For a lean and efficient database that supports scalability and fast queries, a NoSQL database (Firestore/Firebase) will be used. This structure will ensure correct variable placement while maintaining efficiency. The structure optimizes minimal data storage, faster queries, and clear relationships for KPIs.

The Entity Relationship Diagram (ERD) is given in Figure 1. Whilst ERDs are generally linked to relational databases (and not NOSQL) databases, it does provide an idea of information flow. The link to the save version of the ERD is shown in the heading of Figure 1. Figure 1 shows that 55 variables across 10 Collections are needed to ensure that all data necessary to calculate user and organization key performance indicators are collected. The key performance indicators themselves are not stored in the Collections; they are however, calculated through queries may on the Collections. Section 3 shows the Queries that are necessary to calculate both organizational and user KPIs.

|  |
| --- |
|  |
| Figure 1. Collections diagram showing relationship between Collections of PACE Database (Link: https://dbdiagram.io/d/67ba15e4263d6cf9a019100c) |

**Section 02: Firestore Database Collections**

1. **Users Collection**

|  |  |  |
| --- | --- | --- |
| Table 1. **Users Collection:** Stores Core User Info such as credentials and essential information. | | |
| **Column Name** | **Data Type** | **Notes** |
| user\_id | UUID | Unique identifier (PRIMARY KEY) |
| Name | VARCHAR | User’s full name (MAX CHARACTERS: 100) |
| password\_hash | TEXT | Encrypted password |
| date\_started | DATE | When user joined |
| Age | INT | User’s age |
| organization | VARCHAR | Associated org (MAX CHARACTERS: 100) |
| last\_activity\_start | TIMESTAMP | Start of last session |
| last\_activity\_end | TIMESTAMP | End of last session |
| Date\_Created | TIMESTAMP | Date joined service |
| Feedback\_Rating | INT | NPScore |
| Churn\_Date | TIMESTAMP | Date left service |
| Last\_Login | TIMESTAMP | Last login date |
| **NB**: Last activity timestamps (for engagement tracking) | | |

|  |  |
| --- | --- |
| Table 2. Users Collection: Associated Code to create Collection | |
| 1 | { |
| 2 | "user\_id": "string", |
| 3 | "name": "string", |
| 4 | "password": "string", |
| 5 | "date\_started": "timestamp", |
| 6 | "age": "number", |
| 7 | "organization": "string", |
| 8 | "last\_activity\_start": "timestamp", |
| 9 | "last\_activity\_end": "timestamp", |
| 10 | "date\_created": "timestamp", |
| 11 | "last\_login": "timestamp", |
| 12 | "churn\_date": "timestamp", |
| 13 | "feedback\_rating": "number" // (For NPS, scale 1-10) |
| 14 | } |
| password\_hash instead of plain password is used to ensure Firebase Authentication.  Timestamp format: ISO 8601 format to keep consistency. | |

1. **Sustainable Records Collection**

|  |  |  |
| --- | --- | --- |
| Table 3. Sustainable Records Collection: Stores User-Performed SDG Actions | | |
| **Column Name** | **Data Type** | **Notes** |
| record\_id | UUID | Unique identifier (PRIMARY KEY) |
| user\_id | UUID | Links to Users (FOREIGN KEY) |
| sustainable\_actions\_id | UUID | Links to Actions (FOREIGN KEY) |
| lesson\_id | UUID | Links to Sustainability Lessons (FOREIGN KEY) |
| week\_of\_year | INT | Week number (1-52) |
| Weekly\_experience\_points | INT | XP assigned by user |
| **NB:** Tracks what users did related to sustainability  Enables Reduce, Reuse, Recycle KPIs  Supports Public Participation Rate calculations | | |

|  |  |
| --- | --- |
| Table 4. Sustainable Records Collection: Associated Code to create Collection | |
| 1 | { |
| 2 | "record\_id": "string", |
| 3 | "sustainable\_actions\_id": "string", |
| 4 | "sustainability\_lessons\_id": "string", |
| 5 | "week\_of\_year": "number", |
| 6 | "user\_id": "string", |
| 7 | "weekly\_experience\_points": "number" |
| 8 | “lesson\_id”: “string” |
| 9 | } |

