ChatGPT

ElectricityBillSuite – Comprehensive Implementation Guide

The **ElectricityBillSuite** is an ASP.NET Web Forms application (with an `.sln` solution) designed to manage customer connections, billing, and payments. Its folder structure defines separate areas for account management, user features, admin features, and utilities (e.g. database helpers, email, PDF generation). Below we outline how to implement each component fully, including UI elements and business logic.

Project Structure Overview

- **Solution file:** `ElectricityBillSuite.sln` contains one web project (`ElectricityBillSuite.Web`).
- **Web project (`ElectricityBillSuite.Web`):** Organized into subfolders for **App_Code**, **Account**, **User**, **Admin**, **Assets**, **Scripts**, **Styles**, plus `Web.config` and a master page (`Site.Master`).
- **App_Code:** Contains reusable helper classes (`DbHelper.cs`, `PdfHelper.cs`, `EmailService.cs`).
- **Account:** Contains pages for user login, registration, and admin login (`Login.aspx`, `Register.aspx`, `AdminLogin.aspx`).
- **User:** Customer-facing pages (`Dashboard.aspx`, `ViewBills.aspx`, `PayBill.aspx`, `Transactions.aspx`, `NewConnection.aspx`, `RaiseConcern.aspx`, `Profile.aspx`).
- **Admin:** Admin pages (`Dashboard.aspx`, `ManageConnections.aspx`, `GenerateBills.aspx`, `Revenue.aspx`, `Concerns.aspx`, `LegalNotices.aspx`).
- **Assets:** Static assets like `logo.png`, `hero.jpg`, and a `Notices/` folder where generated PDF notices (e.g. bills, legal notices) are stored.
- **Scripts:** Client-side scripts (`jquery.min.js`, `site.js`) for dynamic behavior.
- **Styles:** CSS files (`site.css`) for consistent styling.
- **Site.Master:** The master page defines a common layout (header, footer, navigation) using `<asp:ContentPlaceHolder>` regions [1].
- **Database:** Contains `schema.sql` (table definitions) and `sample_data.sql`.
- **Docs:** Documentation like `README.md` and `Flowchart.pdf` describing the system.

To **implement** this structure, ensure each directory and file is created as listed. The web project uses ASP.NET Web Forms conventions: each `.aspx` page pairs with a code-behind `.aspx.cs` file, and all pages share the master page (`Site.Master`).

Master pages allow a common layout: put shared HTML (header, menu, footer) into `Site.Master` and use `<asp:ContentPlaceHolder>` for page-specific content [1]. For example, the master page might define navigation links to Dashboard, View Bills, etc., and each content page (like `User/Dashboard.aspx`) fills the `ContentPlaceHolder` with its UI.

Database Schema

The **Database/schema.sql** should define tables for customers, accounts/connections, bills/invoices, payments, and other entities. A typical electricity billing database includes tables like **Customer**, **Account/Connection**, **Billing/Invoice**, **Tariff**, **Admin/User**, and **Concerns/Feedback** [2]. For example, one design uses tables for `customer`, `account`, `admin`, `billing`, `invoice`, and so on [2] [3]. Your `schema.sql` should create:

- **Users/Customers:** Fields for name, address, email, password (hashed), status, etc.
- **Admin:** Admin credentials and roles.
- **Accounts/Connections:** Details of electricity connection (meter number, address, connection status).
- **Bills/Invoices:** Records per billing cycle – consumption, amount due, due date, status.
- **Transactions/Payments:** Records of payments made (date, amount, mode).
- **Concerns/Feedback:** Customer complaints or issues.
- **Tariff:** Rate per unit, fixed charges, etc.

Each table needs appropriate primary keys and foreign keys (e.g., `Billing.customer_id` references `Customer.id`). The `sample_data.sql` can insert demo records (e.g., one admin user, some tariffs). Using SQL Server or another RDBMS, execute these scripts to create and populate the database. Be sure connection strings in `Web.config` (below) point to this database.

Configuration (Web.config)

In **Web.config**, configure:

- **Connection Strings:** Define a `<connectionStrings>` entry for your database. For example:

```
<connectionStrings>
  <add name="DbConn" connectionString="Data Source=.;Initial
Catalog=ElectricityDB;Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

- **Authentication:** Use Forms Authentication. In `<system.web>`:

```
<authentication mode="Forms">
  <forms loginUrl="~/Account/Login.aspx" timeout="30" />
</authentication>
<authorization>
  <deny users="?" />  <!-- Deny anonymous users -->
</authorization>
```

  This forces login for protected pages. You can set `loginUrl` for general users (e.g. `Login.aspx`) and have a separate admin page (`AdminLogin.aspx`) for admin credentials. The Microsoft guidance shows using forms auth with a Users table [4].
- **Other Settings:** Register any needed HTTP handlers/modules (if generating PDFs or using other libraries).

Helpers in App_Code

DbHelper.cs

This static class centralizes database access (using ADO.NET). Typical patterns:

- **Creating commands:** A helper method (similar to [33]) can construct a `SqlCommand` with parameters to avoid SQL injection. For example:

```
public static class DbHelper {
    // Gets the connection string from Web.config
    private static string connString =
ConfigurationManager.ConnectionStrings["DbConn"].ConnectionString;

    // Execute a query and return a DataTable
    public static DataTable GetDataTable(string sql, params
SqlParameter[] parameters) {
        using (SqlConnection conn = new SqlConnection(connString))
        using (SqlCommand cmd = new SqlCommand(sql, conn)) {
            cmd.Parameters.AddRange(parameters);
            conn.Open();
            DataTable dt = new DataTable();
            dt.Load(cmd.ExecuteReader());
            return dt;
        }
    }
    // Similar methods: ExecuteScalar, ExecuteNonQuery, etc.
}
```

  StackOverflow examples show building a command with parameters via a helper [5]. Use `SqlParameter` for each parameter in queries. For instance:

```
// Example usage in code-behind:
string sql = "SELECT COUNT(*) FROM Users WHERE Username=@user AND
PasswordHash=@pass";
int count = (int)DbHelper.ExecuteScalar(sql, new SqlParameter("@user",
username), new SqlParameter("@pass", passwordHash));
```

- **Connection String:** Pull from `Web.config`. Ensure `DbHelper` reads the same connection string used in the app.

EmailService.cs

Implement a service to send emails (e.g. confirmation or concern notifications). A simple class uses `System.Net.Mail.SmtpClient` and `MailMessage`. For example:

```
public class EmailService {
    public static void SendEmail(string toEmail, string subject, string
body) {
```

```
        MailMessage msg = new MailMessage();
        msg.From = new MailAddress("noreply@yourelectriccompany.com",
"ElectricityBoard");
        msg.To.Add(toEmail);
        msg.Subject = subject;
        msg.Body = body;
        msg.IsBodyHtml = true;

        SmtpClient smtp = new SmtpClient();
        smtp.Host = "smtp.youremail.com";
        smtp.Port = 587;
        smtp.EnableSsl = true;
        smtp.Credentials = new NetworkCredential("smtp_user",
"smtp_password");
        smtp.Send(msg);
    }
}
```

Set the SMTP host, port, and credentials according to your email provider. This follows the standard .NET email pattern [6] [7]. Call `EmailService.SendEmail(...)` when you need to notify a user (e.g., after registration or when a bill is generated).

PdfHelper.cs

Use a PDF library (e.g. **iTextSharp** or **iText7**) to generate PDF files for bills and notices. For example, using iText7 (see [21]):

```
public class PdfHelper {
    public static void CreateBillPdf(string pdfPath, string customerName,
decimal amount, DateTime dueDate) {
        using var writer = new iText.Kernel.Pdf.PdfWriter(pdfPath);
        using var pdfDoc = new iText.Kernel.Pdf.PdfDocument(writer);
        using var document = new iText.Layout.Document(pdfDoc);

        // Add content
        document.Add(new iText.Layout.Element.Paragraph("Electricity Bill")
            .SetFontSize(18).SetBold());
        document.Add(new iText.Layout.Element.Paragraph($"Name:
{customerName}"));
        document.Add(new iText.Layout.Element.Paragraph($"Amount Due:
{amount:C}"));
        document.Add(new iText.Layout.Element.Paragraph($"Due Date:
{dueDate:yyyy-MM-dd}"));
        // Add more details as needed...
    }
}
```

This mirrors the basic PDF creation shown in [21†L263-L272]: initialize `PdfWriter`, `PdfDocument`, then add paragraphs [8] [9]. After creating the PDF file (`pdfPath`), you can save it to `Assets/`

`Notices/` and optionally email it (with `EmailService`). Ensure `Assets/Notices` exists and is writable by the app.

Account Management (Login / Register)

User Registration (`Register.aspx`)

- **UI:** A form to collect user details: name, email, address, password, etc. Use `<asp:TextBox>` controls and validators (RequiredFieldValidator, RegularExpressionValidator for email).
- **Logic:** In `Register.aspx.cs`, on submission:
- Validate inputs server-side.
- Hash the password (e.g. using SHA256 or a more secure algorithm).
- Insert a new user record in the database (via `DbHelper.ExecuteNonQuery`).
- Optionally send a confirmation email (using `EmailService`).
- Redirect to Login page or show a success message.

Example (pseudo-code):

```
protected void btnRegister_Click(object sender, EventArgs e) {
    string name = txtName.Text.Trim();
    string email = txtEmail.Text.Trim();
    string passwordHash = HashPassword(txtPassword.Text);
    string sql = "INSERT INTO Users (Name, Email, PasswordHash) VALUES
(@n,@e,@p)";
    DbHelper.ExecuteNonQuery(sql,
        new SqlParameter("@n", name),
        new SqlParameter("@e", email),
        new SqlParameter("@p", passwordHash));
    EmailService.SendEmail(email, "Welcome", "Thanks for registering!");
    Response.Redirect("~/Account/Login.aspx");
}
```

User Login (`Login.aspx`)

- **UI:** Form with username (or email) and password fields. Optionally a “Remember me” checkbox.
- **Logic:** When the user submits, check credentials:
- Query the database for a matching user (`SELECT * FROM Users WHERE Email=@e AND PasswordHash=@p`).
- If found, issue an authentication ticket:

```
FormsAuthentication.SetAuthCookie(email, chkRemember.Checked);
Response.Redirect("~/User/Dashboard.aspx");
```

  This uses ASP.NET forms auth (as configured in Web.config) to mark the user as logged in.
- If invalid, show an error message.

Alternatively, use the built-in `<asp:Login>` control which integrates with membership providers [10]. But custom logic is fine for learning.

Reference: Microsoft documentation explains creating a login page and using forms authentication (the `<forms loginUrl>` in Web.config points here) [4]. In that example, they create a login form and validate against a `Users` table [4].

Admin Login (`AdminLogin.aspx`)

Similar to user login, but verifying against an `Admins` table or an `IsAdmin` flag in the Users table. On success, redirect to `~/Admin/Dashboard.aspx`.

Maintain separate sessions or roles: e.g., after `FormsAuthentication.SetAuthCookie`, you can check in pages whether the authenticated user is an admin (via roles in DB or simply by checking `User.Identity.Name` against admin list).

User Pages

Authenticated users (after login) access pages under the **User** folder. Each page should check authentication (if using Forms auth, unauth users get redirected to login automatically). Use `User.Identity.Name` (or session variables) to get the current user's email/ID.

1. **Dashboard.aspx:** Overview of account – show welcome message and summary (e.g. last bill amount, due date). Query DB for the latest bill record for this user and display.
2. **ViewBills.aspx:** List all bills for the user. Retrieve from `Billing` table:

```
string sql = "SELECT BillId, Date, Amount, Status FROM Billing WHERE
CustomerEmail=@e";
DataTable dt = DbHelper.GetDataTable(sql, new SqlParameter("@e",
userEmail));
gridViewBills.DataSource = dt; gridViewBills.DataBind();
```

   Provide links/buttons to download PDF (if generated) or pay.
3. **PayBill.aspx:** Interface to pay a selected bill. For simplicity, implement as:
4. Select a pending bill from a dropdown or query string.
5. Enter payment details (card info or just a placeholder).
6. On submit, update the `Billing` record (`UPDATE Billing SET PaidAmount = Amount, Status='Paid' WHERE BillId=@id`). Insert a record in `Transactions` table.
7. Optionally send a payment receipt email.
8. **Transactions.aspx:** Show past payments. Query `Transactions` table for this user and display as a list or grid.
9. **NewConnection.aspx:** Form for user to request a new electricity connection:
10. Fields: desired address, account type, etc.
11. On submit, insert a record in `Connections` table with status "Pending".
12. Possibly email an acknowledgement.
13. **RaiseConcern.aspx:** Complaint submission form:
14. Fields: subject, details.
15. On submit, insert into `Concerns` or `Feedback` table with user ID and timestamp.
16. Email notification to admin (using `EmailService`).
17. **Profile.aspx:** Allow user to view/edit their personal info. Fields from the `Users` table. On submit, update the DB.

Each `.aspx` page uses the master page for layout (so the header/navigation is consistent). In code-behind, always refer to `Session` or `User.Identity` for the current user. Example:

