Airline Use Case & Test Document

EverythingBlockchain, Inc
https://builddb.io

Disclaimer

- This document and its associated content are prepared by EverythingBlockchain, Inc (EBI).
- This document and its associated content are protected trade secrets and MAY NOT BE SHARED with anyone outside of the following party(s) specified in this disclaimer without written EBI approval.
- This document and its associated content are prepared exclusively for the benefit of the following party(s): Turkish Airlines, Inc (THY) – A Star Alliance Member.
- This document and its associated content are provided under a 30-day proof of concept (POC) evaluation of BuildDB as a cloud database alternative using an airline-specific use case.
- Use of this POC grant EverythingBlockchain, Inc the right to use and publish performance results.

Introduction

If your cloud services require storage, then avoid the common technical debt that accompanies the sprawl of multiple databases. The BuildDB™ database stack has evolved into a Software as a Service (SaaS) solution where one database solution can outperform in cost and performance against the following competitors!

1. PostgreSQL
2. AWS DynamoDB
3. Azure CosmoDB
4. Google Cloud Datastore
5. Oracle NoSQL Database
6. MongoDB
7. RavenDB
8. Redis
9. Couchbase
10. MongoDB

.

Airline Industry Use Cases

Operating within the cloud is the prevailing trend across the airline industry. Unfortunately, another undeniable trend is the escalating costs of lifting and shifting into the cloud.

In today's modern airline, optimizing cloud costs while retaining customer loyalty are the keys to profitability! Achieving this balance depends upon identifying and evaluating and adopting technologies that provide a competitive advantage.

These competitive advantages take the form of:

1. **Trend Analysis Driving Customer Loyalty thru Cross Selling Opportunities**
    - Flight RecentSearches
    - Flight Hot Markets
    - Merchandising

2. **Regulatory Compliance thru Data Retention and Data Deduplication**
    - Flight
        - Shopping
        - Re-shopping
    - Passenger Name Record (PNR) Management
    - Electronic Miscellaneous Documents (EMD)

3. **Infinite Data Scalability Safeguards against Irregular Operation (IROP) Events**
    - Schedule Engine

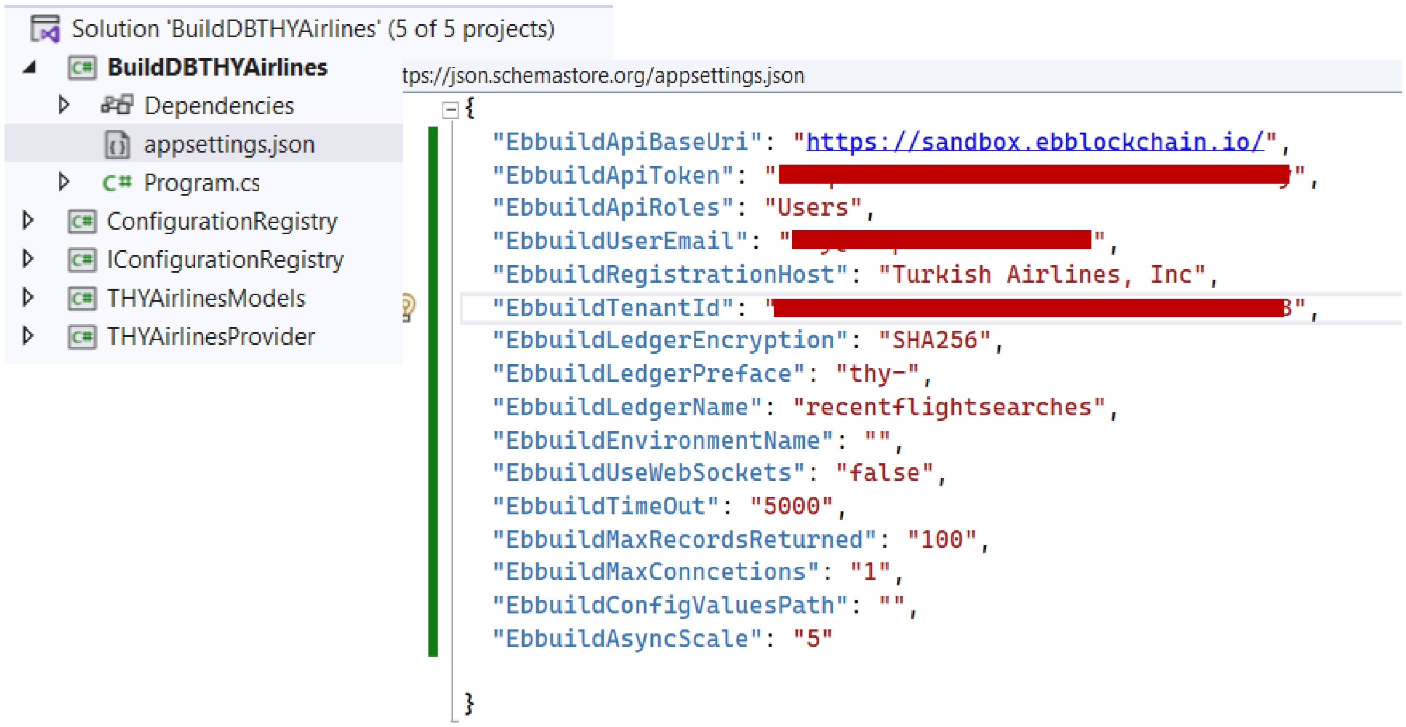
The Demo Project Structure

The associated demo is written in Microsoft C# under .NET Core 7.0 and designed via SOLID design principles. As such this project leverages common interfaces.
The Solution consists of five (5) projects which includes:

1.	BuildDBTHYAirlines.csproj	- The main entry point project where the testing begins.
2.	ConfigurationRegistry.csproj	- The supporting project used to simulate .NET ConfigurationRegistry.
3.	IConfigurationRegistry.csproj	- The interface contract for the ConfigurationRegistry project.
4.	THYAirlinesModels.csproj	- The C# classes used to represent the table schemas within the database.
5.	THYAirlinesProvider.csproj	- The airline domain-specific logic for flight recent searches and hot markets.

The Demo Project Credentials

The project was built custom for Turkish Airlines, Inc and contains all credentials within the appsettings.json file located under the BuildDBTHYAirlines.csproj.
Everything needed to run the demo is already contained within the projects and ready to run, test and modify as needed.



```
{
  "EbbuildApiBaseUri": "https://sandbox.ebblockchain.io/",
  "EbbuildApiToken": "[redacted]",
  "EbbuildApiRoles": "Users",
  "EbbuildUserEmail": "[redacted]",
  "EbbuildRegistrationHost": "Turkish Airlines, Inc",
  "EbbuildTenantId": "[redacted]",
  "EbbuildLedgerEncryption": "SHA256",
  "EbbuildLedgerPreface": "thy-",
  "EbbuildLedgerName": "recentflightsearches",
  "EbbuildEnvironmentName": "",
  "EbbuildUseWebSockets": "false",
  "EbbuildTimeOut": "5000",
  "EbbuildMaxRecordsReturned": "100",
  "EbbuildMaxConncetions": "1",
  "EbbuildConfigValuesPath": "",
  "EbbuildAsyncScale": "5"

}
```

The Demo - BuildDBTHYAirlines.csproj

This demo simulates the underlying activities conducted when passengers visit the homepage of the target airline website. The target airline uses a customer facing website to capture input for potential flight searches. The search activity is split between several underlying systems.
One of the underlying systems is called RecentSearches.

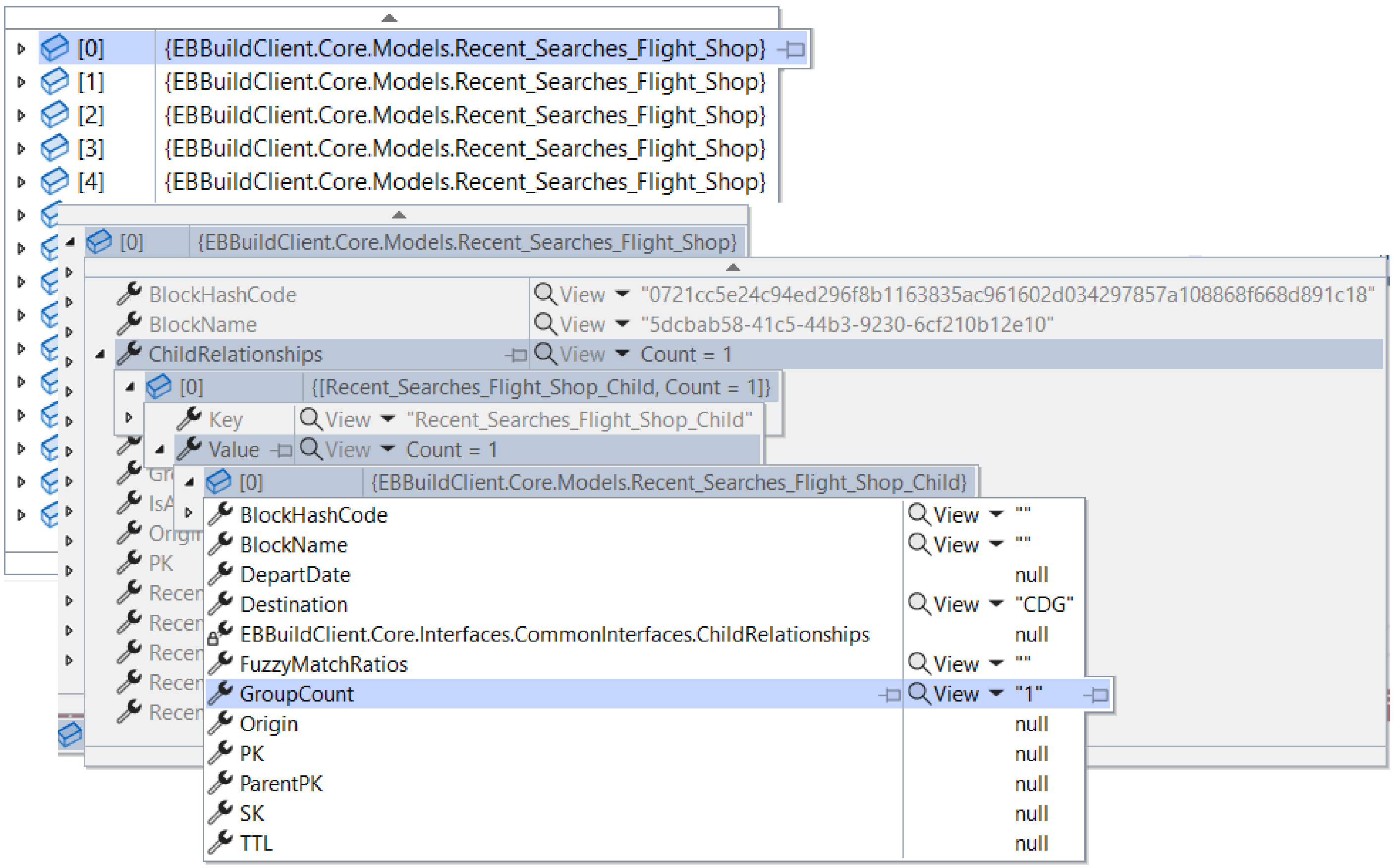
RecentSearch services is used to capture all flight search activities for both loyalty passengers and non-loyalty passengers. The loyalty ID of loyalty passengers is captured and used to conduct searches. The device ID of the physical device is captured and used to conduct searches for guest passengers.

The RecentSearch service is a critical aspect of customer retention as it provides the convenience of displaying all past flight searches for passengers to drive bookings.

Passengers engage in numerous flight searches before arriving at a booking. The ability to recall all origin and destination flight searches spanning several days is critical to passengers arriving at a booking decision in-order to create a revenue event.

Having a database that can scale in-order to accommodate the volume of flight searches is critical to the customer experience and retention. The next evolution in flight recent searches is the ability to mine data for what is termed as “Hot Markets”.

Hot Markets create cross selling opportunities for the airline to offset non-revenue generating award travel bookings. Using BuildDB the airline could mine recent searches for not only flight searches but also the most frequently searched travel destinations and then offer bundles through the merchandising services. The following screenshot shows the number of time a passenger searched for a destination of Charles de Gaulle Airport in France (CDG) from George Bush Airport (IAH) using BuildDB.



```
[0]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop}
[1]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop}
[2]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop}
[3]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop}
[4]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop}

[0]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop}
  BlockHashCode        View  "0721cc5e24c94ed296f8b1163835ac961602d034297857a108868f668d891c18"
  BlockName            View  "5dcbab58-41c5-44b3-9230-6cf210b12e10"
  ChildRelationships   View  Count = 1
    [0]  {[Recent_Searches_Flight_Shop_Child, Count = 1]}
      Key    View  "Recent_Searches_Flight_Shop_Child"
      Value  View  Count = 1
        [0]  {EBBuildClient.Core.Models.Recent_Searches_Flight_Shop_Child}
          BlockHashCode                                         View  ""
          BlockName                                             View  ""
          DepartDate                                                  null
          Destination                                           View  "CDG"
          EBBuildClient.Core.Interfaces.CommonInterfaces.ChildRelationships  null
          FuzzyMatchRatios                                      View  ""
          GroupCount                                            View  "1"
          Origin                                                      null
          PK                                                          null
          ParentPK                                                    null
          SK                                                          null
          TTL                                                         null
  Gr
  IsA
  Origi
  PK
  Recen
  Recen
  Recen
  Recen
  Recen
```

The Demo Project Testing Instructions

In-order to demonstrate this use case BuildDB will showcase several key features not available on other document databases. To test the following features simply open and run the Visual Studio Solution BuildDBTHYAirlines.csproj project file.

Once you have opened the BuildDBTHYAirlines.csproj project file you will locate the Program.cs file which lists functions called in-order to highlight the following features.

To see the actual BuildDB code open the THYAirlinesProvider.csproj project file and open the THYAirlinesProvider.cs file. This file contains the actual implementation used to call the BuildDB database consisting of CRUD functionality common to most developers.

1. **BuildDB demos high data throughput**
   a. BuildDB can ingest massive amounts of data at hi-speeds ensuring high throughput.

2. **BuildDB demos hierarchical relationships (joins) without the aid of costly table indexes.**
   a. All data stored with BuildDB is distributed and stored without indexes using a patent pending architecture.
   b. BuildDB will conduct hierarchical relationships between parent and child entities using complex joins and filtering without the use of any costly indexes.

3. **BuildDB demos advanced aggregate functions and data filtering with AI search and semantic matching**
   a. BuildDB can filter on fields semantically where the spelling may be inaccurate like invalid IATA airport codes that could potentially affect search results.
   b. BuildDB can filter on child records by a specific field like origin and conduct aggregate functions on a different field like destination to get group counts to determine the “Hot Markets”.

4. **BuildDB demos DDoS functions internally to combat throttling and potential denial of service attacks**
   a. BuildDB can conduct built in analysis against Denial of Service (DDoS) attacks by third parties scrapping valuable flight search results through legitimate credentials used maliciously.

5. **BuildDB demos enforcement of RBAC security**
   a. BuildDB can assign roles on tables and users/systems conducting searches to restrict flight search activity by a specific role.
   b. BuildDB can create flight search records securely using blockchain encryption and thereby protecting Personally Identifiable Information (PII) data like passenger loyalty IDs.