1. **SDGActions Collection**

|  |  |  |
| --- | --- | --- |
| Table 5. SDGActions Collection: Defines and Stores Information about Sustainable Actions | | |
| **Column Name** | **Data Type** | **Notes** |
| action\_id | UUID | Unique identifier (Primary Key) |
| duration | INT | Duration in minutes |
| cost | DECIMAL(10,2) | Estimated cost |
| persons\_affected | INT | Number impacted |
| location | VARCHAR(255) | Impacted location |
| description | TEXT | Details |
| creator\_id | UUID | References Users (FOREIGN KEY) |
| recycle\_amount | INT | Amount of recycle material in kg |
| persons\_recycling | INT | Number of persons recycling |
| total\_persons\_on\_service | INT | Total number of persons on service |
| unit\_recycling\_cost | INT | Unit cost of recycling |
| contaminated\_waste | INT | For Waste Contamination Rate |
| **NB:** Duration, cost, and persons impacted by sustainability actions  Enables cost savings from recycling & reuse KPI  Allows tracking impact per user | | |

|  |  |
| --- | --- |
| Table 6. SDGActions Collection: Associated Code to create Collection | |
| 1 | { |
| 2 | "action\_id": "string", |
| 3 | "duration": "number", |
| 4 | "cost": "number", |
| 5 | "persons\_affected": "number", |
| 6 | "location": "string", |
| 7 | "description": "string", |
| 8 | "creator": "string", |
| 9 | "recycle\_amount": "number", |
| 10 | "persons\_recycling": "number", |
| 11 | "total\_persons\_on\_service": "number", |
| 12 | "unit\_recycling\_cost": "number", |
| 13 | "contaminated\_waste": "number" // (For Waste Contamination Rate) |
| 14 | } |

1. **Rewards Collection**

|  |  |  |
| --- | --- | --- |
| Table 7 Rewards Collection: Stores User Points for Actions | | |
| **Column Name** | **Data Type** | **Notes** |
| reward\_id | UUID | Unique identifier (PRIMARY KEY) |
| user\_id | UUID | Unique Identifier (FOREIGN KEY) |
| points\_for\_joining | INT | Points for join service |
| points\_for\_promotion | INT | Points for promoting service |
| points\_for\_coalitions | INT | Points for joining or creating coalitions |
| points\_for\_levels | INT | Points for moving up level |
| **NB:** Points for various user activities  Enables loyalty & engagement tracking | | |

|  |  |
| --- | --- |
| Table 8. Rewards Collection: Associated Code for creating Collection | |
| 1 | { |
| 2 | “reward\_id”: “string” //PRIMARY KEY |
| 3 | “user\_id”: “string” |
| 4 | “points\_for\_joining”: number //DEFAULT 0, |
| 5 | “points\_for\_promotion”: number //DEFAULT 0, |
| 6 | “points\_for\_coalitions”: number //DEFAULT 0, |
| 7 | “points\_for\_levels”: number //DEFAULT 0 |
| 8 | } |

1. **Waste Data collection**

|  |  |  |
| --- | --- | --- |
| Table 9 Waste Data: Stores User Points for Actions | | |
| **Column Name** | **Data Type** | **Notes** |
| waste\_id | UUID | Unique identifier (PRIMARY KEY) |
| user\_id | UUID | Unique Identifier (FOREIGN KEY) |
| waste\_amount | INT | Amount wasted from recycling |
| waste\_recycled | INT | Amount of waste recycled |
| Recycle\_cost | INT | Total cost of recycling |

|  |  |
| --- | --- |
| Table 10. Waste Data: Associated Code for creating Collection | |
| 1 | { |
| 2 | "waste\_id": "string", |
| 3 | "user\_id": "string", |
| 4 | "waste\_amount": "number", |
| 5 | "waste\_recycled": "number", |
| 6 | "recycling\_cost": "number" |
| 7 | } |

1. **Support Request Collection (For IT KPIs)**

|  |  |  |
| --- | --- | --- |
| Table 11 Support Request Collection: For IT KPIs | | |
| **Column Name** | **Data Type** | **Notes** |
| request\_id | UUID | Unique identifier (PRIMARY KEY) |
| user\_id | VARCHAR | User who filed request |
| timestamp | TIMESTAMP | Time for the event |
| status | STRING | Status of request |

|  |  |
| --- | --- |
| Table 12. Support Request Collection: Associated Code for creating Collection | |
| 1 | { |
| 2 | "request\_id": "string", |
| 3 | "user\_id": "string", |
| 4 | "timestamp": "timestamp", |
| 5 | "status": "string" // (open, closed, pending) |
| 6 | } |

1. **Visitor Logs Collection**

|  |  |  |
| --- | --- | --- |
| Table 13 Visitor Logs Collection | | |
| **Column Name** | **Data Type** | **Notes** |
| session\_id | UUID | Unique identifier (PRIMARY KEY) |
| visitor\_id | UUID | Person visiting site or app |
| conversion\_status | BOOLEAN | ('uptime', 'failure')) |

|  |  |
| --- | --- |
| Table 14. Visitor Logs Collection: Associated Codee for creating Collection | |
| 1 | { |
| 2 | “session\_id”: “string” //PRIMARY KEY |
| 3 | “visitor\_id”: “string” |
| 4 | “conversion\_status”: boolean |
| 5 | } |

1. **System Logs Collection**

|  |  |  |
| --- | --- | --- |
| Table 15 System Logs Collection: For IT KPIs | | |
| **Column Name** | **Data Type** | **Notes** |
| log\_id | UUID | Unique identifier (PRIMARY KEY) |
| event\_type | VARCHAR | ('uptime', 'failure')) |
| event\_timestamp | TIMESTAMP | Time for the event |
| system\_id | UUID | System where event occurred |
| **NB:** Logs system uptime/failures  Used for Uptime % and Mean Time Between Failures | | |

|  |  |
| --- | --- |
| Table 16. System Logs Collection: Associated Codee for creating Collection | |
| 1 | { |
| 2 | “log\_id”: “log\_4445” //PRIMARY KEY |
| 3 | “event\_type”: “uptime” //event\_type IN ('uptime', 'failure') |
| 4 | “event\_timestamp”: 1124512 |
| 5 | “system\_id”: “system\_2223” |
| 6 | } |

1. **Transactions Collection**

|  |  |  |
| --- | --- | --- |
| Table 17 Transactions Collection | | |
| **Column Name** | **Data Type** | **Notes** |
| Transactions\_id | UUID | Unique identifier (PRIMARY KEY) |
| User\_id | UUID | ('uptime', 'failure')) |
| Amount\_spent | INT | Time for the event |
| timestamp | TIMESTAMP | System where event occurred |
| **NB:** Logs system uptime/failures  Used for Uptime % and Mean Time Between Failures | | |

|  |  |
| --- | --- |
| Table 18. Transactions Collection: Associated Code for creating Collection | |
| 1 | { |
| 2 | “Transactions\_id”: “STRING” |
| 3 | “User\_id”: “STRING” |
| 4 | “Amount\_spent”: “INT” |
| 5 | “timestamp”: “TIMESTAMP” |
| 6 | } |

1. **Customer Feedback Collection**

|  |  |  |
| --- | --- | --- |
| Table 19 Customer Feedback Collection: For Net Promoter Score - NPS | | |
| **Column Name** | **Data Type** | **Notes** |
| feedback\_id | UUID | Unique identifier (PRIMARY KEY) |
| user\_id | UUID | Unique Identifier (FOREIGN KEY) |
| Rating | INT | Net promoter Score |
| feedback\_text | TEXT | Feedback on Service |
| submitted\_on | TIMESTAMP | Date feedback was submitted |
| **NB:** NPS Score to calculate customer satisfaction  User feedback stored for insights | | |

|  |  |
| --- | --- |
| Table 20. Customer Feedback Collection: Associated Code for Creating Collection | |
| 1 | { |
| 2 | “feedback\_id”: 00235 //PRIMARY KEY |
| 3 | “user\_id”: “user\_11456” //Users(user\_id), |
| 4 | “rating”: 2 //Check (nps\_score BETWEEN 0 AND 10), |
| 5 | “feedback\_text”: “Excellent service” |
| 6 | “submitted\_on”: 0002354 // DEFAULT CURRENT\_TIMESTAMP |
| 7 | } |

**Section 03: Firestore Database Security rules**

Firestore Security Rules to protect data while allowing authorized access:

* Users can only access their own data
* Admins can manage all records
* Data integrity is preserved (no overwriting of system-generated fields)
* Uers can only modify their own data
* Only admins can modify SDG actions, rewards, and KPIs
* Prevents users from exposing or modifying passwordHash
* SDG actions and KPIs are public for transparency

|  |  |
| --- | --- |
| Table 21. Firestore Security Rules | |
| 1 | rules\_version = '2'; |
| 2 | service cloud.firestore { |
| 3 | match /databases/{database}/documents { |
| 4 |  |
| 5 | // Users Collection - Users can only read/update their own data |
| 6 | match /users/{userId} { |
| 7 | allow read, update: if request.auth != null && request.auth.uid == userId; |
| 8 | allow create: if request.auth != null; |
| 9 | allow delete: if request.auth.token.admin == true; // Only admins can delete |
| 10 | } |
| 11 |  |
| 12 | // Sustainable Records - Users can only access records linked to them |
| 13 | match /sustainable\_records/{recordId} { |
| 14 | allow read, update, delete: if request.auth != null && request.auth.uid == resource.data.userId; |
| 15 | allow create: if request.auth != null; |
| 16 | } |
| 17 |  |
| 18 | // Sustainable Development Goals Actions - Read access for everyone, write access for admins |
| 19 | match /sdg\_actions/{actionId} { |
| 20 | allow read: if true; // Publicly accessible |
| 21 | allow create, update, delete: if request.auth.token.admin == true; |
| 22 | } |
| 23 |  |
| 24 | // Rewards - Read access for all users, but only admins can modify |
| 25 | match /rewards/{rewardId} { |
| 26 | allow read: if true; |
| 27 | allow create, update, delete: if request.auth.token.admin == true; |
| 28 | } |
| 29 |  |
| 30 | // KPIs - Read access for everyone, but only admins can update |
| 31 | match /kpis/{kpiId} { |
| 32 | allow read: if true; |
| 33 | allow create, update, delete: if request.auth.token.admin == true; |
| 34 | } |
| 35 |  |
| 36 | // Ensure passwordHash field is never exposed |
| 37 | match /users/{userId} { |
| 38 | allow read: if request.auth != null && request.auth.uid == userId; |
| 39 | allow update: if request.auth != null && request.auth.uid == userId |
| 40 | && !("passwordHash" in request.resource.data); // Prevent modifying passwordHash directly |
| 41 | } |
| 42 | } |
| 43 | } |

**Section 3: Queries to obtain Key Performance Indicators**

Table 22 to Table 50 provides the necessary information to calculate the Key Performance Indicators from the organization perspective. The Key performance indicators can be obtained from simple queries (a request for data) done on these Tables. The next section provides all the queries necessary to obtain the key performance indicators.

**Proposed Viability KPI Queries**

|  |  |
| --- | --- |
| Table 22. Customer Acquisition Rate (CAR) Query | |
| 1 | async function getCustomerAcquisitionRate(startDate, endDate) { |
| 2 | const q = query(collection(db, "Users"), where("date\_started", ">=", startDate), where("date\_started", "<=", endDate)); |
| 3 | const snapshot = await getDocs(q); |
| 4 | return snapshot.size; // Number of users acquired in the given period |
| 5 | } |

|  |  |
| --- | --- |
| Table 23. Net Promoter Score (NPS) Query | |
| 1 | async function getNetPromoterScore() { |
| 2 | const q = query(collection(db, "Customer Feedback")); |
| 3 | const snapshot = await getDocs(q); |
| 4 | let promoters = 0, detractors = 0, total = 0; |
| 5 | snapshot.forEach(doc => { |
| 6 | const rating = doc.data().Rating; |
| 7 | if (rating >= 9) promoters++; |
| 8 | else if (rating <= 6) detractors++; |
| 9 | total++; |
| 10 | }); |
| 11 | return total > 0 ? ((promoters - detractors) / total) \* 100 : 0; |
| 12 | } |

|  |  |
| --- | --- |
| Table 24. Customer Lifetime Value (CLV) Query | |
| 1 | async function getCustomerLifetimeValue() { |
| 2 | const q = query(collection(db, "Transactions")); |
| 3 | const snapshot = await getDocs(q); |
| 4 | let totalRevenue = 0, totalUsers = new Set(); |
| 5 | snapshot.forEach(doc => { |
| 6 | totalRevenue += doc.data().Amount\_spent; |
| 7 | totalUsers.add(doc.data().User\_id); |
| 8 | }); |
| 9 | return totalUsers.size > 0 ? totalRevenue / totalUsers.size : 0; |
| 10 | } |

|  |  |
| --- | --- |
| Table 25. Churn Rate Query | |
| 1 | async function getChurnRate() { |
| 2 | const q = query(collection(db, "Users"), where("Churn\_Date", "!=", null)); |
| 3 | const snapshot = await getDocs(q); |
| 4 | const churnedUsers = snapshot.size; |
| 5 | const totalUsersSnapshot = await getDocs(collection(db, "Users")); |
| 6 | const totalUsers = totalUsersSnapshot.size; |
| 7 | return totalUsers > 0 ? (churnedUsers / totalUsers) \* 100 : 0; |
| 8 | } |

**Proposed Information Technology KPI Queries**

|  |  |
| --- | --- |
| Table 26. Total Support Requests Query | |
| 1 | async function getTotalSupportRequests() { |
| 2 | const snapshot = await getDocs(collection(db, "Support Request")); |
| 3 | return snapshot.size; |
| 4 | } |

**Proposed Customer Service KPI Queries**

|  |  |
| --- | --- |
| Table 27. Customer Retention Rate Query | |
| 1 | async function getCustomerRetentionRate(startDate, endDate) { |
| 2 | const totalUsersSnapshot = await getDocs(collection(db, "Users")); |
| 3 | const totalUsers = totalUsersSnapshot.size; |
| 4 | const retainedUsersQuery = query(collection(db, "Users"), where("last\_activity\_end", ">=", startDate), where("last\_activity\_end", "<=", endDate)); |
| 5 | const retainedUsersSnapshot = await getDocs(retainedUsersQuery); |
| 6 | return totalUsers > 0 ? (retainedUsersSnapshot.size / totalUsers) \* 100 : 0; |
| 7 | } |

**Growth KPI Queries**

|  |  |
| --- | --- |
| Table 28. Helper function to get date ranges | |
| 1 | function getDateRange(daysAgo) { |
| 2 | const today = new Date(); |
| 3 | today.setHours(0, 0, 0, 0); |
| 4 | const pastDate = new Date(today); |
| 5 | pastDate.setDate(today.getDate() - daysAgo); |
| 6 | return { today, pastDate }; |
| 7 | } |

|  |  |
| --- | --- |
| Table 29. New Users Query (Users created in the last 7 days) | |
| 1 | async function getNewUsers() { |
| 2 | const { pastDate } = getDateRange(7); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Date\_Created", ">=", pastDate) |
| 5 | .get(); |
| 6 | return snapshot.size; |
| 7 | } |

|  |  |
| --- | --- |
| Table 30. Current Users Query (Users who have logged in recently) | |
| 1 | async function getCurrentUsers() { |
| 2 | const { pastDate } = getDateRange(30); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Last\_Login", ">=", pastDate) |
| 5 | .get(); |
| 6 | return snapshot.size; |
| 7 | } |

|  |  |
| --- | --- |
| Table 31. Reactivated Users Query (Users who were dormant but recently became active) | |
| 1 | async function getReactivatedUsers() { |
| 2 | const { pastDate } = getDateRange(30); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Last\_activity\_start", ">=", pastDate) |
| 5 | .where("Last\_activity\_end", "<=", pastDate) |
| 6 | .get(); |
| 7 | return snapshot.size; |
| 8 | } |

|  |  |
| --- | --- |
| Table 32. Resurrected Users Query (Users who churned but logged in again) | |
| 1 | async function getResurrectedUsers() { |
| 2 | const { pastDate } = getDateRange(30); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Churn\_Date", "!=", null) |
| 5 | .where("Last\_Login", ">=", pastDate) |
| 6 | .get(); |
| 7 | return snapshot.size; |
| 8 | } |

|  |  |
| --- | --- |
| Table 33. At-risk Weekly Active Users Query (Users active last week but not this week) | |
| 1 | async function getAtRiskWeeklyUsers() { |
| 2 | const { pastDate: lastWeekStart } = getDateRange(14); |
| 3 | const { pastDate: thisWeekStart } = getDateRange(7); |
| 4 | const lastWeekUsers = await db.collection("Users") |
| 5 | .where("Last\_activity\_start", ">=", lastWeekStart) |
| 6 | .where("Last\_activity\_start", "<", thisWeekStart) |
| 7 | .get(); |
| 8 | return lastWeekUsers.size; |
| 9 | } |

