Functional Requirements Analysis - Utility Billing System

1. Authentication & Authorization Functional Requirements

Functional Requirements

- **FR-AUTH-01:** The system shall allow users to log in using valid credentials.
- **FR-AUTH-02:** The system shall implement JWT-based authentication.
- **FR-AUTH-03:** The system shall generate a JWT token upon successful login.
- **FR-AUTH-04:** The system shall implement role-based access control (RBAC) with roles: Admin, Billing Officer, Account Officer, Consumer.
- **FR-AUTH-05:** The system shall secure billing and payment APIs with authentication.

Business Rules

- Password must be stored in hashed format (BCrypt implementation).
- JWT token must include user role in claims.
- Unauthorized users must not access protected APIs.
- All billing and payment APIs require authentication except login and register endpoints.
- BaseController provides `CurrentUserId` and `CurrentUserEmail` properties for authenticated requests.

2. User Management (Admin) Functional Requirements

Functional Requirements

- **FR-ADMIN-01:** Admin shall be able to create users.
- **FR-ADMIN-02:** Admin shall be able to assign roles (Admin / Billing Officer / Account Officer / Consumer).
- **FR-ADMIN-03:** Admin shall be able to view all users.
- **FR-ADMIN-04:** Admin shall be able to manage utility types.
- **FR-ADMIN-05:** Admin shall be able to manage tariffs.
- **FR-ADMIN-06:** Admin shall be able to manage billing cycles.
- **FR-ADMIN-07:** Admin shall be able to view audit logs of system actions.
- **FR-ADMIN-08:** Admin shall be able to manage connections (create, update, delete).
- **FR-ADMIN-09:** Admin shall be able to approve or reject utility connection requests.

Business Rules

- Only Admin can manage users, utility types, tariffs, billing cycles, and connections.
- Each user must have exactly one role.
- Admin can approve utility connection requests which automatically creates connections.
- All admin actions are logged in the audit log system.

3. Consumer & Connection Management Functional Requirements

Functional Requirements

- **FR-CONSUMER-01:** Consumer shall be able to register with profile information.
- **FR-CONSUMER-02:** Consumer shall be able to view own profile information.
- **FR-CONSUMER-03:** Consumer shall be able to update own profile information.
- **FR-CONSUMER-04:** System shall store utility connection details including:
  - Utility type
  - Meter number
  - Tariff plan
- **FR-CONSUMER-05:** Consumer shall be able to request new utility connections.
- **FR-CONSUMER-06:** Consumer shall be able to view own utility connections.
- **FR-CONSUMER-07:** Consumer shall be able to view own bills and invoices.
- **FR-CONSUMER-08:** Consumer shall be able to pay bills through the system.
- **FR-CONSUMER-09:** Consumer shall be able to view consumption details and trends.

Business Rules

- Consumers can only view and modify their own profile and data.
- Each connection must have a utility type, meter number, and tariff plan.
- Consumers must request new connections through the utility request system.
- Consumers can only view and pay their own bills.
- Payment can only be recorded for unpaid bills.

---

4. Meter Reading Management Functional Requirements

Functional Requirements

- **FR-METER-01:** Billing Officer shall be able to enter monthly meter readings for connections.
- **FR-METER-02:** System shall validate previous reading vs current reading (current reading must be >= previous reading).
- **FR-METER-03:** System shall automatically calculate consumption units (CurrentReading - PreviousReading).
- **FR-METER-04:** Billing Officer shall be able to view meter reading history with filtering.
- **FR-METER-05:** Billing Officer shall be able to view connections needing meter readings.

Business Rules

- Current reading must be >= previous reading, otherwise validation error is raised.
- Consumption calculation: Consumption = CurrentReading - PreviousReading.
- For first reading, previous reading is considered as 0.
- Meter readings are linked to tariff plans for accurate bill calculation.
- Reading status tracks: Pending → Billed (after bill generation).

---

5. Billing & Invoice Generation Functional Requirements

Functional Requirements

- **FR-BILL-01:** System shall automatically generate bills from meter readings based on consumption and tariff.

- **FR-BILL-02:** System shall calculate bill amounts including base charges based on consumption and tariff.
- **FR-BILL-03:** System shall calculate taxes on bill amounts.
- **FR-BILL-04:** System shall apply fixed charges to bills.
- **FR-BILL-05:** System shall apply penalties to bills.
- **FR-BILL-06:** System shall track bill status lifecycle: Generated → Due → Paid → Overdue.
- **FR-BILL-07:** Billing Officer shall be able to generate bills individually or in batch.
- **FR-BILL-08:** Billing Officer shall be able to view pending bills (readings without bills).
- **FR-BILL-09:** System shall automatically update bill status based on due dates.

Business Rules

- Bills are automatically generated from meter readings.
- Bill calculation includes consumption charges, fixed charges, taxes, and penalties.
- Bill status follows lifecycle: Generated → Due → Paid → Overdue.
- Bill status is automatically updated based on current date vs due date (handled in memory for real-time status).
- Batch bill generation allows processing multiple meter readings at once.
- Bills are linked to meter readings and connections.
- Bill generation triggers notifications to consumers.

---

6. Payment Tracking Functional Requirements

Functional Requirements

- **FR-PAYMENT-01:** Consumer shall be able to record payments for bills.
- **FR-PAYMENT-02:** System shall support payment methods: Cash and Online (UPI) - mock implementation.
- **FR-PAYMENT-03:** System shall calculate outstanding balance for consumers.
- **FR-PAYMENT-04:** Consumer shall be able to view payment history.
- **FR-PAYMENT-05:** System shall track payment status (Pending, Completed, Failed).
- **FR-PAYMENT-06:** System shall generate receipt numbers for payments.

Business Rules

- Payment can only be recorded for unpaid bills.
- Payment methods: Cash or Online (UPI) - mock implementation.
- For Online payments, UPI ID is required (stored as upiId in payment record).
- For Cash payments, receipt number is generated automatically.
- Payment status: Pending → Completed (or Failed).
- Outstanding balance includes all unpaid bills (Due, Overdue, Generated status).
- **Role-Based Access:**
  - **Consumer:** Can only view own payment history and own outstanding balance.
  - **Account Officer:** Can view all payments and outstanding balances system-wide.

---

7. Account Officer Functional Requirements

Payment Tracking (System-Wide)

- **FR-ACCOUNTOFFICER-01:** Account Officer shall be able to view all payments in the system (payment audit trail).
- **FR-ACCOUNTOFFICER-02:** Account Officer shall be able to search and filter payments by:
  - Consumer name or ID
  - Date range (payment date)
  - Utility type
  - Payment method (Cash/Online)
  - Payment amount range
- **FR-ACCOUNTOFFICER-03:** Account Officer shall be able to view payment details including:
  - Consumer information
  - Bill information
  - Payment amount
  - Payment method
  - Payment reference number
  - Payment date and time
- **FR-ACCOUNTOFFICER-04:** Account Officer shall be able to view paginated list of all payments with sorting capabilities.

Outstanding Balance Tracking (System-Wide)

- **FR-ACCOUNTOFFICER-05:** Account Officer shall be able to view all outstanding balances across all consumers.
- **FR-ACCOUNTOFFICER-06:** Account Officer shall be able to view outstanding bills with filtering by:
  - Bill status (Due, Overdue, Generated)
  - Consumer name or ID
  - Utility type
  - Due date range
  - Outstanding amount range
- **FR-ACCOUNTOFFICER-07:** Account Officer shall be able to view outstanding balance summary grouped by:
  - Consumer
  - Utility type
  - Bill status
- **FR-ACCOUNTOFFICER-08:** Account Officer shall be able to view total outstanding amount across the entire system.

Consumer Billing Summary (System-Wide)

- **FR-ACCOUNTOFFICER-09:** Account Officer shall be able to view consumer-wise billing summary for all consumers, including:
  - Total billed amount
  - Total paid amount
  - Outstanding balance
  - Number of overdue bills
  - Payment history summary
- **FR-ACCOUNTOFFICER-10:** Account Officer shall be able to search and filter consumer billing summaries by:

- Consumer name or ID
- Outstanding balance range
- Number of overdue bills

Dashboard & Analytics

- **FR-ACCOUNTOFFICER-11:** Account Officer shall be able to view dashboard with:
  - Total revenue (sum of all paid bills)
  - Total outstanding balance (sum of all unpaid bills)
  - Unpaid bills count
  - Recent payments (system-wide)
  - Outstanding balances by utility type
- **FR-ACCOUNTOFFICER-12:** Account Officer shall be able to view monthly revenue trends (system-wide).

Business Rules

- Account Officer has **system-wide access** to all payment and balance data (not restricted to own data).
- Account Officer can view all payments regardless of consumer.
- Account Officer can view all outstanding balances across all consumers.
- Account Officer cannot modify payments or bills (read-only access for tracking purposes).
- All Account Officer queries must use LINQ for data retrieval and aggregations.
- Account Officer dashboard must use LINQ GroupBy and Sum aggregations for revenue and outstanding balance calculations.
- Payment audit trail must include all payment records with complete details for accountability.
- Outstanding balance calculations must include penalties for overdue bills.
- Dashboard metrics include:
  - Total Revenue (sum of all completed payments)
  - Outstanding Dues (sum of all unpaid bills)
  - Unpaid Bills Count
  - Total Consumption (system-wide)
- Monthly revenue is calculated from payment dates, grouped by year and month.
- Outstanding bills are filtered by status (Due, Overdue, Generated) and paginated.
- Consumer billing summary aggregates: Total Billed, Total Paid, Outstanding Balance, Overdue Count per consumer.

---

8. Reports & Dashboards Functional Requirements

Functional Requirements

- **FR-REPORT-01:** System shall generate monthly revenue report.
- **FR-REPORT-02:** System shall generate outstanding dues report.
- **FR-REPORT-03:** System shall generate consumption report by utility type.
- **FR-REPORT-04:** System shall generate consumer-wise billing summary.
- **FR-REPORT-05:** System shall generate average consumption report by utility type.
- **FR-REPORT-06:** System shall generate connections count by utility type.

- **FR-REPORT-07:** System shall generate report summary for admin dashboard.

Business Rules

- Reports must use LINQ for calculations and aggregations.
- Revenue reports must group data by month (using payment dates).
- Consumption reports must group data by utility type.
- Consumer billing summary must aggregate data by consumer.
- Average consumption uses LINQ `.Average()` aggregation.
- **Role-Based Access:**
  - **Admin:** Can access all reports with system-wide data (summary, connections by utility).
  - **Account Officer:** Can access all payment and consumption reports with system-wide data.
  - **Consumer:** Can access reports for own data only (consumption, dashboard).
  - **Billing Officer:** Can access billing-related reports for bill generation purposes.

---

9. Notifications Functional Requirements

Functional Requirements

- **FR-NOTIFY-01:** System shall notify consumer when bill is generated.
- **FR-NOTIFY-02:** System shall send due date reminders to consumers.
- **FR-NOTIFY-03:** Consumer shall be able to view all notifications.
- **FR-NOTIFY-04:** Consumer shall be able to mark notifications as read.
- **FR-NOTIFY-05:** Consumer shall be able to view unread notification count.

Business Rules

- Bill generation automatically triggers notification via notification queue.
- Due date reminders are sent automatically:
  - 7 days before due date
  - 3 days before due date
  - On due date (urgent reminder)
- Reminders are only sent for bills with status "Generated" or "Due" (not paid or overdue).
- Duplicate reminders are prevented (one reminder per day per bill per reminder type).
- Notifications are stored in database with read/unread status.
- Background service runs continuously and checks for due dates every hour.
- Notification messages include bill amount, utility type, billing period, and due date.

---

10. Utility Request Management Functional Requirements

Functional Requirements

- **FR-REQUEST-01:** Consumer shall be able to request new utility connections.
- **FR-REQUEST-02:** Consumer shall be able to view own utility connection requests.
- **FR-REQUEST-03:** Admin shall be able to view all utility connection requests.

- **FR-REQUEST-04:** Admin shall be able to approve utility connection requests.
- **FR-REQUEST-05:** Admin shall be able to reject utility connection requests.
- **FR-REQUEST-06:** System shall automatically create connection when request is approved.

Business Rules

- Consumers can create requests for new utility connections.
- Request status: Pending → Approved/Rejected.
- When approved, a connection is automatically created with the provided details.
- Consumers can only view their own requests.
- Admin and Billing Officers can view all requests.
- Only Admin can approve or reject requests.
- Request approval requires connection details (utility type, meter number, tariff plan, address).

---

11. Audit Logging Functional Requirements

Functional Requirements

- **FR-AUDIT-01:** System shall log all administrative actions.
- **FR-AUDIT-02:** Admin shall be able to view audit logs.
- **FR-AUDIT-03:** Audit logs shall include action type, details, timestamp, and performer.

Business Rules

- All admin actions are logged automatically:
  - User management (create, update, delete)
  - Utility type management
  - Tariff management
  - Billing cycle management
  - Connection management
  - Request approval/rejection
  - Bill generation
- Audit logs include:
  - Timestamp (UTC)
  - Action type (e.g., "BILL_GENERATE", "USER_CREATE")
  - Details (descriptive message)
  - Performed by (user email)
- Only Admin can view audit logs.
- Logs are ordered by timestamp (most recent first).

---

12. API Endpoints Summary

Authentication

- `POST /api/auth/register` - User registration (AllowAnonymous)

- `POST /api/auth/login` - User authentication (AllowAnonymous)

User Management (Admin)

- `GET /api/user` - Get all users
- `GET /api/user/{id}` - Get user by ID
- `POST /api/user` - Create user
- `PUT /api/user/{id}` - Update user
- `DELETE /api/user/{id}` - Delete user (soft delete)

Utility Type Management (Admin)

- `GET /api/utilitytype` - Get all utility types (All authenticated)
- `GET /api/utilitytype/{id}` - Get utility type by ID (Admin, Billing Officer)
- `POST /api/utilitytype` - Create utility type (Admin)
- `PUT /api/utilitytype/{id}` - Update utility type (Admin)
- `DELETE /api/utilitytype/{id}` - Delete utility type (Admin)

Tariff Management (Admin)

- `GET /api/tariff` - Get all tariffs (Admin, Billing Officer)
- `GET /api/tariff/{id}` - Get tariff by ID (Admin, Billing Officer)
- `POST /api/tariff` - Create tariff (Admin)
- `PUT /api/tariff/{id}` - Update tariff (Admin)
- `DELETE /api/tariff/{id}` - Delete tariff (Admin)

Billing Cycle Management (Admin)

- `GET /api/billingcycle` - Get all billing cycles (Admin, Billing Officer)
- `GET /api/billingcycle/{id}` - Get billing cycle by ID (Admin, Billing Officer)
- `POST /api/billingcycle` - Create billing cycle (Admin)
- `PUT /api/billingcycle/{id}` - Update billing cycle (Admin)
- `DELETE /api/billingcycle/{id}` - Delete billing cycle (Admin)

Connection Management

- `GET /api/connection` - Get all connections (Admin)
- `GET /api/connection/my` - Get consumer's connections (Consumer, Admin)
- `GET /api/connection/{id}` - Get connection by ID
- `POST /api/connection` - Create connection (Admin)
- `PUT /api/connection/{id}` - Update connection (Admin)
- `DELETE /api/connection/{id}` - Delete connection (Admin)

Utility Request Management

- `GET /api/utilityrequest` - Get all requests (Admin, Billing Officer)
- `GET /api/utilityrequest/my` - Get consumer's requests (Consumer)
- `GET /api/utilityrequest/{id}` - Get request by ID
- `POST /api/utilityrequest` - Create request (Consumer, Admin)

- `POST /api/utilityrequest/{id}/approve` - Approve request (Admin)
- `POST /api/utilityrequest/{id}/reject` - Reject request (Admin)

Meter Reading Management (Billing Officer)

- `GET /api/meterreading/connections` - Get connections needing readings
- `GET /api/meterreading/previous/{connectionId}` - Get previous reading
- `POST /api/meterreading` - Create meter reading
- `GET /api/meterreading` - Get reading history (with filters)
- `GET /api/meterreading/{id}` - Get reading by ID
- `PUT /api/meterreading/{id}` - Update reading

Bill Management

- `GET /api/bill/pending` - Get pending bills (Billing Officer)
- `POST /api/bill/generate` - Generate single bill (Billing Officer)
- `POST /api/bill/generate/batch` - Generate bills in batch (Billing Officer)
- `GET /api/bill/connection/{connectionId}` - Get bills by connection (Billing Officer)
- `GET /api/bill/{id}` - Get bill by ID (Billing Officer, Account Officer)
- `GET /api/bill/my` - Get consumer's bills (Consumer)
- `GET /api/bill/my/{id}` - Get consumer's specific bill (Consumer)
- `POST /api/bill/{id}/pay` - Record payment (Consumer)
- `GET /api/bill/outstanding` - Get outstanding bills (Account Officer)
- `GET /api/bill/consumers/summary` - Get consumer billing summary (Account Officer)

Payment Management

- `GET /api/payment/my` - Get consumer's payment history (Consumer)
- `GET /api/payment/audit` - Get payment audit trail (Account Officer)

Reports

- `GET /api/report/summary` - Get report summary (Admin)
- `GET /api/report/consumption` - Get consumption by utility (Account Officer)
- `GET /api/report/average-consumption` - Get average consumption (Account Officer)
- `GET /api/report/connections-by-utility` - Get connections by utility (Admin)
- `GET /api/report/consumption/my` - Get consumer's consumption (Consumer)
- `GET /api/report/dashboard` - Get consumer dashboard (Consumer)
- `GET /api/report/dashboard/summary` - Get account officer dashboard (Account Officer)
- `GET /api/report/monthly-revenue-by-billing` - Get monthly revenue (Account Officer)
- `GET /api/report/dashboard/recent-payments` - Get recent payments (Account Officer)
- `GET /api/report/dashboard/outstanding-by-utility` - Get outstanding by utility (Account Officer)

Notifications (Consumer)

- `GET /api/notification` - Get notifications
- `GET /api/notification/unread-count` - Get unread count

- `PUT /api/notification/{id}/read` - Mark as read
- `PUT /api/notification/mark-all-read` - Mark all as read
- `DELETE /api/notification/all` - Delete all notifications

Audit Logs (Admin)

- `GET /api/auditlog` - Get all audit logs

---

13. Security Requirements (Non-Functional)

Requirements

- **NFR-SEC-01:** APIs must be secured using JWT authentication.
- **NFR-SEC-02:** Role-based authorization must be enforced on all endpoints.
- **NFR-SEC-03:** Secure handling of consumer and billing data must be ensured.
- **NFR-SEC-04:** Unauthorized access must be denied with appropriate error responses.

