectures-azure-event-grid/> (Accessed: 24 June 2025).  **Multishoring** (2024) *Building scalable event‑driven architectures with Azure Event Grid and Service Bus*. [Online]. Blog. Multishoring. Available at: <https://multishoring.com/blog/building-scalable-event-driven-architectures-with-azure-event-grid-and-service-bus/> (Accessed: 24 June 2025). |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  | Image | |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |