

Pahana Education Billing System

Prepared by: Krishan Abeywardhana

KD/BSCSD/19/05

Date: 12 August 2025

Table of Contents

List of Figures	3
List of Tables	4
Acknowledgment	1
Introduction.....	2
Task A.....	4
System Design of Pahana Education.....	4
Admin/Staff Requirements	4
Customer Requirements	7
Use Case Diagram.....	9
Class Diagram	10
Entity–Relationship Diagram (Logical)	11
Sequence Diagrams	12
Task B	13
System Design Patterns.....	13
Web Application Structure.....	13
System Servlets – Overview	14
Authentication & Session.....	14
User and Role Management	16
Task C	26
Testing Plans	26
Task 4.....	27
Version Control & Collaboration (GitHub)	27
Conclusion	28
References.....	31

List of Figures

Figure 1: Use Case Diagram – Customer and Staff/Admin interactions	9
Figure 2: Class Diagram – Core entities and relationships.....	10
Figure 3: ER Diagram – Database schema overview	11
Figure 4: Consolidated Sequence – Authentication, item, cart, billing, reporting.....	12

List of Tables

Table 1: Representative functional test cases 26

Acknowledgment

I extend my sincere thanks to all mentors, peers, and contributors who guided and supported the development of the Pahana Education Billing System. Their feedback and collaboration were invaluable throughout the project. First, I am grateful to my academic supervisors for their patient guidance, critical questions, and timely reviews that sharpened the scope and quality of this work. I also thank the lecturers who shared domain knowledge on educational administration and billing workflows, helping align technical choices with real operational needs. My peers deserve special appreciation for countless discussions, code reviews, and late-night debugging sessions that turned ideas into reliable features. I acknowledge the staff and test users at Pahana Education who participated in interviews, walkthroughs, and user acceptance testing; their insights directly shaped the interface, reports, and overall user experience. I am indebted to open-source communities and maintainers whose tools, documentation, and examples formed the backbone of the stack. Finally, I thank my family and friends for their encouragement and understanding during long development cycles. Any remaining errors are mine. This project stands on the combined effort of many people, and I am honoured to recognize their contribution here. Your belief in this effort kept the vision moving forward.

Introduction

The **Pahana Education Billing System (PEBS)** is a web-based application that centralizes billing and day-to-day finance operations for educational institutes. It was conceived to replace manual fee collection and scattered spreadsheets with a single, reliable platform that is secure, auditable, and easy to use. From first login to final bill print, the system guides staff through a streamlined workflow that reduces errors, shortens processing time, and provides managers with the visibility they need to make informed decisions.

Upon entry, users are welcomed by a focused landing page that clearly prompts them to **log in**. Authentication protects all feature access by role. After a successful login, administrators arrive at a **dashboard** that summarizes recent activity: charts highlight transaction counts and amounts for the last seven days, a donut chart shows **user-role distribution**, and quick tiles present totals such as **Total Users** and **Total Items**. This immediate overview helps staff spot trends, verify that daily work is progressing, and detect anomalies early.

Core operations begin with **customer onboarding**. The **Register New Customer** form captures the essentials—email, password, name, address, and phone—and issues an account number. This creates a clean separation between user credentials and customer profile while enabling a single source of truth for future billing. Administrative staff can maintain these records through **User Management**, a tabular view that lists user IDs, emails, roles, account numbers, linked customer names, and actions for edit or delete. This page supports quick corrections and routine updates without navigating deep menus.

PEBS includes lightweight **item and pricing** management. Staff can add items with their prices and receive clear success feedback. The **Available Items** page shows each item with its ID, name, and price, and provides an inline quantity box with **Add to Cart** controls to speed up counter operations. Once the cart is ready, the system produces a **Bill Summary** with the bill ID, date, totals, and line items. A **print-ready** view ensures invoices can be handed to the customer or stored as records without further formatting.

To reduce training time and improve self-service, the platform offers a built-in **Help/FAQ manager**. Administrators can maintain a knowledge base by defining a question, a short clue, and a clear answer, all editable from a grid. This content appears to end users as step-by-step guidance for common tasks such as logging in, adding items to the cart, generating bills, and exporting reports.

Navigation is consistent and task-oriented. The top menu exposes the primary journeys—**Dashboard, Create Bill, View Items, Help, Manage Users, Add Item, Reports, Register Customer, and Manage Help**—so staff can move quickly between operational and supervisory tasks. Behind the interface, the design follows a modular, layered approach with distinct entities (User, Customer, Item, Bill, BillItem, Help) and clear separation between presentation, business rules, and data access. This structure keeps the codebase maintainable and ready for future extensions such as discounts, taxes, exports, or email notifications.

In short, PEBS unifies customer records, item pricing, cart-based billing, knowledge sharing, and analytics into a cohesive, role-aware system that is practical for daily use and robust enough for institutional needs.

Task A

System Design of Pahana Education

Admin/Staff Requirements

Core capabilities:

- Manage users (create, update, deactivate).

User Management

User ID	Email	Role	Account Number	Customer Name	Customer Email	Actions
15	dinethk222@gmail.com	User	10	Dineth	dinethk222@gmail.com	Edit Delete
11	Krishan@gmail.com	User	6	Krishan Digana	Krishan@gmail.com	Edit Delete
12	admin@mail.com	Admin	7	Admin	admin@mail.com	Edit Delete
16	abc@gmail.com	User	11	abc	abc@gmail.com	Edit Delete
17	abeywardhanakrishan@gmail.com	User	12	Abeywardhana Chamara	abeywardhanakrishan@gmail.com	Edit Delete
18	deshan@mail.com	User	13	Deshsan	deshan@mail.com	Edit Delete

- Manage items (add, update price, inactivate).

Available Items

Item ID	Item Name	Price (LKR)	Add to Cart	
5	Asus Vivo Book	5000.00	<input type="text" value="1"/>	<button>Add to Cart</button>
4	Chair	8000.00	<input type="text" value="1"/>	<button>Add to Cart</button>
6	Televisison	45000.00	<input type="text" value="1"/>	<button>Add to Cart</button>

- View and export reports (transactions, inventory, user activity).