|  |  |
| --- | --- |
| Table 34. At-risk Monthly Active Users Query (Users active last month but not this month) | |
| 1 | async function getAtRiskMonthlyUsers() { |
| 2 | const { pastDate: lastMonthStart } = getDateRange(60); |
| 3 | const { pastDate: thisMonthStart } = getDateRange(30); |
| 4 | const lastMonthUsers = await db.collection("Users") |
| 5 | .where("Last\_activity\_start", ">=", lastMonthStart) |
| 6 | .where("Last\_activity\_start", "<", thisMonthStart) |
| 7 | .get(); |
| 8 | return lastMonthUsers.size; |
| 9 | } |

|  |  |
| --- | --- |
| Table 35. Dormant Users Query (Users inactive for more than 60 days) | |
| 1 | async function getDormantUsers() { |
| 2 | const { pastDate } = getDateRange(60); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Last\_activity\_start", "<", pastDate) |
| 5 | .get(); |
| 6 | return snapshot.size; |
| 7 | } |

|  |  |
| --- | --- |
| Table 36. Daily Active Users Query (Users with activity today) | |
| 1 | async function getDailyActiveUsers() { |
| 2 | const { today } = getDateRange(0); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Last\_activity\_start", ">=", today) |
| 5 | .get(); |
| 6 | return snapshot.size; |
| 7 | } |

|  |  |
| --- | --- |
| Table 37. Weekly Active Users Query (Users with activity in the last 7 days) | |
| 1 | async function getWeeklyActiveUsers() { |
| 2 | const { pastDate } = getDateRange(7); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Last\_activity\_start", ">=", pastDate) |
| 5 | .get(); |
| 6 | return snapshot.size; |
| 7 | } |

|  |  |
| --- | --- |
| Table 38. Monthly Active Users Query (Users with activity in the last 30 days) | |
| 1 | async function getMonthlyActiveUsers() { |
| 2 | const { pastDate } = getDateRange(30); |
| 3 | const snapshot = await db.collection("Users") |
| 4 | .where("Last\_activity\_start", ">=", pastDate) |
| 5 | .get(); |
| 6 | return snapshot.size; |
| 7 | } |

|  |  |
| --- | --- |
| Table 39. Run all queries and log results Query | |
| 1 | async function fetchAllKPIs() { |
| 2 | console.log("New Users:", await getNewUsers()); |
| 3 | console.log("Current Users:", await getCurrentUsers()); |
| 4 | console.log("Reactivated Users:", await getReactivatedUsers()); |
| 5 | console.log("Resurrected Users:", await getResurrectedUsers()); |
| 6 | console.log("At-risk Weekly Active Users:", await getAtRiskWeeklyUsers()); |
| 7 | console.log("At-risk Monthly Active Users:", await getAtRiskMonthlyUsers()); |
| 8 | console.log("Dormant Users:", await getDormantUsers()); |
| 9 | console.log("Daily Active Users:", await getDailyActiveUsers()); |
| 10 | console.log("Weekly Active Users:", await getWeeklyActiveUsers()); |
| 11 | console.log("Monthly Active Users:", await getMonthlyActiveUsers()); |
| 12 | } |
| 13 |  |
| 14 | fetchAllKPIs(); |

**Proposed Reduce Recycle and Reuse KPI Queries**

|  |  |
| --- | --- |
| Table 40. Recycle Rate Query | |
| 1 | async function getRecycleRate() { |
| 2 | const snapshot = await getDocs(collection(db, "Waste Data")); |
| 3 | let totalWaste = 0, totalRecycled = 0; |
| 4 | snapshot.forEach(doc => { |
| 5 | totalWaste += doc.data().waste\_amount; |
| 6 | totalRecycled += doc.data().waste\_recycled; |
| 7 | }); |
| 8 | return totalWaste > 0 ? (totalRecycled / totalWaste) \* 100 : 0; |
| 9 | } |

**Proposed Viability KPI Queries**

|  |  |
| --- | --- |
| Table 41. Customer Acquisition Rate Query | |
| 1 | def get\_customer\_acquisition\_rate(): |
| 2 | users\_ref = db.collection("Users Collection") |
| 3 | total\_users = len(users\_ref.stream()) |
| 4 | start\_date = datetime.now() - timedelta(days=30) |
| 5 | new\_users = len([user for user in users\_ref.where("date\_started", ">=", start\_date).stream()]) |
| 6 | return new\_users / total\_users if total\_users > 0 else 0 |