```
string email = User.Identity.Name;
lblWelcome.Text = "Hello, " + email;
```

Admin Pages

Admins manage the system via pages under **Admin**:

1. **Dashboard.aspx:** Show high-level stats (total customers, pending connections, revenue). Query the DB and display numbers.
2. **ManageConnections.aspx:** List all new connection requests (status = "Pending"). Provide Approve/Reject buttons. Approving updates the request's status (to "Active") and creates an account record. You may also email the customer about approval. Rejecting sets status to "Rejected" and optionally logs a reason.
3. **GenerateBills.aspx:** The core billing logic. Typically run monthly to create bills:
4. For each active customer/connection, calculate consumption: e.g. get meter readings or assume usage, apply tariff rates.
5. Insert a new record into `Billing` with amount, due date, etc.
6. Call `PdfHelper.CreateBillPdf` to generate the bill PDF (save under `Assets/Notices/` with a filename like `Bill_<BillId>.pdf`).
7. Email the bill or notice to the customer. This code can loop through customers:

```
var customers =
DbHelper.GetDataTable("SELECT Email, Name, TariffId FROM Customers");
foreach (DataRow row in customers.Rows) {
    decimal usage = /* calculate based on logic */;
    decimal amount = ComputeBillAmount(usage, tariffRate);
    DateTime due = DateTime.Today.AddDays(15);
    string insertSql = "INSERT INTO Billing(CustomerEmail, Amount,
DueDate, Status) VALUES(@e,@amt,@due,'Unpaid')";
    DbHelper.ExecuteNonQuery(insertSql,
        new SqlParameter("@e", row["Email"]),
        new SqlParameter("@amt", amount),
        new SqlParameter("@due", due));
    // Generate PDF bill
    string pdfPath = Server.MapPath("~/Assets/Notices/Bill_" +
generatedBillId + ".pdf");
    PdfHelper.CreateBillPdf(pdfPath, row["Name"].ToString(), amount,
due);
    // Email user
    EmailService.SendEmail(row["Email"].ToString(), "New Bill
Generated",
        "Your electricity bill is ready. Please see attached.");
}
```

This illustrates using `DbHelper`, `PdfHelper`, and `EmailService` together.

8. **Revenue.aspx:** Show aggregate payments/revenue reports. For example, sum of all paid bills:

```
DataTable dt =
DbHelper.GetDataTable("SELECT SUM(Amount) AS TotalRevenue FROM Billing
WHERE Status='Paid'");
lblRevenue.Text = dt.Rows[0]["TotalRevenue"].ToString();
```

   You could also show charts or monthly breakdowns.
9. **Concerns.aspx:** List all raised concerns (from `Concerns` table). Display user, date, and content. Provide a way to mark as resolved (update status) or reply by email. Example:

```
var issues = DbHelper.GetDataTable("SELECT * FROM Concerns ORDER BY
DateCreated DESC");
GridViewConcerns.DataSource = issues; GridViewConcerns.DataBind();
```

10. **LegalNotices.aspx:** This might display static legal PDF notices. You can list files in `Assets/Notices/` (e.g. `LegalNotice2025.pdf`) or store notices in DB. Provide links so users can download or view them.

Admin pages should also use the master page (perhaps a different menu set). Secure them by checking `User.IsInRole("Admin")` or similar in code, redirecting unauthorized users. The Forms auth config (in `Web.config`) plus code-behind checks ensure only logged-in admins see admin pages.

UI Elements and Master Page

- **Site.Master:** Design a common layout (header with logo, navigation menu, footer). In the master page HTML, include `<asp:ContentPlaceHolder>` controls: one in the `<head>` for page-specific `<title>` or scripts, and one in the `<form>` body for main content [1]. For example:

```
<asp:ContentPlaceHolder ID="head" runat="server"></
asp:ContentPlaceHolder>
...
<asp:ContentPlaceHolder ID="MainContent" runat="server"></
asp:ContentPlaceHolder>
```

- **Navigation:** In `Site.Master`, include links (e.g. `<asp:HyperLink NavigateUrl="~/User/Dashboard.aspx">Dashboard</asp:HyperLink>`). You can show/hide certain links based on user role using `LoginView` or checking `User.IsInRole`.
- **CSS/Styling:** Put shared styles in `Styles/site.css`. For example, set consistent colors and layout. Include `<link href="Styles/site.css" rel="stylesheet" />` in the master page `<head>`.
- **Scripts:** Include jQuery and `site.js` for any dynamic UI (e.g. confirm dialogs on deletes, form enhancements). Reference scripts in master page or page-specific content.
- **Responsive UI:** Ensure forms and grids are laid out using tables or CSS for readability. Use ASP.NET validators to enforce input rules.

Summary of Key Implementation Steps

- **Set up database:** Run `schema.sql` to create tables and `sample_data.sql` to populate initial data. Verify tables match the code logic (column names, types).
- **Configure Web.config:** Add connection string and authentication as described. Enable required assemblies (e.g. iTextSharp DLL in bin).
- **Implement helpers:** Code `DbHelper`, `EmailService`, and `PdfHelper` as above. Test each (e.g. test sending an email or creating a PDF via a simple console or page).
- **Build Account pages:**
- `Register.aspx` – capture inputs, hash password, insert user, send email.
- `Login.aspx` – verify credentials, call `FormsAuthentication.SetAuthCookie`.
- `AdminLogin.aspx` – similar to `Login.aspx`, but check admin credentials.
- **Build User pages:** Use GridViews or Lists to display data. In each code-behind, use `DbHelper` to query/update data and bind to controls. Handle events (button clicks) to update the database. Example: In `PayBill.aspx`, update bill status on payment.
- **Build Admin pages:** Similarly, query pending items and update them based on admin actions. Example: In `ManageConnections.aspx`, a GridView with Approve/Reject buttons that update the `Connections` table.
- **Master Page and Styling:** Ensure every content page references `MasterPageFile="~/Site.Master"` and has `<asp:Content>` tags matching placeholders. Use CSS for layout.

By following this design – using forms authentication, ADO.NET for DB access, and helper classes for email/PDF – you will have a **complete implementation** of the ElectricityBillSuite with a working UI. In particular, the helper patterns (DB, Email, PDF) and ASP.NET techniques (Master pages, Login control, GridView) come from standard ASP.NET practices [4] [8].

References: We incorporated patterns from Microsoft and community sources, for example configuring forms authentication [4], using a static DB helper for queries [5], creating Master Pages [1], sending email with `SmtpClient` [6] [7], and generating PDFs with iTextSharp [8]. These guidelines ensure that each file and UI component is implemented correctly according to the given structure.

---

[1] Creating a Site-Wide Layout Using Master Pages (C#) | Microsoft Learn
https://learn.microsoft.com/en-us/aspnet/web-forms/overview/older-versions-getting-started/master-pages/creating-a-site-wide-layout-using-master-pages-cs

[2] [3] Electricity Billing System Database design – Student Project Guidance & Development
https://studentprojectguide.com/project-report/database-design/electricity-billing-system-database-design/

[4] Use ASP.NET forms-based authentication - ASP.NET | Microsoft Learn
https://learn.microsoft.com/en-us/troubleshoot/developer/webapps/aspnet/development/forms-based-authentication

[5] daab - Lightweight ADO.NET Helper Class - Stack Overflow
https://stackoverflow.com/questions/3781426/lightweight-ado-net-helper-class

[6] [7] Sending a Simple Email Using SmtpClient in C#
https://www.c-sharpcorner.com/UploadFile/87b416/sending-a-simple-email-using-smtpclient-in-C-Sharp/

[8] [9] Create a PDF With iText in C#/.NET (Formerly iTextSharp)
https://code-maze.com/csharp-pdf-manipulation-with-itext/

[10] How to: Create an ASP.NET Login Page | Microsoft Learn
https://learn.microsoft.com/en-us/previous-versions/aspnet/ms178331(v=vs.100)