Transaction Report

Bill ID	Date	Total Amount	Account Number	Customer Name	Customer Email	Calculated By	Item ID	Quantity	Item Price
5	2025-08-03	65000.0	2	Admin	admin@mail.com	12	4	4	8000.0
6	2025-08-03	5000.0	2	Admin	admin@mail.com	12	5	1	5000.0
7	2025-08-03	8000.0	2	Dineth	dinethk222@gmail.com	15	4	1	8000.0
8	2025-08-03	8000.0	2	Dineth	dinethk222@gmail.com	15	4	1	8000.0
9	2025-08-03	8000.0	2	Dineth	dinethk222@gmail.com	15	4	1	8000.0
10	2025-08-03	20000.0	2	Dineth	dinethk222@gmail.com	15	5	4	5000.0
11	2025-08-09	16000.0	2	Dineth	dinethk222@gmail.com	15	4	2	8000.0
12	2025-08-09	25000.0	2	Abeywardhana Chamara	abeywardhanakrishan@gmail.com	17	5	5	5000.0
13	2025-08-10	135000.0	2	Admin	admin@mail.com	12	6	3	45000.0

[Export to CSV](#)

- Review and print bills; resend email copies.

Bill Summary

Bill ID: 13**Date:** 2025-08-10**Total:** Rs. 135000.00

Item Name	Price (LKR)
Television	45000.00

[Print Bill](#)

- Maintain Help/FAQ content for customers.


Manage Help Entries				
ID	Question	Clue	Answer	Actions
1	How to register a new user?	Registration process	Go to the Register page, fill the form, and submit. You will receive a confirmation email if successful.	Edit Delete
2	How to login to the system?	Login instructions	Use your username and password on the Login page. If forgotten, use the 'Forgot Password' link.	Edit Delete
3	How to add items to cart?	Adding items to cart	Search for items, then click the 'Add to Cart' button next to the item to include it in your shopping cart.	Edit Delete
4	How to generate a bill?	Billing process	After adding items to the cart, go to the billing page and click 'Generate Bill'.	Edit Delete
5	How to view transaction reports?	Transaction reports	Navigate to the Transaction Report page from the menu after logging in.	Edit Delete
6	How to export reports as CSV or PDF?	Exporting reports	On the reports page, click the export buttons labeled CSV or PDF to download the reports in your preferred format.	Edit Delete
7	What to do if the PDF export fails to open?	PDF export troubleshooting	Ensure the PDF library is included properly in the server. Try updating or reinstalling the required jars.	Edit Delete

Customer Requirements

Core capabilities:

- Register and login securely.

Register New Customer



- Search/browse items and add to cart.

Available Items

Item ID	Item Name	Price (LKR)	Add to Cart	
5	Asus Vivo Book	5000.00	<input type="text" value="1"/>	<input type="button" value="Add to Cart"/>
4	Chair	8000.00	<input type="text" value="1"/>	<input type="button" value="Add to Cart"/>
6	Televisison	45000.00	<input type="text" value="1"/>	<input type="button" value="Add to Cart"/>

- Generate bill, receive email copy, and download/print bill.

Cart Contents

Item ID	Name	Quantity	Unit Price (LKR)	Subtotal (LKR)
6	Televisison	3	45000.00	135000.00
Total:				135000.00

[Back to Items](#)[Generate Bill](#)

Bill Summary

Bill ID: 13**Date:** 2025-08-10**Total:** Rs. 135000.00

Item Name	Price (LKR)
Televisison	45000.00

[Print Bill](#)

- View own order history and profile details.

Transaction Report

Bill ID	Date	Total Amount	Account Number	Customer Name	Customer Email	Calculated By	Item ID	Quantity	Item Price
5	2025-08-03	65000.0	2	Admin	admin@mail.com	12	4	4	8000.0
6	2025-08-03	5000.0	2	Admin	admin@mail.com	12	5	1	5000.0
7	2025-08-03	8000.0	2	Dineth	dinethk222@gmail.com	15	4	1	8000.0
8	2025-08-03	8000.0	2	Dineth	dinethk222@gmail.com	15	4	1	8000.0
9	2025-08-03	8000.0	2	Dineth	dinethk222@gmail.com	15	4	1	8000.0
10	2025-08-03	20000.0	2	Dineth	dinethk222@gmail.com	15	5	4	5000.0
11	2025-08-09	16000.0	2	Dineth	dinethk222@gmail.com	15	4	2	8000.0
12	2025-08-09	25000.0	2	Abeywardhana Chamara	abeywardhanakrishan@gmail.com	17	5	5	5000.0
13	2025-08-10	135000.0	2	Admin	admin@mail.com	12	6	3	45000.0

[Export to CSV](#)

Use Case Diagram

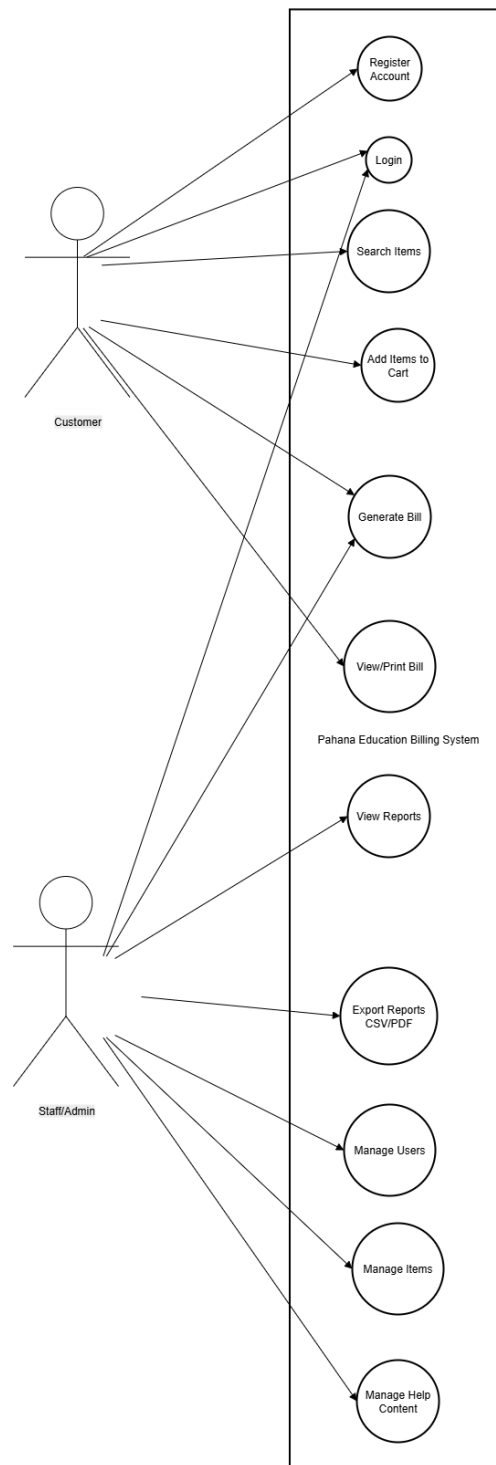


Figure 1: Use Case Diagram – Customer and Staff/Admin interactions

Class Diagram

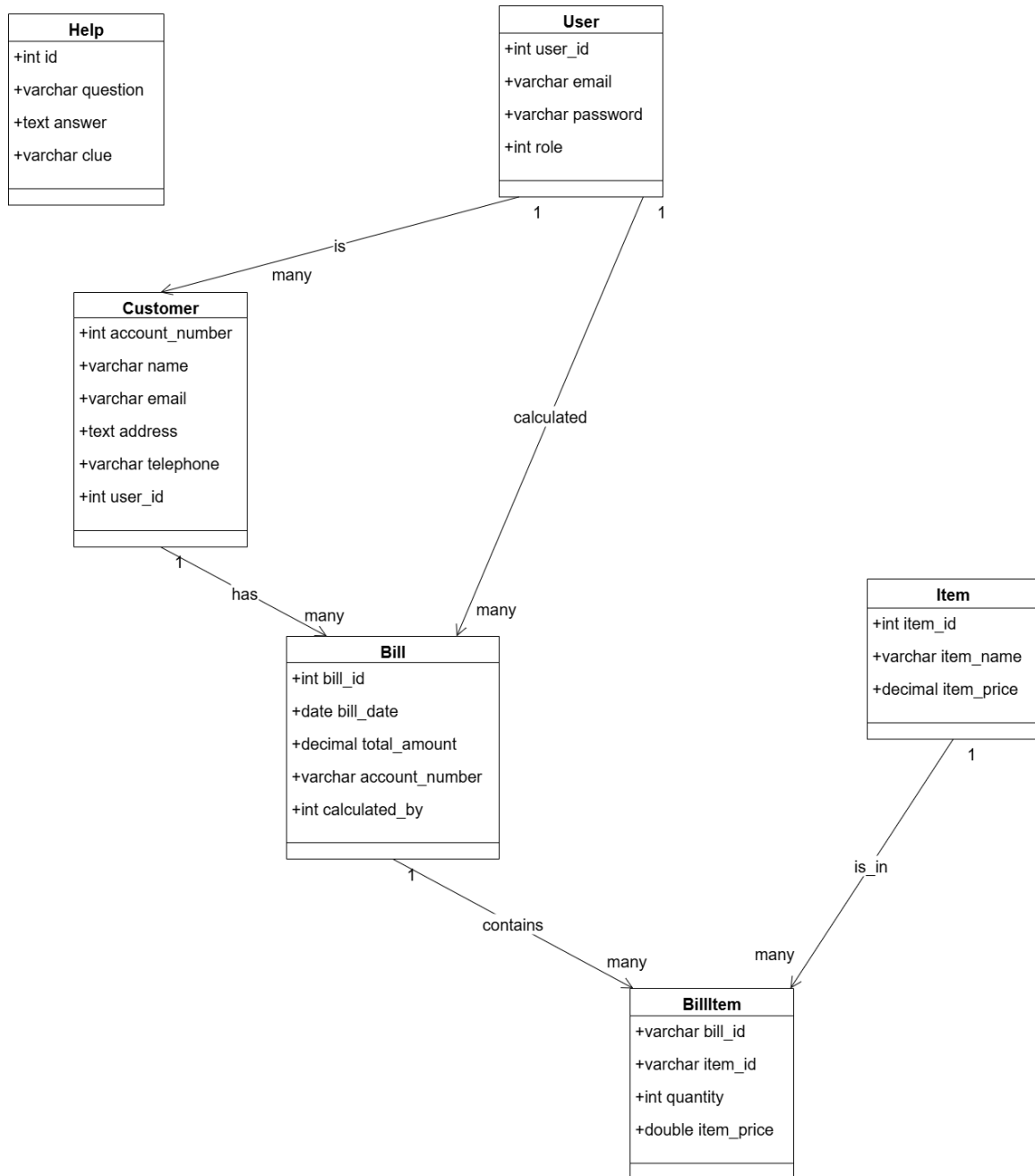


Figure 2: Class Diagram – Core entities and relationships

Entity–Relationship Diagram (Logical)

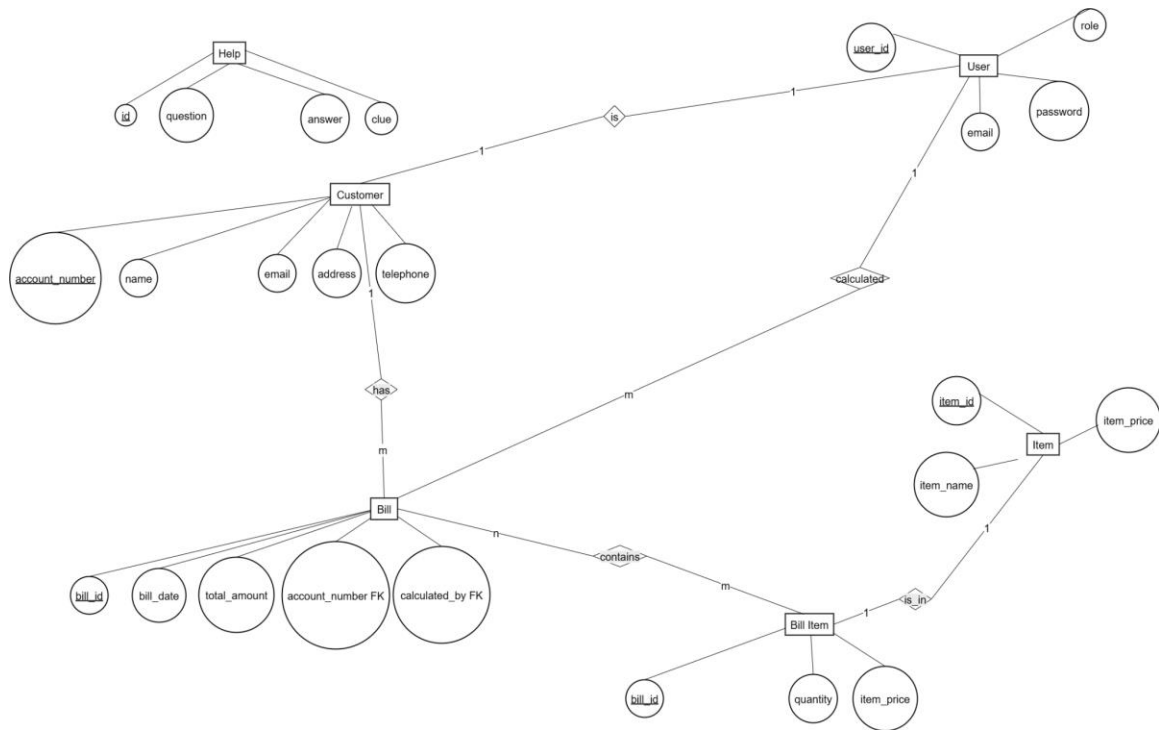


Figure 3: ER Diagram – Database schema overview

Sequence Diagrams

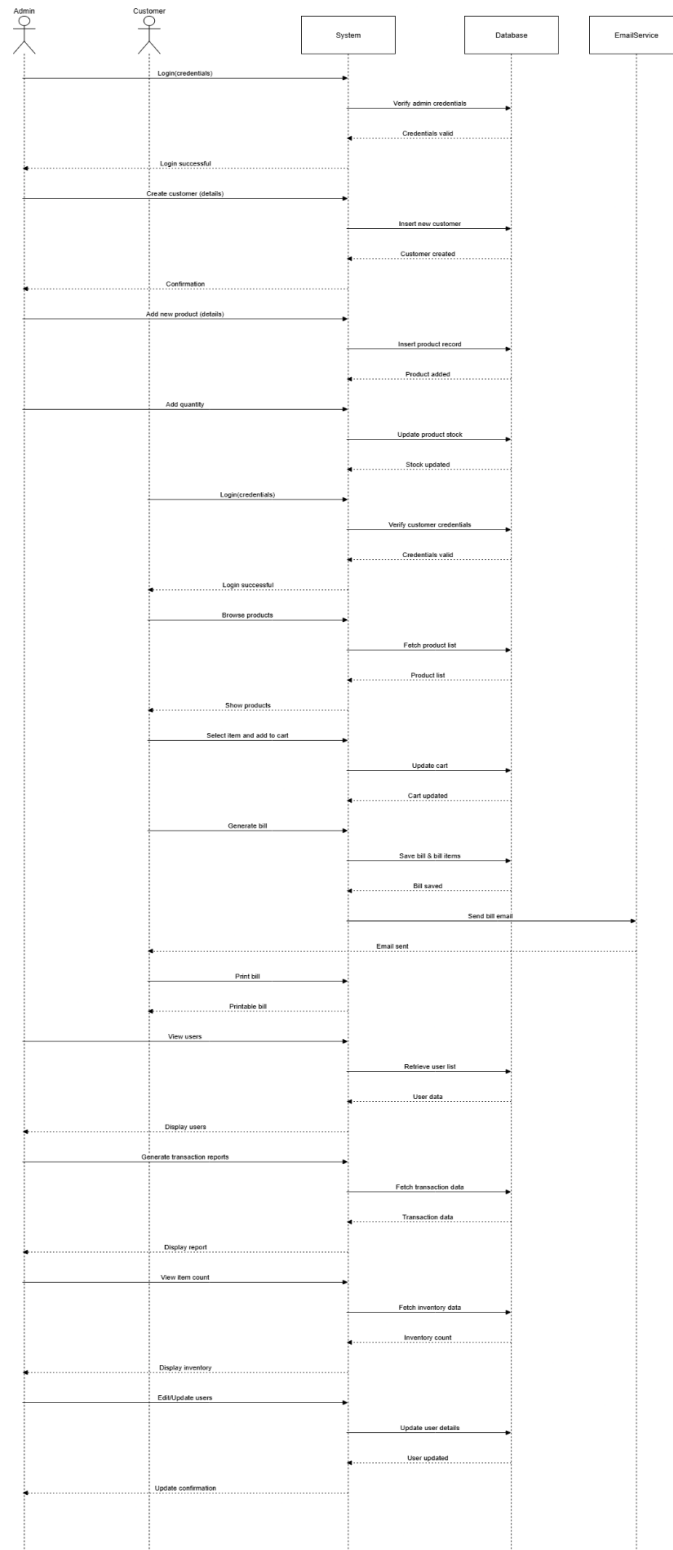


Figure 4: Consolidated Sequence – Authentication, item, cart, billing, reporting

Task B

System Design Patterns

1) Singleton – Database connection/config

Ensure a single shared instance for database connectivity and configuration to reduce connection churn and centralize credentials and retry policies.

2) Factory (or Simple Factory) – User roles

Instantiate role-aware user objects (Customer, Staff/Admin) from a single creation point to keep controller logic minimal and open for extension (e.g., Auditor).

3) Repository – Persistence abstraction

Repositories (UserRepository, ItemRepository, BillRepository) hide query details and return domain models, keeping controllers/services persistence-agnostic and testable.

4) Strategy – Pricing/discounts & tax

Plug-in strategies for discounts and tax rules per institute/term without changing billing code.

5) MVC (Layered Architecture)

Controllers handle HTTP requests, Services enforce business rules (e.g., stock checks, bill totals), Repositories/ORM handle data access, and Views render pages/PDFs.

Web Application Structure

The application follows a clean, layered structure:

- Presentation: Blade/HTML/CSS/JS
- Controllers: Route actions, input validation
- Services: Business logic (billing, email dispatch, reporting)
- Repositories/Models: Data access and mapping
- Infra: Mailer, PDF generation, storage configuration

System Servlets – Overview

This application uses standard **Jakarta Servlets** (namespace `pahanaedu.*`) to implement the billing workflow end-to-end. Each servlet encapsulates one use case and is exposed through a clear URL pattern. Views (JSP/HTML) post to these endpoints, and the servlets coordinate validation, session handling, domain logic, and persistence before forwarding back to views or redirecting.

Authentication & Session

- **LoginServlet** → `/LoginServlet`

Authenticates a user with email/password and establishes the HTTP session (e.g., storing role/account context). On success it redirects to the dashboard; on failure it returns an error state for the login page. Downstream features rely on this session.

```
<html>
  <head>
    </style>
  </head>
  <body>
    <div class="login-container">
      <h2>Login</h2>
      <% if (request.getParameter("error") != null) { %>
        <div class="error">Invalid email or password</div>
      <% }%>
      <form action="LoginServlet" method="post">
        <input type="email" name="email" placeholder="Email Address" required />
        <input type="password" name="password" placeholder="Password" required />
        <input type="submit" value="Login" />
      </form>
      <!-- Help Button -->
      <div style="margin-top: 10px; text-align: center;">
        <a href="help.jsp" target="_blank" style="
          display: inline-block;
          padding: 10px 20px;
          background-color: #007BFF;
          color: white;
          text-decoration: none;
          border-radius: 5px;
          transition: background-color 0.3s ease;
          "
          onmouseover="this.style.backgroundColor = '#0056b3'"
          onmouseout="this.style.backgroundColor = '#007BFF'">
          Need Help?
        </a>
      </div>
    </body>
  </html>
```

```

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String email = request.getParameter("email");
    String password = request.getParameter("password");
    String hashedPassword = hashPassword(password);

    try (Connection conn = DBConnection.getConnection()) {
        PreparedStatement stmt = conn.prepareStatement(
            "SELECT user_id, email, role FROM user WHERE email = ? AND password = ?"
        );
        stmt.setString(1, email);
        stmt.setString(2, hashedPassword);

        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            int userId = rs.getInt("user_id");
            int role = rs.getInt("role");

            HttpSession session = request.getSession();
            session.setAttribute("user_id", userId);
            session.setAttribute("email", email);
            session.setAttribute("role", role);
            session.setAttribute("username", email);

            response.sendRedirect("dashboard.jsp");
        } else {
            response.sendRedirect("login.jsp?error=1");
        }
    } catch (Exception e) {
        e.printStackTrace();
        response.sendRedirect("login.jsp?error=1");
    }
}

```

User and Role Management

- **RegisterUserServlet** → /RegisterUserServlet

Creates a generic user account (credentials and base role).

```

<!-- @page contentType="text/html"; charset="UTF-8" language="java" %>
<html>
<head>
<style>
</style>
</head>
<body>
<div class="form-container">
<h2>Register New Customer</h2>

<% if (request.getParameter("status") != null && request.getParameter("status").equals("success")) { %>
<div class="success">Customer registered successfully!</div>
<% } else if (request.getParameter("status") != null && request.getParameter("status").equals("error")) { %>
<div class="error">Something went wrong. Try again.</div>
<% }%>

<form action="RegisterCustomerServlet" method="post">
<input type="email" name="email" placeholder="Email Address" required />
<input type="password" name="password" placeholder="Password" required />
<input type="text" name="name" placeholder="Full Name" required />
<input type="text" name="address" placeholder="Address" required />
<input type="tel" name="phone" placeholder="Telephone Number" required />
<input type="submit" value="Register Customer" />
</form>
</div>
</body>
</html>

```

```

public class RegisterUserServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String username = request.getParameter("username");
        String password = request.getParameter("password");
        int role = Integer.parseInt(request.getParameter("role"));

        try {
            // Hash the password
            String hashedPassword = hashPassword(password);

            Connection conn = DBConnection.getConnection();
            PreparedStatement ps = conn.prepareStatement("INSERT INTO user(username, password, role) VALUES (?, ?, ?)");
            ps.setString(1, username);
            ps.setString(2, hashedPassword); // store hashed password
            ps.setInt(3, role);

            ps.executeUpdate();
            conn.close();

            response.sendRedirect("manageUsers.jsp");

        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().println("Error registering user.");
        }
    }

    // Password hashing method
    private String hashPassword(String password) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] hashBytes = md.digest(password.getBytes());

        // Convert byte array into hex string
        StringBuilder sb = new StringBuilder();
        for (byte b : hashBytes) {
            sb.append(String.format("%02x", b));
        }
        return sb.toString();
    }
}

```

- **RegisterCustomerServlet** → **/RegisterCustomerServlet**
Registers a customer profile linked to a user (name, address, phone, account number).
- **EditUserServlet** → **/editUser** and **DeleteUserServlet** → **/deleteUser**
Allows admins to update user attributes or deactivate/remove users.
- **UpdateUserCustomerServlet** → **/UpdateUserCustomerServlet**
Synchronizes edits across the user and customer entities (e.g., email or profile changes).

```
public class EditUserServlet extends HttpServlet {
    private static final Long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Log servlet invocation
        System.out.println("EditUserServlet: Servlet invoked for POST /editUser");

        // Get form parameters
        String userIdStr = request.getParameter("userId");
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        String roleStr = request.getParameter("role");
        String name = request.getParameter("name");
        String custEmail = request.getParameter("custEmail");
        String address = request.getParameter("address");
        String telephone = request.getParameter("telephone");
        String status = "success";
        String errorMessage = "";

        // Validate inputs
        if (userIdStr == null || userIdStr.trim().isEmpty() || email == null || email.trim().isEmpty() || roleStr == null || roleStr.trim().isEmpty()) {
            status = "error";
            errorMessage = "User ID, email, and role are required.";
            System.out.println("EditUserServlet: Invalid input parameters");
        } else {
            Connection conn = null;
            PreparedStatement psUser = null;
            PreparedStatement psCustomer = null;
            PreparedStatement psCheckCustomer = null;

