Final Milestone

for

Green Button Alliance

Version 1.0

Prepared by Ryan Duong, Daeun Lee, John Nguyen, Ali Ors, Connor Woods, Tin Le, Shrishak Dahal

04/29/22

Table of Contents

Title **1**

Table of Contents **2**

1. Introduction **3**
1.1 Purpose 3
1.2 Problem Statement 3
1.3 Intended Audience and Reading Suggestions 3
1.4 Product Scope 4
1.5 Project Success Metrics 4
1.6 Success of Project Metrics 4
1.7 References 4
2. Milestone 3 Overview **5**
2.1 Deliverables 5
2.2 UUID Pros and Cons 5
2.3 DataCustodian Database Recommendations 6
2.4 DataCustodian Database Model 7
2.5 Client Database Model 8
3. Milestone 3 Documentation **9**
3.1 Files Included 9
3.2 Dependencies 9
3.3 Background 9
3.4 Project Scope 10
3.5 Database Tables 10
3.6 Notes 11
3.7 Implementation Steps 11
3.8 Carrying the Project Forward 11
4. Conclusion **12**

Revision History

Name	Date	Reason For Changes	Version
Ryan Duong, Daeun Lee, John Nguyen, Ali Ors, Connor Woods, Tin Le, Shrishak Dahal	**05/03/22**	**Final Version**	**1.0**

1. Introduction

1.1 Purpose

GBA's current application does not run. By performing a database update and application prototype, UTDsolv students will provide a foundation for the development of an interface allowing third-party users to access ESPI-related information for their own use. The UTDsolv team will do this by providing database recommendations and an SQL database model.

1.2 Problem Statement

Green Button Alliance has developed an application that provides third-party users the ability to allow third parties to develop their own uses for energy data. However, this energy usage data is stored in a database using an atom structure - where relationships between tables are formed through the use of HREF links. While this allows the application to create a model with table relationships, many redundant tables and relationships are also formed, leading to inefficiencies in data querying performance. We intend to improve the database structure through the use of normalized primary and foreign key relationships over the current atom structure implementation. We will also provide a model for the Client database.

1.3 Intended Audience and Reading Suggestions

The target audience is directed toward customers and third parties who want to learn and have information about their energy usage. This document is aimed at project owners who would like to understand what is happening with their projects. The project team will be maintaining this document in order to convey the most accurate representation of the project's current scope and accomplishments.

1.4 Product Scope

This software development project will analyze the structure of the currently implemented database and offer recommendations for improving the database schema with structural improvements. This project will also cover the creation of a relational database that implements the ESPI standard as set by the NAESB. The project intends to deliver a database as a prototype which will be used for the implementation of a retail customer database with the existing DataCustodian database.

1.5 Project Success Metrics

1. **Increase in Performance of DataCustodian Database**
    a. Measured by a reduction in query time
    b. Measured by a reduction in memory usage
2. **Decrease Complexity of DataCustodian Database**
    a. Measured by a number of SQL tables
    b. Measured by joins necessary to query record
3. **Retain Data Structure of DataCustodian Database**
    a. Measured by a total number of data columns
    b. Measured by comparing old query to new query and counting dissimilarity

1.6 Success of Project Metrics

1. **Increase in Performance of DataCustodian Database**
    a. We have reduced the query *"SELECT * FROM USAGE_POINTS"* from 0.015 seconds to 0.014 seconds.
    b. We have reduced the DataCustodian script file from 673 lines to 590 lines.
2. **Decrease Complexity of DataCustodian Database**
    a. We have reduced the number of tables from 24 unique tables to 19 unique tables.
    b. We have eliminated the links tables thus reducing the necessary joins for a *Usage_Points* query by 1 join.
3. **Retain Data Structure of DataCustodian Database**
    a. We have retained the number of necessary data columns per table while reducing the overall amount of tables.
    b. The new queries show the same amount of information as the previous queries. The only difference is that we do not have to join with a *Links* table.

1.7 References

[1] Green Button Alliance. (2020, March). *Data Custodian MySQL Database*

[2] NAESB. (2020, January 30). *Energy Service Provider Interface*

2. Milestone 3 Overview

2.1 Deliverables

The UTDsolv students will be delivering:

1. **UUID Analysis**
   A brief report on the benefits and drawbacks of using UUID in a relational database.
2. **DataCustodian Database Recommendations**
   After our analysis, we have determined recommendations for the DataCustodian database.
3. **DataCustodian Database Model**
   Implementing our recommendations, we have provided a MySQL model for the DataCustodian database.
4. **Client Database Model**
   Incorporating necessary business requirements, we have created a MySQL model for the Client database.
5. **Documentation about Client Database**
   a. Handoff information regarding the development of the Client database.

2.2 UUID Pros and Cons

A UUID, Universally Unique Identifier, is used to label information in computer systems information. UUID has many advantages when used exclusively. When you use a UUID, you can be certain it will be unique across every table, database, server, etc you have. Since it is unique across all these platforms, it is insurmountable to generate a duplicate UUID. That means whatever information you are labeling will be extremely difficult to access thus ensuring security. Since it is unique across everything, it is straightforward to merge data across multiple data tables and databases. This also allows for painless distribution of databases across multiple servers if needed. Finally, UUID can be generated not only inside the database but outside the database if needed. This allows for more fluidity for the database UUID generation and gives more options when trying to decide if it is needed.

While there are numerous advantages to using a UUID, there are also disadvantages. Firstly, it is long. More accurately, it is around 4 times longer than your usually normal 4-byte value. This makes calling the UUID a pain because you will need to input the identifier every time you write a query where you need the user id. This length can also cause debugging

issues when your queries aren't working. Imagine needing to debug something with a long id made of random letters, numbers, and symbols. Secondly, again due to its length, it will result in a significant drop in index performances. Thirdly, it is database-dependent when you are generating using a database function. This means it will be difficult to generate because you can't use a database function and need to use an outside source. This can cause some clog in your query functions as well as not having the advantage of using normal database functions to generate. Lastly, the clog also passes onto your URLs if you are going to need that. This means that the URLs you are trying to generate will also be clunky and not straightforward. Even though you can mitigate this by using slugs, it is still an inconvenience to use UUID.

In conclusion, while a UUID has many advantages to its use, you still have to consider the disadvantages when deciding to use it. The main problem is an inconvenience not only for the user but for the developer. The recommendation is that UUID should be considered the last option once you have outweighed all the other options. Even though it has many advantages, sometimes the disadvantage of inconvenience is sizable enough to go down a different route.

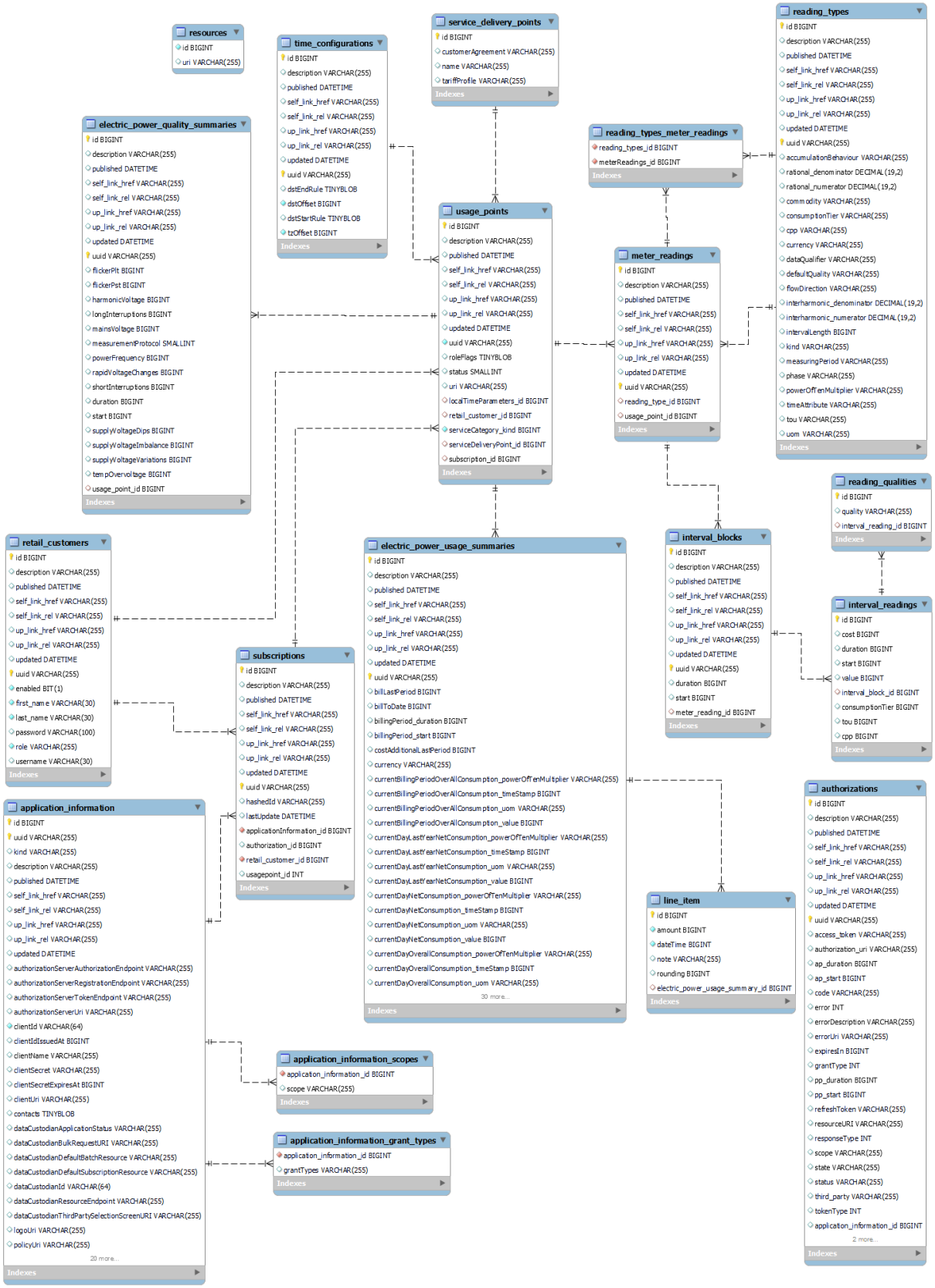
2.3 DataCustodian Database Recommendations

- Reducing the links to finding the relevant information
- Primary and secondary key rework
- "Universal ID" recommendation
- Optimal Indexes and Queries Rework

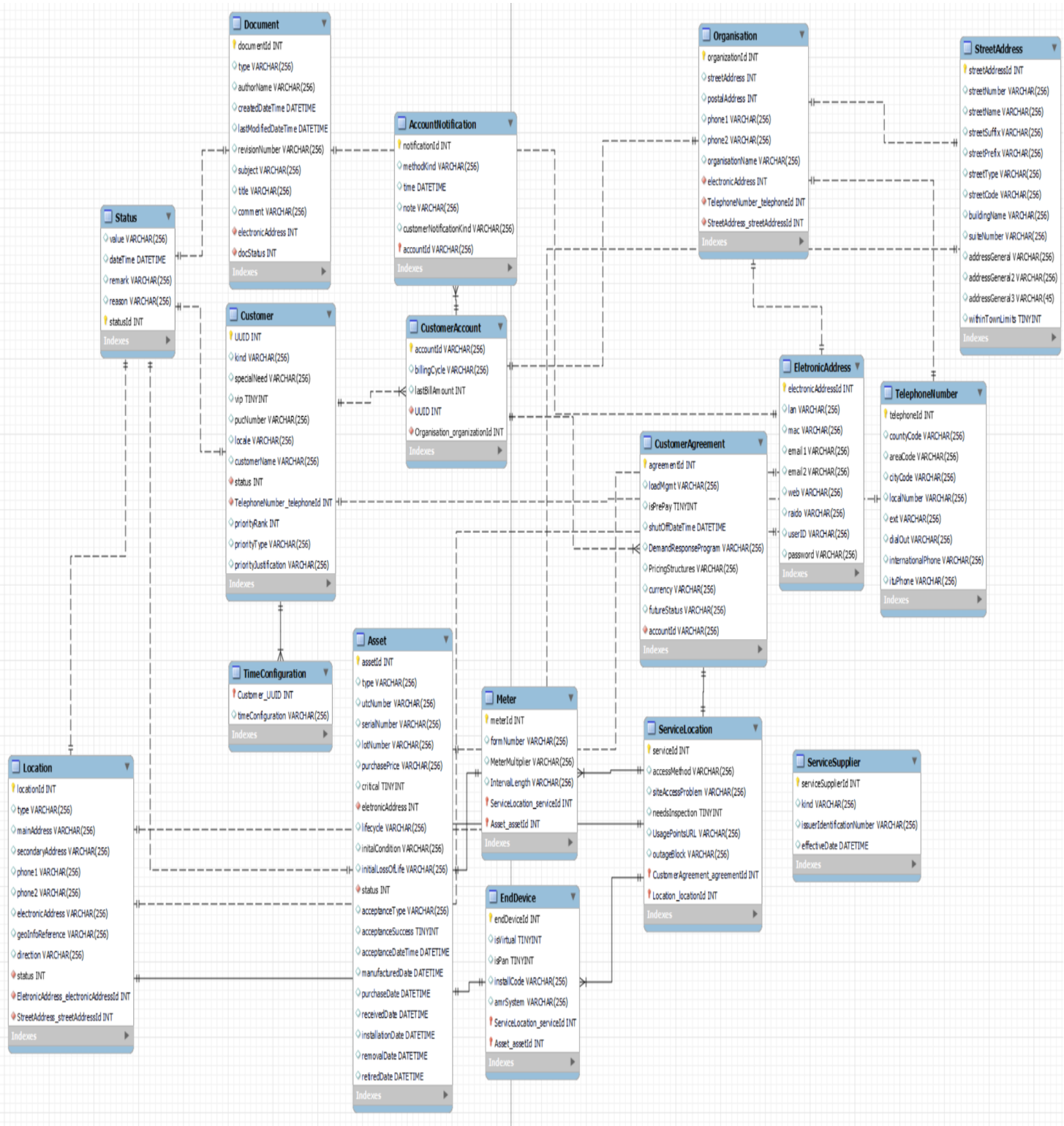
Although the DataCustodian database model was not developed by UTDsolv students and was sourced from GBA's MySQL server, we have included a database model for the DataCustodian database with our recommendations already implemented. For the DataCustodian database model we are handing over, we have removed the link tables while retaining the link columns for future implementation purposes. We believe that on the server-side the database should be purely relational, while the atom model can be represented dynamically on the application side. We've also made the UUID the primary key in most tables while retaining the ID primary key for

indexing purposes. This data model is accurate to the NAESB ESPI data schema and will provide a solid foundation for the development of an application.