BaseController: Provides `CurrentUserId` and `CurrentUserEmail` properties from JWT claims

Global Exception Middleware: `GlobalExceptionMiddleware` handles all exceptions consistently

Password Security: BCrypt hashing for password storage

Business Rules

- All protected endpoints must validate JWT token.
- Sensitive data must not be exposed in API responses.
- Role-based access control enforced at controller and action levels.
- Consumers can only access their own data (enforced in service layer).
- Error responses follow consistent format via global exception handler.

---

14. Backend Architecture Requirements (Non-Functional)

Requirements

- **NFR-ARCH-01:** System must use DTOs to avoid exposing entities directly.
- **NFR-ARCH-02:** Global exception handling middleware must be implemented for consistent error responses.
- **NFR-ARCH-03:** System must use LINQ for billing calculations and reporting.
- **NFR-ARCH-04:** Configuration must be stored in appsettings.json.
- **NFR-ARCH-05:** Codebase must be maintainable and scalable with clean architecture.
- **NFR-ARCH-06:** Centralized exception handling must be implemented.

Implementation Details

Project Structure:

- `Controllers/` - API endpoints (13 controllers)
- `Services/Interfaces/` - Service interfaces
- `Services/Implementations/` - Service implementations (14 services)
- `Models/Core/` - Entity models

- `Models/Dto/` - Data Transfer Objects (46 DTOs organized by feature)
- `Data/` - DbContext and data seeding
- `Middleware/` - Global exception middleware
- `Services/BackgroundServices/` - Background services (notification service)
- `Services/Helpers/` - Helper classes (DashboardMetricsHelper)

Services Implemented:

- `AuthService` - Authentication and JWT
- `UserService` - User management
- `UtilityTypeService` - Utility type management
- `TariffService` - Tariff management
- `BillingCycleService` - Billing cycle management
- `ConnectionService` - Connection management
- `UtilityRequestService` - Request management
- `MeterReadingService` - Meter reading operations
- `BillService` - Bill generation and management
- `BillCalculationService` - Bill amount calculations
- `PaymentService` - Payment processing
- `AccountOfficerService` - Account Officer operations
- `ReportService` - Report generation
- `NotificationService` - Notification management
- `AuditLogService` - Audit logging

Database:

- Entity Framework Core with SQL Server
- Migrations for schema management
- Soft delete for users

Background Services:

- `NotificationBackgroundService` - Processes notifications and due date reminders

Business Rules

- All controllers must use DTOs, not entities directly.
- LINQ must be used for data querying, filtering, sorting, and aggregations.
- Configuration values must be stored in appsettings.json, not hardcoded.
- Separation of concerns: Controllers, Services, Models must be separated.
- All services implement interfaces for dependency injection and testability.
- BaseController provides common functionality for all controllers.
- Global exception middleware catches all unhandled exceptions and returns consistent error format.

---

15. RESTful API Design Requirements (Non-Functional)

Requirements

- **NFR-API-01:** APIs must follow RESTful design principles.

Business Rules

- RESTful endpoint naming conventions must be followed.
- HTTP methods (GET, POST, PUT, DELETE) must be used appropriately.
- Status codes must be used correctly.

---

16. LINQ Usage Requirements (Mandatory Technical)

Filter Bills by Status

- **FR-LINQ-01:** The system shall filter bills by status from the database.

Technical Requirement:

- **TR-LINQ-01:** The system shall use LINQ queries (Where clause) to filter bills by status.

Business Rules:

- Status values must follow bill lifecycle: Generated, Due, Paid, Overdue.

Calculate Outstanding Dues

- **FR-LINQ-02:** The system shall calculate total outstanding balance.

Technical Requirement:

- **TR-LINQ-02:** The system shall use LINQ aggregation (Sum) to compute outstanding balances.

Business Rules:

- Outstanding balance = Sum of unpaid bill amounts.
- Calculation must use LINQ Sum aggregation.

Monthly Revenue Report

- **FR-LINQ-03:** The system shall generate monthly revenue report grouped by month.

Technical Requirement:

- **TR-LINQ-03:** The system shall use LINQ GroupBy to group bills by month, then use Sum aggregation to calculate total revenue per month.

Business Rules:

- Revenue must be grouped by Year and Month.
- Total revenue per month must use LINQ Sum aggregation.

Consumption by Utility Type

- **FR-LINQ-04:** The system shall generate consumption report grouped by utility type.

Technical Requirement:

- **TR-LINQ-04:** The system shall use LINQ GroupBy to group consumption by utility type, then use aggregations (Sum, Average).

Business Rules:

- Consumption must be grouped by Utility Type.
- Total and average consumption must use LINQ aggregations.

Consumer-Wise Billing Summary

- **FR-LINQ-05:** The system shall generate consumer-wise billing summary with aggregations.

Technical Requirement:

- **TR-LINQ-05:** The system shall use LINQ GroupBy to group bills by consumer, then use multiple aggregations.

Business Rules:

- Bills must be grouped by Consumer.
- Must calculate aggregations such as Total Billed, Total Paid, Outstanding Balance.
- Must use LINQ GroupBy with multiple aggregations.

---

17. Frontend Architecture Requirements (Non-Functional)

Requirements

- **NFR-FRONT-01:** Frontend must use Angular with modular architecture.
- **NFR-FRONT-02:** Frontend must implement route guards for role-based access control.
- **NFR-FRONT-03:** Frontend must use HTTP interceptor for JWT token injection.
- **NFR-FRONT-04:** Frontend must use Angular Reactive Forms for all forms.
- **NFR-FRONT-05:** Frontend must use Angular Material or Bootstrap for UI components.
- **NFR-FRONT-06:** Frontend must implement mandatory pages:
  - Login
  - Dashboard
  - Consumer Management
  - Meter Reading Entry
  - Bills & Invoices
  - Payment History
  - Reports
  - Admin Panel
- **NFR-FRONT-07:** Frontend must use HTTPClient for all API calls.

Implementation Details

Angular Version: Latest (v20.3.0)

Project Structure:

- `src/app/components/` - Feature components organized by role
  - `admin/` - Admin pages and components
  - `billing-officer/` - Billing Officer pages
  - `consumer/` - Consumer pages
  - `account-officer/` - Account Officer pages
  - `auth/` - Authentication components
- `src/app/auth-guard/` - Route guards
- `src/app/services/` - Angular services for API calls
- `src/app/models/` - TypeScript models/interfaces
- `src/app/config/` - Configuration files (navigation config)

Route Guards:

- `authGuard` - Protects authenticated routes
- `loginGuard` - Redirects authenticated users away from login

HTTP Interceptor:

- `authInterceptor` - Injects JWT token in Authorization header

Implemented Pages:

Admin:

- `/admin/reports` - Reports & Analytics
- `/admin/users` - User Management
- `/admin/utilities` - Utility Types
- `/admin/tariffs` - Tariff Plans
- `/admin/billing` - Billing Cycles
- `/admin/connections` - Connections
- `/admin/requests` - Service Requests
- `/admin/logs` - Audit Logs

Billing Officer:

- `/billing/meter-readings/entry` - Meter Reading Entry
- `/billing/meter-readings/history` - Reading History
- `/billing/bill-generation` - Bill Generation

Consumer:

- `/consumer/dashboard` - Dashboard
- `/consumer/bills` - View Bills and Invoices
- `/consumer/payments` - Payment History
- `/consumer/consumption` - Consumption Details
- `/consumer/requests` - My Service Requests

Account Officer:

- `/account-officer/dashboard` - Dashboard
- `/account-officer/payments` - Track Payments
- `/account-officer/outstanding` - Outstanding Balances
- `/account-officer/billing-summary` - Consumer Billing Summary

Auth:

- `/auth` - Login
- `/register` - Registration

Navigation:

- Role-based navigation menu (`navigation.config.ts`)
- Default routes per role

Business Rules

- Modular architecture implemented with feature-based component organization.

- Routes must be protected with guards and role checking.
- All API calls must include JWT token in Authorization header (via interceptor).
- Forms use Angular Reactive Forms with client-side validation.
- Angular Material is installed and used for UI components.
- Role-based navigation shows only relevant menu items per role.

---

18. Key LINQ Demonstrations (Mandatory)

Required LINQ Operations

- **Filtering:** `.Where()` - Filter bills by status
- **Sorting:** `.OrderBy()`, `.OrderByDescending()` - Sort bills and payments
- **Aggregation - Sum:** `.Sum()` - Total revenue, outstanding amounts
- **Aggregation - Average:** `.Average()` - Average consumption
- **Aggregation - GroupBy:** `.GroupBy()` - Monthly revenue, consumption by utility type, consumer-wise billing summary

LINQ Usage Examples

Filter Bills by Status:

```
var bills = _context.Bills
    .Where(b => b.Status == "Overdue")
    .ToList();
```

Calculate Outstanding Balance:

```
var outstanding = _context.Bills
    .Where(b => b.Status != "Paid")
    .Sum(b => b.TotalAmount + b.PenaltyAmount);
```

Monthly Revenue Report:

```
var monthlyRevenue = _context.Bills
    .GroupBy(b => new { b.GenerationDate.Year, b.GenerationDate.Month })
    .Select(g => new {
        Year = g.Key.Year,
        Month = g.Key.Month,
        TotalRevenue = g.Sum(b => b.TotalAmount)
    })
    .ToList();
```

Consumption by Utility Type:

```
var consumptionByUtility = _context.Bills
    .GroupBy(b => b.Connection.UtilityType.Name)
    .Select(g => new {
        Utility = g.Key,
        TotalConsumption = g.Sum(b => b.Consumption),
        AvgConsumption = g.Average(b => b.Consumption)
    })
    .ToList();
```

Consumer-Wise Billing Summary:

```
var consumerSummary = _context.Bills
    .GroupBy(b => b.Connection.User)
    .Select(g => new {
        Consumer = g.Key,
        TotalBilled = g.Sum(b => b.TotalAmount),
        TotalPaid = g.Sum(b => b.PaymentAmount ?? 0),
        Outstanding = g.Where(b => b.Status != "Paid").Sum(b => b.TotalAmount + b.PenaltyAmount)
    })
    .ToList();
```

System Overview

Backend:

- 13 Controllers
- 14 Services
- 46 DTOs
- Entity Framework Core with SQL Server
- JWT Authentication & Role-Based Authorization
- Global Exception Handling
- Background Services for Notifications

Frontend:

- Angular v20.3.0
- 24+ Pages across 4 roles
- Route Guards & HTTP Interceptors
- Reactive Forms
- Angular Material UI
- Role-Based Navigation

Roles:

- **Admin** - Full system management and configuration
- **Billing Officer** - Meter readings and bill generation
- **Account Officer** - System-wide payment and balance tracking
- **Consumer** - View bills, make payments, manage requests

Key Features:

- JWT-based authentication
- Role-based access control
- Automated bill generation from meter readings
- Payment processing (Cash/Online)
- Real-time notifications
- Due date reminders
- Comprehensive reports with LINQ

- Audit logging
- Utility connection requests
- System-wide analytics and dashboards

Requirements Traceability Matrix (RTM)

Utility Billing System

RTM Legend

- **FR** - Functional Requirement
- **NFR** - Non-Functional Requirement
- **LINQ** - Mandatory LINQ usage
- **TC** - Test Case

RTM Table

Req ID	Requirement Description	Module	Backend API	Frontend Page	LINQ Usage
FR-01	User registration	Auth	POST /api/auth/register	Register Page	-
FR-02	User login with JWT	Auth	POST /api/auth/login	Login Page	-
FR-03	Role-based access control (RBAC)	Auth	Secured APIs	All Pages	-
FR-04	Consumer registration and profile management	Consumer	POST /api/auth/register GET /api/user/{id} PUT /api/user/{id}	Register Page Profile View	-
FR-05	Store utility connection details (type, meter number, tariff plan)	Connection	POST /api/connection GET /api/connection/{id}	Connection Management /admin/connections	-
FR-06	Monthly meter reading entry	Meter Reading	POST /api/meterreading	Meter Reading Entry /billing/meter-readings/entry	-
FR-07	Previous vs current reading validation	Meter Reading	POST /api/meterreading GET /api/meterreading/previous/{connectionId}	Meter Reading Entry	-
FR-08	Consumption unit calculation	Meter Reading	POST /api/meterreading	Meter Reading Entry	-
FR-09	Auto bill generation based on consumption and tariff	Billing	POST /api/bill/generate POST /api/bill/generate/batch	Bill Generation /billing/bill-generation	-
FR-10	Apply taxes, fixed charges, and penalties to bills	Billing	POST /api/bill/generate	Bill Generation	-
FR-11	Bill lifecycle management	Billing	POST /api/bill/generate POST /api/bill/{id}/pay	Bill Generation View Bills	-

Req ID	Requirement Description	Module	Backend API	Frontend Page	LINQ Usage
	(Generated → Due → Paid → Overdue)				
FR-12	Record payments (cash / online – mock)	Payment	POST /api/bill/{id}/pay	View Bills /consumer/bills	-
FR-13	Outstanding balance calculation	Payment	GET /api/payment/my GET /api/bill/outstanding	Payment History Outstanding Balances	LINQ-02
FR-14	Payment history view	Payment	GET /api/payment/my	Payment History /consumer/payments	-
FR-15	Monthly revenue report	Reports	GET /api/report/monthly-revenue-by-billing	Reports & Analytics /admin/reports /account-officer/dashboard	LINQ-01
FR-16	Outstanding dues report	Reports	GET /api/bill/outstanding GET /api/report/summary	Reports & Analytics Outstanding Balances	LINQ-02
FR-17	Consumption by utility type	Reports	GET /api/report/consumption	Reports & Analytics	LINQ-03
FR-18	Consumer-wise billing summary	Reports	GET /api/bill/consumers/summary	Consumer Billing Summary /account-officer/billing-summary	LINQ-04
FR-19	Bill generation notification	Notifications	Background Service	Notification Center	-
FR-20	Due date reminders	Notifications	Background Service	Notification Center	-
FR-21	Admin: Manage users	Admin	GET /api/user POST /api/user PUT /api/user/{id} DELETE /api/user/{id}	User Management /admin/users	-
FR-22	Admin: Manage utility types	Admin	GET /api/utilitytype POST /api/utilitytype PUT /api/utilitytype/{id} DELETE /api/utilitytype/{id}	Utility Types /admin/utilities	-
FR-23	Admin: Manage tariffs	Admin	GET /api/tariff POST /api/tariff PUT /api/tariff/{id} DELETE /api/tariff/{id}	Tariff Plans /admin/tariffs	-
FR-24	Admin: Manage billing cycles	Admin	GET /api/billingcycle POST /api/billingcycle PUT /api/billingcycle/{id} DELETE /api/billingcycle/{id}	Billing Cycles /admin/billing	-
FR-25	Consumer: View bills and invoices	Consumer	GET /api/bill/my GET /api/bill/my/{id}	View Bills /consumer/bills	-
FR-26	Consumer: View payment history	Consumer	GET /api/payment/my	Payment History /consumer/payments	-
FR-27	Consumer: View	Consumer	GET /api/report/consumption/my GET /api/report/dashboard	Consumption Details Dashboard /consumer/consumption	-

Req ID	Requirement Description	Module	Backend API	Frontend Page	LINQ Usage
	consumption details				
FR-28	Billing Officer: Enter meter readings	Billing Officer	POST /api/meterreading	Meter Reading Entry /billing/meter-readings/entry	-
FR-29	Billing Officer: Generate bills	Billing Officer	POST /api/bill/generate POST /api/bill/generate/batch	Bill Generation /billing/bill-generation	-
FR-30	Account Officer: Track payments	Account Officer	GET /api/payment/audit	Track Payments /account-officer/payments	-
FR-31	Account Officer: Track outstanding balances	Account Officer	GET /api/bill/outstanding	Outstanding Balances /account-officer/outstanding	LINQ-02

Mandatory LINQ Requirements

Req ID	LINQ Requirement	Module	Backend API	LINQ Operation	Test Case ID	Status
LINQ-01	Total revenue per month	Reports	GET /api/report/monthly-revenue-by-billing	GroupBy(), Sum()	TC-L1	Impleme
LINQ-02	Outstanding dues calculation	Payment/Reports	GET /api/bill/outstanding GET /api/report/dashboard/summary	Where(), Sum()	TC-L2	Impleme
LINQ-03	Average consumption	Reports	GET /api/report/average-consumption	GroupBy(), Average()	TC-L3	Impleme
LINQ-04	Consumer-wise billing summary	Reports	GET /api/bill/consumers/summary	GroupBy(), Sum(), Count()	TC-L4	Impleme
LINQ-05	Filter bills by status	Billing	GET /api/bill/outstanding GET /api/bill/pending	Where()	TC-L5	Impleme
LINQ-06	Sort bills and payments	Billing/Payment	GET /api/bill/my GET /api/payment/my	OrderBy(), OrderByDescending()	TC-L6	Impleme
LINQ-07	Consumption by utility type	Reports	GET /api/report/consumption	GroupBy(), Sum()	TC-L7	Impleme

Non-Functional Requirements (NFR)

Req ID	Requirement Description	Module	Implementation	Test Case ID	Status
NFR-01	RESTful API design	All	All controllers follow REST conventions	TC-N1	Implemented
NFR-02	Centralized exception handling	All	GlobalExceptionMiddleware	TC-N2	Implemented
NFR-03	Secure handling of consumer and billing data	All	JWT authentication, role-based access, DTOs	TC-N3	Implemented
NFR-04	Maintainable and scalable codebase	All	Clean architecture, services, interfaces	TC-N4	Implemented
NFR-05	DTO usage to avoid exposing entities	All	46 DTOs in Models/Dto/	TC-N5	Implemented
NFR-06	Global exception handling middleware	All	GlobalExceptionMiddleware.cs	TC-N6	Implemented

Req ID	Requirement Description	Module	Implementation	Test Case ID	Status
NFR-07	Configuration via appsettings.json	All	JWT, database config in appsettings.json	TC-N7	Implemented
NFR-08	Angular modular architecture	Frontend	Feature modules by role	TC-N8	Implemented
NFR-09	Route guards for role-based access	Frontend	authGuard, loginGuard	TC-N9	Implemented
NFR-10	HTTP interceptor for JWT token	Frontend	authInterceptor	TC-N10	Implemented
NFR-11	Angular Reactive Forms	Frontend	Used in all forms	TC-N11	Implemented
NFR-12	Angular Material / Bootstrap	Frontend	Angular Material installed	TC-N12	Implemented

Test Case Mapping

Authentication & Authorization Tests

- **TC-01**: User registration test (AuthControllerTests.cs)
- **TC-02**: User login with JWT test (AuthControllerTests.cs)
- **TC-03**: Role-based access control test (AuthorizationTests.cs)

Consumer & Connection Tests

- **TC-04**: Consumer registration and profile management test (UserControllerTests.cs)
- **TC-05**: Connection management test (ConnectionControllerTests.cs)

Meter Reading Tests

- **TC-06**: Meter reading entry test (MeterReadingControllerTests.cs)
- **TC-07**: Reading validation test (MeterReadingServiceTests.cs)
- **TC-08**: Consumption calculation test (MeterReadingServiceTests.cs)

Billing Tests

- **TC-09**: Bill generation test (BillControllerTests.cs, BillServiceTests.cs)
- **TC-10**: Bill calculation (taxes, charges, penalties) test (BillCalculationServiceTests.cs)
- **TC-11**: Bill lifecycle test (BillServiceTests.cs)

Payment Tests

- **TC-12**: Payment recording test (PaymentControllerTests.cs, PaymentServiceTests.cs)
- **TC-13**: Outstanding balance calculation test (PaymentServiceTests.cs)
- **TC-14**: Payment history test (PaymentControllerTests.cs)

Reports Tests

- **TC-15**: Monthly revenue report test (ReportControllerTests.cs, ReportServiceTests.cs)
- **TC-16**: Outstanding dues report test (ReportControllerTests.cs, ReportServiceTests.cs)
- **TC-17**: Consumption by utility test (ReportControllerTests.cs, ReportServiceTests.cs)
- **TC-18**: Consumer billing summary test (AccountOfficerServiceTests.cs)

Notification Tests

- **TC-19**: Bill generation notification test (Background service)
- **TC-20**: Due date reminder test (Background service)

Admin Tests

- **TC-21**: User management test (UserControllerTests.cs, UserServiceTests.cs)
- **TC-22**: Utility type management test (UtilityTypeControllerTests.cs, UtilityTypeServiceTests.cs)
- **TC-23**: Tariff management test (TariffControllerTests.cs, TariffServiceTests.cs)
- **TC-24**: Billing cycle management test (BillingCycleControllerTests.cs, BillingCycleServiceTests.cs)

Consumer Tests

- **TC-25**: View bills test (BillControllerTests.cs)
- **TC-26**: Payment history test (PaymentControllerTests.cs)
- **TC-27**: Consumption details test (ReportControllerTests.cs)

Billing Officer Tests

- **TC-28**: Meter reading entry test (MeterReadingControllerTests.cs)
- **TC-29**: Bill generation test (BillControllerTests.cs)

Account Officer Tests

- **TC-30**: Track payments test (PaymentControllerTests.cs, AccountOfficerServiceTests.cs)
- **TC-31**: Track outstanding balances test (BillControllerTests.cs, AccountOfficerServiceTests.cs)

LINQ Tests

- **TC-L1**: Monthly revenue LINQ test (ReportServiceTests.cs, AccountOfficerServiceTests.cs)
- **TC-L2**: Outstanding dues LINQ test (PaymentServiceTests.cs, AccountOfficerServiceTests.cs)
- **TC-L3**: Average consumption LINQ test (ReportServiceTests.cs)
- **TC-L4**: Consumer billing summary LINQ test (AccountOfficerServiceTests.cs)
- **TC-L5**: Filter bills by status LINQ test (BillServiceTests.cs)
- **TC-L6**: Sort bills and payments LINQ test (BillServiceTests.cs, PaymentServiceTests.cs)
- **TC-L7**: Consumption by utility LINQ test (ReportServiceTests.cs)

Non-Functional Tests

- **TC-N1**: RESTful API design test (All controller tests)
- **TC-N2**: Exception handling test (GlobalExceptionMiddleware)
- **TC-N3**: Security test (AuthorizationTests.cs)
- **TC-N4**: Code architecture test (Code review)
- **TC-N5**: DTO usage test (Code review)
- **TC-N6**: Exception middleware test (Integration tests)
- **TC-N7**: Configuration test (appsettings.json validation)

Screen-wise Field Details

Notes

1. **Read-only Fields**: Fields marked as "Read-only" are displayed for information purposes and cannot be edited by the user.
2. **Auto Fields**: Fields marked as "Auto" are automatically generated or calculated by the system and do not require user input.

1. Login Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Email	String	Yes	Valid email format	Users
Password	Password	Yes	Minimum 6 characters	Users

2. Register Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Name	String	Yes	Max 100 characters	Users
Email	String	Yes	Unique, valid email format	Users
Password	Password	Yes	Minimum 6 characters	Users

3. Admin – User Management Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Name	String	Yes	Max 100 characters	Users
Email	String	Yes	Unique, valid email format	Users
Password	Password	Yes (Create only)	Minimum 6 characters	Users
Role	String (Enum)	Yes	Admin, Billing Officer, Account Officer, Consumer	Users
Status	String	Yes	Active, Inactive	Users

4. Admin – Utility Type Management Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Name	String	Yes	Unique name, Max 100 characters	UtilityTypes
Description	String	Yes	Max 500 characters	UtilityTypes
Status	String	Yes	Enabled, Disabled	UtilityTypes
BillingCycleId	String	No	Valid billing cycle ID	UtilityTypes, BillingCycles

5. Admin – Tariff Management Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Name	String	Yes	Max 100 characters	Tariffs
UtilityTypeId	String	Yes	Valid utility type ID	Tariffs, UtilityTypes
BaseRate	Decimal	Yes	Value >= 0	Tariffs
FixedCharge	Decimal	Yes	Value >= 0	Tariffs
TaxPercentage	Decimal	Yes	Value >= 0 and <= 100	Tariffs

6. Admin – Billing Cycle Management Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Name	String	Yes	Max 100 characters	BillingCycles
GenerationDay	Integer	Yes	Value between 1-31	BillingCycles
DueDateOffset	Integer	Yes	Value >= 0 (days after generation)	BillingCycles
GracePeriod	Integer	Yes	Value >= 0 (days)	BillingCycles
IsActive	Boolean	Yes	true/false	BillingCycles

7. Admin – Connection Management Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
UserId	String	Yes	Valid user ID	Connections, Users
UtilityTypeId	String	Yes	Valid utility type ID	Connections, UtilityTypes
TariffId	String	Yes	Valid tariff ID	Connections, Tariffs
MeterNumber	String	Yes	Unique meter number	Connections
Status	String	Yes	Active, Inactive	Connections

8. Admin – Utility Request Management Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
RequestId	String	Auto	System generated	UtilityRequests
UserId	String	Read-only	—	UtilityRequests, Users
UtilityTypeId	String	Read-only	—	UtilityRequests, UtilityTypes
Status	String	Read-only	Pending, Approved, Rejected	UtilityRequests
RequestDate	DateTime	Read-only	—	UtilityRequests
TariffId	String	Yes (Approval)	Valid tariff ID	UtilityRequests, Tariffs
MeterNumber	String	Yes (Approval)	Unique meter number	UtilityRequests

9. Billing Officer – Meter Reading Entry Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
ConnectionId	String	Yes	Valid connection ID	MeterReadings, Connections
ConsumerName	String	Read-only	—	MeterReadings, Connections, Users
UtilityName	String	Read-only	—	MeterReadings, Connections, UtilityTypes
MeterNumber	String	Read-only	—	MeterReadings, Connections
PreviousReading	Decimal	Read-only	—	MeterReadings
CurrentReading	Decimal	Yes	Value >= PreviousReading, >= 0	MeterReadings
ReadingDate	DateTime	Yes	Date <= Today	MeterReadings
TariffId	String	Yes	Valid tariff ID	MeterReadings, Tariffs

10. Billing Officer – Bill Generation Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
ReadingId	String	Yes	Valid meter reading ID	Bills, MeterReadings
ConnectionId	String	Read-only	—	Bills, Connections
BillingPeriod	String	Auto	System generated (Month Year)	Bills
GenerationDate	DateTime	Auto	System generated	Bills

Field Name	Data Type	Mandatory	Validation Rules	Related Table
DueDate	DateTime	Auto	Calculated from billing cycle	Bills
PreviousReading	Decimal	Read-only	—	Bills, MeterReadings
CurrentReading	Decimal	Read-only	—	Bills, MeterReadings
Consumption	Decimal	Auto	CurrentReading - PreviousReading	Bills
BaseAmount	Decimal	Auto	Calculated from consumption and tariff	Bills
TaxAmount	Decimal	Auto	Calculated from base amount and tax %	Bills
FixedCharge	Decimal	Auto	From tariff	Bills
PenaltyAmount	Decimal	Auto	Calculated if overdue	Bills
TotalAmount	Decimal	Auto	BaseAmount + TaxAmount + FixedCharge + PenaltyAmount	Bills
Status	String	Auto	Generated	Bills

11. Consumer – Payment Screen

Field Name	Data Type	Mandatory	Validation Rules	Related Table
BillId	String	Yes	Valid unpaid bill ID	Payments, Bills
PaymentMethod	String	Yes	Cash or Online	Payments
ReceiptNumber	String	Conditional	Required if PaymentMethod = Cash	Payments
UpiId	String	Conditional	Required if PaymentMethod = Online	Payments
Amount	Decimal	Read-only	Bill total amount	Payments, Bills

12. Consumer – View Bills Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
BillId	String	Read-only	—	Bills
BillingPeriod	String	Read-only	—	Bills
UtilityName	String	Read-only	—	Bills, Connections, UtilityTypes
MeterNumber	String	Read-only	—	Bills, Connections
PreviousReading	Decimal	Read-only	—	Bills
CurrentReading	Decimal	Read-only	—	Bills
Consumption	Decimal	Read-only	—	Bills
BaseAmount	Decimal	Read-only	—	Bills
TaxAmount	Decimal	Read-only	—	Bills
FixedCharge	Decimal	Read-only	—	Bills
PenaltyAmount	Decimal	Read-only	—	Bills
TotalAmount	Decimal	Read-only	—	Bills
GenerationDate	DateTime	Read-only	—	Bills
DueDate	DateTime	Read-only	—	Bills
Status	String	Read-only	Generated, Due, Paid, Overdue	Bills

13. Consumer – Payment History Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
PaymentId	String	Read-only	—	Payments
BillId	String	Read-only	—	Payments, Bills

Field Name	Data Type	Mandatory	Validation Rules	Related Table
PaymentDate	DateTime	Read-only	—	Payments
Amount	Decimal	Read-only	—	Payments
PaymentMethod	String	Read-only	Cash, Online	Payments
ReceiptNumber	String	Read-only	—	Payments
UpiId	String	Read-only	—	Payments
Status	String	Read-only	Completed, Failed	Payments
UtilityName	String	Read-only	—	Payments, Bills, Connections, UtilityTypes
BillingPeriod	String	Read-only	—	Payments, Bills

14. Consumer – Consumption Details Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Month	String	Read-only	—	MeterReadings
UtilityName	String	Read-only	—	MeterReadings, Connections, UtilityTypes
Consumption	Decimal	Read-only	—	MeterReadings
EstimatedCost	Decimal	Read-only	—	Calculated

15. Consumer – Dashboard Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
OutstandingBalance	Decimal	Read-only	—	Bills (calculated)
MonthlySpending	Decimal	Read-only	—	Payments (calculated)
ActiveConnections	Integer	Read-only	—	Connections (count)
DueBillsCount	Integer	Read-only	—	Bills (count)
ConsumptionTrend	Array	Read-only	—	MeterReadings (grouped by month)

16. Account Officer – Track Payments Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
PaymentId	String	Read-only	—	Payments
PaymentDate	DateTime	Read-only	—	Payments
ConsumerName	String	Read-only	—	Payments, Bills, Connections, Users
UtilityName	String	Read-only	—	Payments, Bills, Connections, UtilityTypes
Amount	Decimal	Read-only	—	Payments
PaymentMethod	String	Read-only	Cash, Online	Payments
Reference	String	Read-only	ReceiptNumber or UpiId	Payments

17. Account Officer – Outstanding Balances Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
BillId	String	Read-only	—	Bills
ConsumerName	String	Read-only	—	Bills, Connections, Users
UtilityName	String	Read-only	—	Bills, Connections, UtilityTypes
BillMonth	String	Read-only	—	Bills
Amount	Decimal	Read-only	—	Bills

Field Name	Data Type	Mandatory	Validation Rules	Related Table
Status	String	Read-only	Due, Overdue, Generated	Bills
DueDate	DateTime	Read-only	—	Bills

18. Account Officer – Consumer Billing Summary Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
ConsumerId	String	Read-only	—	Users
ConsumerName	String	Read-only	—	Users
TotalBilled	Decimal	Read-only	—	Bills (Sum)
TotalPaid	Decimal	Read-only	—	Bills (Sum where Status = Paid)
OutstandingBalance	Decimal	Read-only	—	Bills (Sum where Status != Paid)
OverdueCount	Integer	Read-only	—	Bills (Count where Status = Overdue)

19. Account Officer – Dashboard Screen (Display Only)

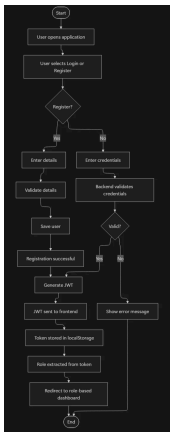
Field Name	Data Type	Mandatory	Validation Rules	Related Table
TotalRevenue	Decimal	Read-only	—	Payments (Sum)
OutstandingDues	Decimal	Read-only	—	Bills (Sum where Status != Paid)
UnpaidBillsCount	Integer	Read-only	—	Bills (Count where Status != Paid)
TotalConsumption	Decimal	Read-only	—	MeterReadings (Sum)
RecentPayments	Array	Read-only	—	Payments (Last 5)
OutstandingByUtility	Array	Read-only	—	Bills (Grouped by UtilityType)

20. Admin – Reports Screen (Display Only)

Field Name	Data Type	Mandatory	Validation Rules	Related Table
TotalConsumers	Integer	Read-only	—	Users (Count where Role = Consumer)
ActiveBillingOfficers	Integer	Read-only	—	Users (Count where Role = BillingOfficer)
ActiveAccountOfficers	Integer	Read-only	—	Users (Count where Role = AccountOfficer)
PendingUtilityRequests	Integer	Read-only	—	UtilityRequests (Count where Status = Pending)
ConnectionsByUtility	Array	Read-only	—	Connections (Grouped by UtilityType)

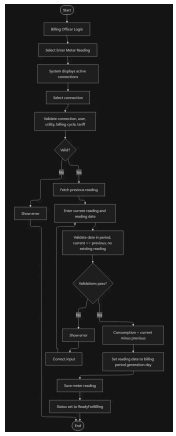
Flowcharts

1. Authentication and Authorization Flow

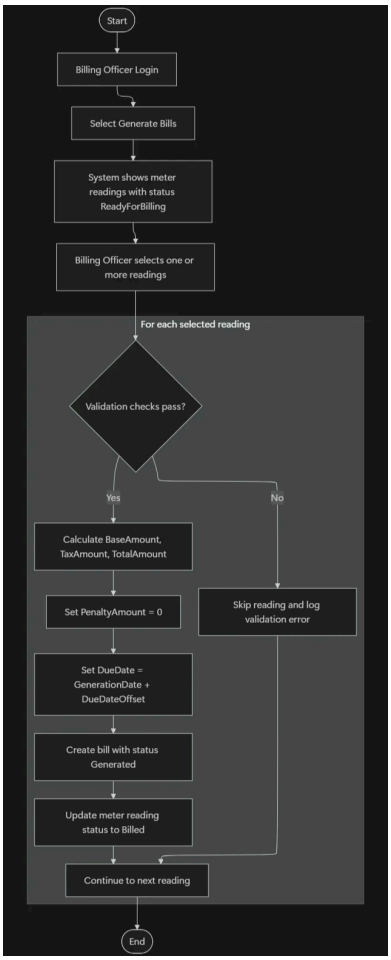


2. Service Request and Connection Setup Flow

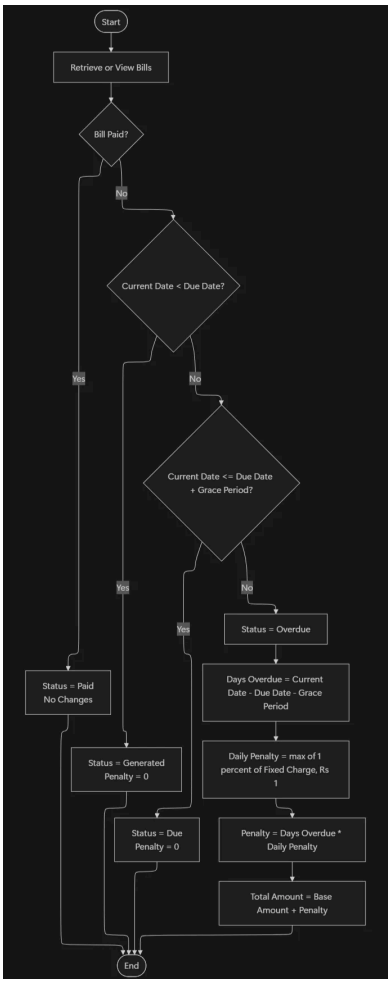
3. Meter reading entry and Validation flow



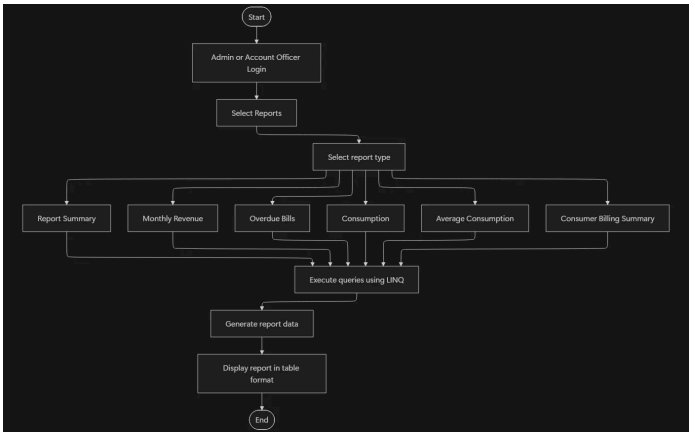
4. Bill generation flow



5. Bill Calculation and Status Update Flow



6. Report Generation Flow



7. Payment Processing Flow

Database Schema

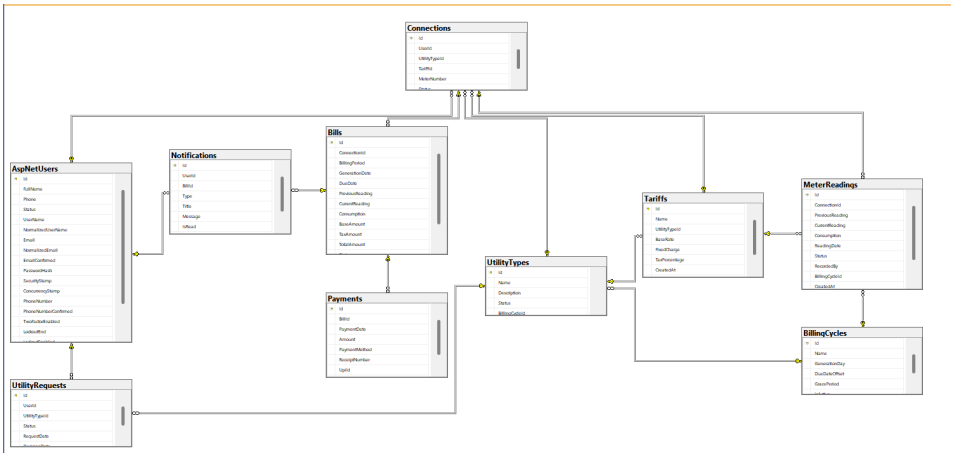


Table Definitions

1. AspNetUsers (User)

Base Table: Inherits from ASP.NET Core Identity `IdentityUser`

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key (inherited from IdentityUser)
UserName	string	Required, Unique	Username for login
Email	string	Required, Unique	Email address
EmailConfirmed	bool		Email confirmation status
PasswordHash	string		Hashed password
PhoneNumber	string		Phone number
PhoneNumberConfirmed	bool		Phone confirmation status
TwoFactorEnabled	bool		Two-factor authentication status
LockoutEnabled	bool		Account lockout status
AccessFailedCount	int		Failed login attempts

Column Name	Data Type	Constraints	Description
LockoutEnd	DateTimeOffset?		Lockout end time
FullName	string(200)	Required	Full name of the user
Status	string(20)	Default: "Active"	User status: Active, Inactive, Deleted
DeletedAt	DateTime?	Nullable	Soft delete timestamp

Navigation Properties:

- `Connections` → Collection of Connection entities
- `UtilityRequests` → Collection of UtilityRequest entities

---

2. UtilityType

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
Name	string(100)	Required	Name of the utility type (e.g., "Electricity", "Water")
Description	string(500)		Description of the utility type
Status	string(20)	Default: "Enabled"	Status: Enabled, Disabled
BillingCycleId	string (GUID)	FK, Nullable	Reference to BillingCycle

Navigation Properties:

- `BillingCycle` → BillingCycle entity (nullable)
- `Tariffs` → Collection of Tariff entities
- `Connections` → Collection of Connection entities
- `UtilityRequests` → Collection of UtilityRequest entities

---

3. BillingCycle

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
Name	string(200)	Required	Name of the billing cycle
GenerationDay	int	Required, Check: 1-28	Day of month when bills are generated
DueDateOffset	int	Required	Days after generation date for due date
GracePeriod	int	Required	Days after due date before penalty applies
IsActive	bool	Default: true	Whether the billing cycle is active

Navigation Properties:

- `UtilityTypes` → Collection of UtilityType entities

Constraints:

- Check constraint: `GenerationDay BETWEEN 1 AND 28`

---

4. Tariff

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
Name	string(200)	Required	Name of the tariff plan
UtilityTypeId	string (GUID)	FK, Required	Reference to UtilityType

Column Name	Data Type	Constraints	Description
BaseRate	decimal(18,2)	Required, Range: ≥0	Rate per unit of consumption
FixedCharge	decimal(18,2)	Required, Range: ≥0	Monthly fixed charge
TaxPercentage	decimal(5,2)	Required, Range: 0-100	Tax percentage applied
CreatedAt	DateTime	Required, Default: UtcNow	Creation timestamp
IsActive	bool	Default: true	Whether the tariff is active

Navigation Properties:

- `UtilityType` → UtilityType entity
- `Connections` → Collection of Connection entities

---

5. Connection

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
UserId	string (GUID)	FK, Required	Reference to User (AspNetUsers)
UtilityTypeId	string (GUID)	FK, Required	Reference to UtilityType
TariffId	string (GUID)	FK, Required	Reference to Tariff
MeterNumber	string(50)	Required, Unique	Unique meter number
Status	string(20)	Default: "Active"	Status: Active, Inactive

Navigation Properties:

- `User` → User entity
- `UtilityType` → UtilityType entity
- `Tariff` → Tariff entity
- `Bills` → Collection of Bill entities

Indexes:

- Unique index on `MeterNumber`

---

6. UtilityRequest

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
UserId	string (GUID)	FK, Required	Reference to User
UtilityTypeId	string (GUID)	FK, Required	Reference to UtilityType
Status	string(20)	Default: "Pending"	Status: Pending, Approved, Rejected
RequestDate	DateTime	Required, Default: UtcNow	Date when request was made
DecisionDate	DateTime?	Nullable	Date when request was approved/rejected

Navigation Properties:

- `User` → User entity
- `UtilityType` → UtilityType entity

---

7. MeterReading

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
ConnectionId	string (GUID)	FK, Required	Reference to Connection
PreviousReading	decimal(18,2)	Range: ≥0	Previous meter reading value
CurrentReading	decimal(18,2)	Range: ≥0	Current meter reading value
Consumption	decimal(18,2)	Range: ≥0	Calculated consumption (Current - Previous)
ReadingDate	DateTime	Required	Date when reading was taken
Status	string(20)	Default: "ReadyForBilling"	Status: ReadyForBilling, Billed
RecordedBy	string(200)	Required	User who recorded the reading
BillingCycleId	string (GUID)	FK, Nullable	Reference to BillingCycle
TariffId	string (GUID)	FK, Required	Reference to Tariff (snapshot at reading time)
CreatedAt	DateTime	Required, Default: UtcNow	Creation timestamp

Navigation Properties:

- `Connection` → Connection entity
- `BillingCycle` → BillingCycle entity (nullable)
- `Tariff` → Tariff entity

Indexes:

- Unique composite index on `(ConnectionId, BillingCycleId)` with filter `Status = 'ReadyForBilling'`
- Index on `ReadingDate`
- Index on `Status`

---

8. Bill

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
ConnectionId	string (GUID)	FK, Required	Reference to Connection
BillingPeriod	string(50)	Required	Billing period (e.g., "July 2024")
GenerationDate	DateTime	Required, Default: UtcNow	Date when bill was generated
DueDate	DateTime	Required	Date when payment is due
PreviousReading	decimal(18,2)	Range: ≥0	Previous meter reading
CurrentReading	decimal(18,2)	Range: ≥0	Current meter reading
Consumption	decimal(18,2)	Range: ≥0	Units consumed
BaseAmount	decimal(18,2)	Range: ≥0	Base amount before tax
TaxAmount	decimal(18,2)	Range: ≥0	Tax amount
PenaltyAmount	decimal(18,2)	Range: ≥0, Default: 0	Late payment penalty
TotalAmount	decimal(18,2)	Range: ≥0	Total amount due
Status	string(20)	Default: "Generated"	Status: Generated, Due, Paid, Overdue

Navigation Properties:

- `Connection` → Connection entity
- `Payments` → Collection of Payment entities

---

9. Payment

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
BillId	string (GUID)	FK, Required	Reference to Bill
PaymentDate	DateTime	Required, Default: UtcNow	Date of payment
Amount	decimal(18,2)	Required, Range: ≥0	Payment amount
PaymentMethod	string(20)	Required	Payment method: Cash, Online
ReceiptNumber	string	Nullable	Receipt number
UpiId	string	Nullable	UPI ID for online payments
Status	string(20)	Default: "Completed"	Payment status

Navigation Properties:

- `Bill` → Bill entity

Indexes:

- Index on `BillId`
- Index on `PaymentDate`

---

10. Notification

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
UserId	string (GUID)	FK, Required	Reference to User
BillId	string (GUID)	FK, Nullable	Reference to Bill (optional)
Type	string(50)	Required	Notification type
Title	string(200)	Required	Notification title
Message	string(1000)	Required	Notification message
IsRead	bool	Default: false	Whether notification is read
CreatedAt	DateTime	Required, Default: UtcNow	Creation timestamp

Navigation Properties:

- `User` → User entity
- `Bill` → Bill entity (nullable)

Indexes:

- Index on `UserId`
- Index on `CreatedAt`
- Composite index on `(BillId, Type, CreatedAt)`

---

11. AuditLog

Column Name	Data Type	Constraints	Description
Id	string (GUID)	PK	Primary key
Timestamp	DateTime	Required, Default: UtcNow	When the action occurred
Action	string(100)	Required	Action type (e.g., "USER_CREATE", "TARIFF_UPDATE")
Details	string(1000)		Additional details about the action
PerformedBy	string(200)	Required	User who performed the action

Indexes:

- Index on `Timestamp`

Table Relationships

One-to-Many Relationships

1. User → Connections

- **Relationship**: One User can have many Connections
- **Foreign Key**: `Connection.UserId` → `User.Id`
- **Delete Behavior**: `Restrict` (cannot delete user with active connections)
- **Navigation**: `User.Connections` ↔ `Connection.User`

2. User → UtilityRequests

- **Relationship**: One User can make many UtilityRequests
- **Foreign Key**: `UtilityRequest.UserId` → `User.Id`
- **Delete Behavior**: `Restrict`
- **Navigation**: `User.UtilityRequests` ↔ `UtilityRequest.User`

3. BillingCycle → UtilityTypes

- **Relationship**: One BillingCycle can be used by many UtilityTypes
- **Foreign Key**: `UtilityType.BillingCycleId` → `BillingCycle.Id`
- **Delete Behavior**: `SetNull` (if billing cycle is deleted, utility type's billing cycle is set to null)
- **Navigation**: `BillingCycle.UtilityTypes` ↔ `UtilityType.BillingCycle`

4. UtilityType → Tariffs

- **Relationship**: One UtilityType can have many Tariffs
- **Foreign Key**: `Tariff.UtilityTypeId` → `UtilityType.Id`
- **Delete Behavior**: `Restrict` (cannot delete utility type with tariffs)
- **Navigation**: `UtilityType.Tariffs` ↔ `Tariff.UtilityType`

5. UtilityType → Connections

- **Relationship**: One UtilityType can have many Connections
- **Foreign Key**: `Connection.UtilityTypeId` → `UtilityType.Id`
- **Delete Behavior**: `Restrict`
- **Navigation**: `UtilityType.Connections` ↔ `Connection.UtilityType`

6. UtilityType → UtilityRequests

- **Relationship**: One UtilityType can have many UtilityRequests
- **Foreign Key**: `UtilityRequest.UtilityTypeId` → `UtilityType.Id`
- **Delete Behavior**: `Restrict`
- **Navigation**: `UtilityType.UtilityRequests` ↔ `UtilityRequest.UtilityType`

7. Tariff → Connections

- **Relationship**: One Tariff can be assigned to many Connections

- **Foreign Key:** `Connection.TariffId` → `Tariff.Id`
- **Delete Behavior:** `Restrict`
- **Navigation:** `Tariff.Connections` ↔ `Connection.Tariff`

8. Connection → Bills

- **Relationship:** One Connection can have many Bills
- **Foreign Key:** `Bill.ConnectionId` → `Connection.Id`
- **Delete Behavior:** `Restrict`
- **Navigation:** `Connection.Bills` ↔ `Bill.Connection`

9. Bill → Payments

- **Relationship:** One Bill can have many Payments (partial payments allowed)
- **Foreign Key:** `Payment.BillId` → `Bill.Id`
- **Delete Behavior:** `Restrict`
- **Navigation:** `Bill.Payments` ↔ `Payment.Bill`

10. User → Notifications

- **Relationship:** One User can have many Notifications
- **Foreign Key:** `Notification.UserId` → `User.Id`
- **Delete Behavior:** `Restrict`
- **Navigation:** `Notification.User` → `User` (no collection navigation)

11. Bill → Notifications

- **Relationship:** One Bill can have many Notifications (optional)
- **Foreign Key:** `Notification.BillId` → `Bill.Id`
- **Delete Behavior:** `SetNull` (if bill is deleted, notification's bill reference is set to null)
- **Navigation:** `Notification.Bill` → `Bill` (no collection navigation)

Many-to-One Relationships

1. MeterReading → Connection

- **Relationship:** Many MeterReadings belong to one Connection
- **Foreign Key:** `MeterReading.ConnectionId` → `Connection.Id`
- **Delete Behavior:** `Restrict`
- **Navigation:** `MeterReading.Connection` → `Connection` (no collection navigation)

2. MeterReading → BillingCycle

- **Relationship:** Many MeterReadings can belong to one BillingCycle (optional)
- **Foreign Key:** `MeterReading.BillingCycleId` → `BillingCycle.Id`
- **Delete Behavior:** `Restrict`
- **Navigation:** `MeterReading.BillingCycle` → `BillingCycle` (no collection navigation)

3. MeterReading → Tariff

- **Relationship:** Many MeterReadings reference one Tariff (snapshot at reading time)

- **Foreign Key**: `MeterReading.TariffId` → `Tariff.Id`
- **Delete Behavior**: `Restrict`
- **Navigation**: `MeterReading.Tariff` → `Tariff` (no collection navigation)

Constraints and Indexes

Primary Keys

All tables use `Id` as the primary key, which is a GUID (string) type.

Foreign Keys

All foreign key relationships are enforced at the database level with appropriate delete behaviors:

- **Restrict**: Prevents deletion of parent record if child records exist
- **SetNull**: Sets foreign key to null when parent is deleted (for optional relationships)

Unique Constraints

1. **Connection.MeterNumber**: Unique constraint ensures no duplicate meter numbers
2. **MeterReading (ConnectionId, BillingCycleId)**: Unique composite index with filter `Status = 'ReadyForBilling'` ensures only one ready-for-billing reading per connection per billing cycle

Check Constraints

1. **BillingCycle.GenerationDay**: Must be between 1 and 28

   ```
   CK_BillingCycle_GenerationDay: [GenerationDay] BETWEEN 1 AND 28
   ```

Indexes

Performance Indexes

1. **Connection**
   - Unique index on `MeterNumber`
2. **MeterReading**
   - Unique composite index on `(ConnectionId, BillingCycleId)` with filter
   - Index on `ReadingDate`
   - Index on `Status`
3. **Payment**
   - Index on `BillId`
   - Index on `PaymentDate`
4. **Notification**
   - Index on `UserId`
   - Index on `CreatedAt`
   - Composite index on `(BillId, Type, CreatedAt)`
5. **AuditLog**
   - Index on `Timestamp`