```

```
            errorMessage = "No user found with ID " + userIdStr;
        } else {
            // Check if customer record exists
            String sqlCheckCustomer = "SELECT COUNT(*) FROM customer WHERE user_id = ?";
            psCheckCustomer = conn.prepareStatement(sqlCheckCustomer);
            psCheckCustomer.setInt(1, Integer.parseInt(userIdStr));
            rs = psCheckCustomer.executeQuery();
            rs.next();
            boolean customerExists = rs.getInt(1) > 0;

            // Update or insert customer record
            if (customerExists) {
                // Update existing customer
                String sqlCustomer = "UPDATE customer SET name = ?, email = ?, address = ?, telephone = ? WHERE user_id = ?";
                psCustomer = conn.prepareStatement(sqlCustomer);
                psCustomer.setString(1, name != null ? name : "");
                psCustomer.setString(2, custEmail != null ? custEmail : "");
                psCustomer.setString(3, address != null ? address : "");
                psCustomer.setString(4, telephone != null ? telephone : "");
                psCustomer.setInt(5, Integer.parseInt(userIdStr));
                int customerRowsAffected = psCustomer.executeUpdate();
                System.out.println("EditUserServlet: Updated " + customerRowsAffected + " rows in customer table for user_id=" + userIdStr);
            } else if (name != null && !name.trim().isEmpty()) {
                // Insert new customer record if name is provided
                String sqlCustomer = "INSERT INTO customer (name, email, address, telephone, user_id) VALUES (?, ?, ?, ?, ?)";
                psCustomer = conn.prepareStatement(sqlCustomer);
                psCustomer.setString(1, name);
                psCustomer.setString(2, custEmail != null ? custEmail : "");
                psCustomer.setString(3, address != null ? address : "");
                psCustomer.setString(4, telephone != null ? telephone : "");
                psCustomer.setInt(5, Integer.parseInt(userIdStr));
                int customerRowsAffected = psCustomer.executeUpdate();
                System.out.println("EditUserServlet: Inserted " + customerRowsAffected + " rows in customer table for user_id=" + userIdStr);
            }
        }
    }
}
```

Catalog, Cart, and Billing

- **AddItemServlet** → /AddItemServlet

Admin-only endpoint to add catalog items and prices.

```
<body>
  <div class="form-container">
    <h2>Add New Item</h2>

    <form action="AddItemServlet" method="post">
      <label for="item_name">Item Name:</label>
      <input type="text" id="item_name" name="item_name" required>

      <label for="item_price">Item Price:</label>
      <input type="number" id="item_price" name="item_price" step="0.01" required>

      <input type="submit" value="Add Item">
    </form>

    <%
      String status = request.getParameter("status");
      if ("success".equals(status)) {
    %>
      <div class="success">Item added successfully!</div>
    <%
      } else if ("error".equals(status)) {
    %>
      <div class="error">Failed to add item. Please try again.</div>
    <%
      }
    %>
  </div>
</body>
```

```
package pahanaedu;

import java.io.IOException;
import java.sql.*;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.*;

@WebServlet("/AddItemServlet")
public class AddItemServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        String itemName = request.getParameter("item_name");
        double itemPrice = 0;

        try {
            itemPrice = Double.parseDouble(request.getParameter("item_price"));
        } catch (NumberFormatException e) {
            response.sendRedirect("addItem.jsp?status=error");
            return;
        }

        try (Connection conn = DBConnection.getConnection()) {
            String sql = "INSERT INTO item (item_name, item_price) VALUES (?, ?)";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setString(1, itemName);
                stmt.setDouble(2, itemPrice);
                stmt.executeUpdate();
                response.sendRedirect("addItem.jsp?status=success");
            }
        } catch (Exception e) {
            e.printStackTrace();
            response.sendRedirect("addItem.jsp?status=error");
        }
    }
}
```

- **AddToCartServlet** → /AddToCartServlet

Handles cart mutations (add/update quantity). Uses the session to persist cart state between requests.

```
public class AddToCartServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        // Parse parameters safely
        int itemId = Integer.parseInt(request.getParameter("item_id"));
        String itemName = request.getParameter("item_name");
        double itemPrice = Double.parseDouble(request.getParameter("item_price"));

        // Quantity: get from request, default 1 if not present or invalid
        int quantity = 1;
        String qtyParam = request.getParameter("quantity");
        if (qtyParam != null) {
            try {
                quantity = Integer.parseInt(qtyParam);
                if (quantity < 1) quantity = 1; // minimum quantity = 1
            } catch (NumberFormatException e) {
                quantity = 1;
            }
        }

        HttpSession session = request.getSession();

        // Retrieve or create cart: List of Map<String, Object>
        List<Map<String, Object>> cart = (List<Map<String, Object>>) session.getAttribute("cart");
        if (cart == null) {
            cart = new ArrayList<>();
        }

        // Check if item already in cart – update quantity if yes
        boolean found = false;
        for (Map<String, Object> cartItem : cart) {
            if ((Integer) cartItem.get("item_id") == itemId) {
                int currentQty = (Integer) cartItem.get("quantity");
                cartItem.put("quantity", currentQty + quantity);
                found = true;
                break;
            }
        }
    }
}
```

- **GenerateBillServlet** → /GenerateBillServlet

Converts the current cart into a persisted bill with line items and totals, then forwards to a bill summary/print view. This is the transactional boundary of the purchasing flow.

```
public class GenerateBillServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        HttpSession session = request.getSession();
        List<Map<String, Object>> cart = (List<Map<String, Object>>) session.getAttribute("cart");
        Integer userId = (Integer) session.getAttribute("user_id");

        // Validate session data
        if (cart == null || cart.isEmpty()) {
            request.setAttribute("error", "Cart is empty or not found in session.");
            RequestDispatcher rd = request.getRequestDispatcher("error.jsp"); // or back to cart page
            rd.forward(request, response);
            return;
        }
        if (userId == null) {
            request.setAttribute("error", "User not logged in or session expired.");
            RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
            rd.forward(request, response);
            return;
        }

        // Calculate total with quantity
        double total = 0;
        for (Map<String, Object> item : cart) {
            try {
                double price = Double.parseDouble(item.get("item_price").toString());
                int quantity = Integer.parseInt(item.get("quantity").toString());
                total += price * quantity;
            } catch (NumberFormatException e) {
                request.setAttribute("error", "Error parsing item data: " + e.getMessage());
                RequestDispatcher rd = request.getRequestDispatcher("error.jsp");
                rd.forward(request, response);
                return;
            }
        }
    }
}
```



```

public class GenerateBillServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    {
        String subject = "Your Bill from Pahana Education";

        // Set attributes for email template
        request.setAttribute("cart", cart);
        request.setAttribute("total", total);
        request.setAttribute("billId", billId);
        request.setAttribute("billDate", LocalDate.now().toString());

        // Render email body using JSP
        RequestDispatcher rd = request.getRequestDispatcher("emailTemplate.jsp");
        StringWriter sw = new StringWriter();
        HttpServletResponseWrapper responseWrapper = new HttpServletResponseWrapper(response) {
            private final PrintWriter writer = new PrintWriter(sw);

            @Override
            public PrintWriter getWriter() {
                return writer;
            }
        };
        rd.include(request, responseWrapper);
        String htmlBody = sw.toString();

        // Send email
        MailUtil.sendHtmlEmail(to, subject, htmlBody);

        // Save bill ID to session and clear cart
        session.setAttribute("bill_id", billId);
        session.removeAttribute("cart");
        response.sendRedirect("viewBill.jsp");

    } catch (ClassNotFoundException e) {
        request.setAttribute("error", "MySQL Driver not found: " + e.getMessage());
        RequestDispatcher rd = request.getRequestDispatcher("error.jsp");
        rd.forward(request, response);
    } catch (SQLException e) {
        request.setAttribute("error", "Database error: " + e.getMessage());
        RequestDispatcher rd = request.getRequestDispatcher("error.jsp");
        rd.forward(request, response);
    } catch (Exception e) {
        request.setAttribute("error", "Unexpected error: " + e.getMessage());
        RequestDispatcher rd = request.getRequestDispatcher("error.jsp");
        rd.forward(request, response);
    }
}

```

Reporting

- **TransactionReportServlet** → /TransactionReportServlet

Produces transaction summaries (counts, totals, date ranges) for the dashboard and reports pages. Results feed charts and export features.

```
<html>
    <td><%= t.getItemId()%></td>
    <td><%= t.getQuantity()%></td>
    <td><%= t.getItemPrice()%></td>
  </tr>
<% } %>
</tbody>
</table>

<div class="form-container">
  <form action="ExportServlet" method="get">
    <input type="hidden" name="type" value="csv">
    <input type="hidden" name="report" value="transaction">
    <input type="submit" class="export-btn" value="Export to CSV">
  </form>

  <!--
    <form action="ExportServlet" method="get">
      <input type="hidden" name="type" value="pdf">
      <input type="hidden" name="report" value="transaction">
      <input type="submit" class="export-btn" value="Export to PDF">
    </form-->
  </div>

  <% } else { %>
  <p class="no-data">No transactions found.</p>
  <% }
  }
%>
</div>
</body>
</html>
```

```

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    List<Transaction> transactions = new ArrayList<>();

    String url = "jdbc:mysql://localhost:3306/pahana_education?useSSL=false&serverTimezone=UTC";
    String user = "root";
    String pass = "";

    String sql = "SELECT b.bill_id, b.bill_date, b.total_amount, b.account_number, b.calculated_by, "
        + "bi.item_id, bi.quantity, bi.item_price, "
        + "c.name AS customer_name, c.email AS customer_email "
        + "FROM bill b "
        + "LEFT JOIN bill_item bi ON b.bill_id = bi.bill_id "
        + "LEFT JOIN customer c ON b.calculated_by = c.user_id";

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        throw new ServletException("MySQL JDBC Driver not found", e);
    }

    try (Connection con = DriverManager.getConnection(url, user, pass); PreparedStatement pstmt = con.prepareStatement(sql); ResultSet rs = pstmt.executeQuery()) {

        while (rs.next()) {
            transactions.add(new Transaction(
                rs.getInt("bill_id"),
                rs.getDate("bill_date"),
                rs.getDouble("total_amount"),
                rs.getInt("account_number"),
                rs.getInt("calculated_by"),
                rs.getInt("item_id"),
                rs.getInt("quantity"),
                rs.getDouble("item_price"),
                rs.getString("customer_name"),
                rs.getString("customer_email")
            ));
        }
    }
}

```

\

Help/Knowledge Base

- **HelpServlet → /HelpServlet**

Renders end-user help/FAQ content (questions, clues, answers).

```

        this.answer = answer;
        this.clue = clue;
    }

    public int getId() { return id; }
    public String getQuestion() { return question; }
    public String getAnswer() { return answer; }
    public String getClue() { return clue; }
}

private final String url = "jdbc:mysql://localhost:3306/pahana_education?useSSL=false&serverTimezone=UTC";
private final String user = "root";
private final String pass = "";

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    List<HelpEntry> helpEntries = new ArrayList<>();

    try {
        Class.forName("com.mysql.cj.jdbc.Driver");

        try (Connection con = DriverManager.getConnection(url, user, pass);
             PreparedStatement stmt = con.prepareStatement("SELECT id, question, answer, clue FROM help");
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                helpEntries.add(new HelpEntry(
                    rs.getInt("id"),
                    rs.getString("question"),
                    rs.getString("answer"),
                    rs.getString("clue")
                ));
            }
        }
    } catch (Exception e) {
        throw new ServletException("Database error", e);
    }

    request.setAttribute("helpEntries", helpEntries);
    request.getRequestDispatcher("/adminHelp.jsp").forward(request, response);
}

```


- **AdminHelpServlet → /AdminHelpServlet**

Admin interface for CRUD on help entries so guidance stays current without code changes.

Typical request flow

1. User signs in via **LoginServlet**; role stored in session.
2. Admins maintain users with **Register/Edit/Delete/Update** servlets.
3. Catalog is curated via **AddItemServlet**; customers or staff **search** items and **add to cart**.
4. **GenerateBillServlet** finalizes the purchase and returns a printable summary.

Your Bill from Pahana Education Inbox x

 **krishanabeywardhana99@gmail.com**
to me ▾

Sat 9 Aug, 15:18 (3 days ago) ☆ 😊 ↶ ⋮

Pahana Education - Bill Summary

Bill ID: 11
Date: 2025-08-09

Item	Qty	Unit Price (LKR)	Subtotal (LKR)
Chair	2	8000.00	16000.00
Total:			16000.00

Thank you for shopping with Pahana Education!

5. Managers view trends via **TransactionReportServlet**.
6. Users consult **HelpServlet**, while admins update content with **AdminHelpServlet**.

Task C

Testing Plans

Representative functional test cases are outlined below.