2.4 DataCustodian Database Model:



2.5 Client Database Model



3. Milestone 3 Documentation

Database Engineer: John Nguyen (jnn180001@utdallas.edu)

3.1 Project History

- **January 2022 - March 2022**
    - Created plan of attack for the development of GBA's Energy Data Application.
- **March 2022 - April 2022**
    - Reviewed and Analyzed DataCustodian Database.
- **April 2022 - May 2022**
    - Gathered business requirements and created a Client Database model.

3.1 Files Included

- DataCustodian Database Model File
- DataCustodian Database MySQL Script
- Client Database Model File
- Client Database MySQL Script

3.2 Dependencies

- NAESB Energy Service Provider Interface, Version 3.3
- Developed on MySQL Workbench 8.0 CE

3.3 Background

Green Button Alliance has reached out to UTDsolv to perform a database update and optimization based on the North American Energy Standards Board's Energy Service Provider Interface data schema. This database will be used to develop an application that allows third-party companies to access an energy service provider customer's data if granted. The UTDsolv students have provided recommendations for updating the DataCustodian database and a MySQL model with the recommendations implemented. The UTDsolv students have also developed a database model which implements NAESB's retail customer data schema into a MySQL relational database.

3.4 Project Scope

Both databases are based on the NAESB ESPI data schema. The Client database model is focused on the ESPI retail customer data schema and does not include any other data schemas found in the ESPI. The UTDsolv students have not developed an application that implements the database.

3.5 Database Tables

DataCustodian Database:

- Resources
- Time Configurations
- Service Delivery Points
- Meter Readings
- Reading Types Meter Readings
- Reading Types
- Electric Power Quality Summaries
- Electric Power Usage Summaries
- Line Item
- Usage Points
- Interval Blocks
- Interval Readings
- Reading Qualities
- Retail Customer
- Subscriptions
- Applicant Information
- Application Information Scopes
- Application Information Grant Types
- Authorizations

Client Database:

- Customer
- CustomerAccount
- AccountNotification
- Document
- Status
- TimeConfiguration
- Organization
- ServiceSupplier
- Location
- ServiceLocation
- StreetAddress
- TelephoneNumber
- ElectronicAddress
- Meter
- EndDevice
- Asset

3.6 Notes

Regarding the Client database, we've condensed the way some tables are laid out from the ESPI data schemat. For *StreetAddress* and *TelephoneNumber*, we've preserved all the necessary columns from the retail customer schema, but we've removed extraneous tables by combining them all into a single *StreetAddress* and *TelephoneNumber* table. The combination of tables has allowed us to reduce the complexity of the data schema, and improve ease of use.

Although the *Asset* table can be combined into the Meter and *EndDevice* table, we have decided it would be more readable to keep it as a standalone table. We have not implemented atom link columns in this database. Should atom links become necessary, we believe these will be best implemented on the application side, rather than stored in the database, in order to conserve data space.

The Client database should be updated in conformance to NAESB ESPI data standards as necessary.

3.7 Implementation Steps

1. Download database MySQL script we've provided
2. Load the forward-engineered MySQL script into MySQL database
3. Verify successful implementation of MySQL database

3.8 Carrying the Project Forward

1. Take our database models and initialize them on a MySQL server
2. Create an application utilizing REST technology to interface with the database
3. Load real-world energy usage data into the database and share it with third-party companies for implementation

4. Conclusion

Green Button Alliance (“GBA”) has developed an application that provides third-party users the ability to interface with the information provided by the NAESB Energy Services Provider Interface model, which is populated with data from energy service providers. GBA intends for this application to allow third parties to develop their own uses for the energy data. GBA’s application is outdated and does not function as intended. Over the course of 5 months, the UTDsolv team has analyzed current systems and formulated a plan of attack for revitalizing GBA’s application. The first step in this endeavor is developing an up-to-date MySQL database based on the NAESB ESPI standards. By performing a database evaluation, UTDsolv students have provided a foundation for the development of an application that allows third-party users to access ESPI-related information for their own use. Additionally, the team has generated a layout for a potential retail customer (client) database for GBA’s independent development and eventual merger with their existing application.