|  |  |
| --- | --- |
| Table 42. Net Promotor Score Query | |
| 1 | def get\_net\_promoter\_score(): |
| 2 | feedback\_ref = db.collection("Customer Feedback Collection") |
| 3 | feedbacks = [doc.to\_dict() for doc in feedback\_ref.stream()] |
| 4 | promoters = len([f for f in feedbacks if f['Rating'] >= 9]) |
| 5 | detractors = len([f for f in feedbacks if f['Rating'] <= 6]) |
| 6 | total\_responses = len(feedbacks) |
| 7 | return ((promoters - detractors) / total\_responses) \* 100 if total\_responses > 0 else 0 |

|  |  |
| --- | --- |
| Table 43. Customer Lifetime Value Query | |
| 1 | def get\_customer\_lifetime\_value(): |
| 2 | transactions\_ref = db.collection("Transactions Collection") |
| 3 | users\_ref = db.collection("Users Collection") |
| 4 | total\_revenue = sum([t.to\_dict()['Amount\_spent'] for t in transactions\_ref.stream()]) |
| 5 | total\_users = len(users\_ref.stream()) |
| 6 | return total\_revenue / total\_users if total\_users > 0 else 0 |

|  |  |
| --- | --- |
| Table 44. Breakeven point Query | |
| 1 | def get\_break\_even\_point(): |
| 2 | total\_revenue = sum([t.to\_dict()['Amount\_spent'] for t in db.collection("Transactions Collection").stream()]) |
| 3 | total\_cost = sum([w.to\_dict()['Recycle\_cost'] for w in db.collection("Waste Data collection").stream()]) |
| 4 | return total\_cost / total\_revenue if total\_revenue > 0 else 0 |

|  |  |
| --- | --- |
| Table 45. Conversion Rate Query | |
| 1 | def get\_conversion\_rate(): |
| 2 | visitor\_logs\_ref = db.collection("Visitor Logs Collection") |
| 3 | total\_visitors = len(visitor\_logs\_ref.stream()) |
| 4 | converted\_visitors = len([v for v in visitor\_logs\_ref.where("conversion\_status", "==", True).stream()]) |
| 5 | return converted\_visitors / total\_visitors if total\_visitors > 0 else 0 |

|  |  |
| --- | --- |
| Table 46. Churn Rate Query | |
| 1 | def get\_churn\_rate(): |
| 2 | users\_ref = db.collection("Users Collection") |
| 3 | total\_users = len(users\_ref.stream()) |
| 4 | churned\_users = len([u for u in users\_ref.stream() if u.to\_dict().get("Churn\_Date")]) |
| 5 | return churned\_users / total\_users if total\_users > 0 else 0 |

**Proposed Information Technology KPI Queries**

|  |  |
| --- | --- |
| Table 47. Total Support Request Query | |
| 1 | def get\_total\_support\_requests(): |
| 2 | return len(db.collection("Support Request Collection").stream()) |

|  |  |
| --- | --- |
| Table 48. Average Time Between Failures Query | |
| 1 | def get\_average\_time\_between\_failures(): |
| 2 | system\_logs = [log.to\_dict() for log in db.collection("System Logs Collection").stream()] |
| 3 | failure\_events = sorted([log for log in system\_logs if log['event\_type'] == 'failure'], key=lambda x: x['event\_timestamp']) |
| 4 | if len(failure\_events) < 2: |
| 5 | return None |
| 6 | total\_time = sum([(failure\_events[i]['event\_timestamp'] - failure\_events[i - 1]['event\_timestamp']).total\_seconds() for i in range(1, len(failure\_events))]) |
| 7 | return total\_time / (len(failure\_events) - 1) |

|  |  |
| --- | --- |
| Table 49. Uptime Percentage Query | |
| 1 | def get\_uptime\_percentage(): |
| 2 | total\_events = len(db.collection("System Logs Collection").stream()) |
| 3 | failure\_events = len([log for log in db.collection("System Logs Collection").where("event\_type", "==", "failure").stream()]) |
| 4 | return ((total\_events - failure\_events) / total\_events) \* 100 if total\_events > 0 else 100 |

**Proposed Customer Service KPI Queries**

|  |  |
| --- | --- |
| Table 50. Average Time Between Failures Query | |
| 1 | def get\_customer\_retention\_rate(): |
| 2 | users\_ref = db.collection("Users Collection") |
| 3 | total\_users = len(users\_ref.stream()) |
| 4 | retained\_users = len([u for u in users\_ref.stream() if not u.to\_dict().get("Churn\_Date")]) |
| 5 | return retained\_users / total\_users if total\_users > 0 else 0 |