Table 1: Representative functional test cases

ID	Feature	Preconditions	Steps	Expected Result
TC-001	Customer Login	Registered user; valid credentials	1) Open login page 2) Enter email/password 3) Submit	Redirect to customer dashboard; session created
TC-002	Add Item to Cart	Logged-in customer; item exists; stock > 0	1) Browse/search item 2) Set quantity 3) Click 'Add to Cart'	Cart shows item with correct quantity and subtotal
TC-003	Generate Bill	Cart has ≥ 1 item	1) Open cart 2) Click 'Generate Bill' 3) Confirm	Bill saved with items, totals, taxes; email sent to customer
TC-004	Print/Download Bill	Bill exists	1) Open Bill page 2) Click 'Print' or 'Download PDF'	Download succeeds; printable view renders
TC-005	View Reports (Admin)	Admin logged in; transactions exist	1) Navigate to Reports 2) Select date range 3) Generate	Report table/chart shows correct totals; export enabled
TC-006	Manage Users (Admin)	Admin logged in	1) Add user 2) Edit role 3) Deactivate user	CRUD operations persist; user status reflected in login

Task 4

Version Control & Collaboration (GitHub)

Link: <https://github.com/kristy999/panahaeducation>

- Use a trunk-based or GitFlow-light workflow: main, develop, and short-lived feature/* branches.
- Protect main with pull requests and required status checks.
- Configure GitHub Actions for CI (lint, tests) and artifact build (PDF reports).
- Use Issues and Projects for backlog and milestones; PR templates and code owners for reviews.

Conclusion

The Pahana Education Billing System (PEBS) achieves its central objective: giving an educational institute a single, trustworthy place to manage users, items, carts, and billing—while keeping administrative work auditable and simple for day-to-day staff. The system’s core flows are clear and tightly scoped: authenticate; register or manage users; curate items; build a cart; generate and print a bill; and review transaction trends. This end-to-end path is supported by consistent navigation, role-aware access, and screens designed to shorten the journey from intention to outcome. Together, these choices reduce operational friction, cut manual errors, and improve turnaround time for finance tasks.

From an engineering perspective, the design balances practicality and maintainability. A layered structure separates presentation, controllers, services, and data access—avoiding the “fat controller” anti-pattern and making business logic independently testable. The domain model is small but expressive (User, Customer, Item, Bill, BillItem, Help), and the relationships map directly to what the organization actually does: customers own bills; bills contain items; users play roles; and help content explains common tasks. On the web tier, discrete servlets encapsulate each use case (e.g., login, item search, add-to-cart, generate bill, reports, help), yielding small, purpose-built endpoints that are easy to reason about and secure behind role checks. This “one responsibility per servlet” approach is a good fit for incremental change and incident isolation.

Key decisions around patterns—Singleton for shared config/connection, Repository for persistence abstraction, Strategy for pricing/tax, and MVC for overall structure—are appropriate for a system of this size and will age well as features grow. They also make future refactors safer: repositories shield controllers from query details; strategies let you introduce discounts or term-based taxes without rewriting billing code; and the MVC boundary keeps view concerns separate from calculation rules. On the platform side, the presence of a help/FAQ subsystem is a quiet but strategic win: it lowers training costs, reduces repeat support questions, and lets non-developers keep guidance current.

Operationally, the system already demonstrates value. The dashboard gives managers instant visibility into recent transactions and user-role distribution, replacing ad-hoc spreadsheets with a consistent, reviewable source of truth. The printable bill view closes the loop with compliant, shareable records. Role-based user management, including customer profile linkage, keeps identity and financial data aligned. Item management is intentionally lightweight, which keeps counter operations fast while still enforcing price accuracy.

Quality assurance is baked in through representative test cases covering authentication, cart operations, billing, printing, reporting, and user administration. These scenarios are the right “minimum viable suite” to avoid regressions in the core money-flow and access-control paths. The recommended Git/GitHub workflow—protected main, short-lived feature branches, CI checks, and issues/projects—sets up the team for controlled change and predictable releases. Over time, expanding CI with unit tests for services, integration tests for repositories, and smoke tests for servlet endpoints will strengthen confidence further.

Security, reliability, and data integrity are treated with the seriousness they deserve. Authentication gates all sensitive operations, and the design naturally supports role checks at servlet entry points. Using POST for mutations (register, add item, cart updates, billing) and GET for idempotent reads (search, reports, help) encourages safe browser behavior and aligns with the PRG (Post/Redirect/Get) pattern to prevent duplicate submissions. Repository-level parameterized queries (or an ORM) will keep injection risks low; centralized validation in services reduces the chance of inconsistent rules across screens. Logging around bill generation and user CRUD will provide an audit trail helpful for dispute resolution and compliance.

No system is finished, only released, and PEBS has a healthy roadmap. Obvious extensions include discounts and tax rules via the existing Strategy seam; exports for CSV/PDF on reports; email copies of bills and password-reset flows; pagination and search in management tables; and soft-delete with restoration for safer administration. As data grows, you can add caching for item lookups and reports, archive old bills to cold storage, and tune indexes on frequently filtered fields. If concurrency increases (e.g.,

peak fee days), consider optimistic locking on inventory-like fields or a small queue around bill generation to keep totals consistent.

In terms of product fit, PEBS is already “useful on day one” for a small-to-medium institute, and its architecture makes it “evolvable on day two.” The team can ship changes confidently, stakeholders can verify outcomes via dashboard and printable artifacts, and new staff can self-serve with built-in help. Most importantly, the design keeps the focus on institutional outcomes—collect fees accurately, serve students quickly, and know what happened each day—rather than on framework ceremony.

In summary, PEBS delivers a coherent, role-aware, and maintainable solution to institutional billing. It turns a previously manual, error-prone process into a streamlined flow backed by clear screens, sensible domain boundaries, and testable business logic. With modest, well-targeted enhancements in security hardening, reporting exports, and CI coverage, the system is well positioned to support operational needs for the long term while remaining flexible enough to adapt to policy or curriculum changes. It is a strong foundation to build on—technically sound, operationally practical, and aligned with how real users work.

References

- Fowler, M. Patterns of Enterprise Application Architecture.
- Gamma, E., et al. Design Patterns: Elements of Reusable Object-Oriented Software.
- Official framework/library documentation used in the project.