PROCUREMENT PORTAL

A Comprehensive Solution for Public Organizations

PROJECT REPORT

Batoul Haidar | Web Engineering | 23/07/2024

Table of Contents

Executive Summary 2
Project Objectives 2
System Overview 2
System Features 3
    User Authentication and Authorization 3
    Purchaser Features 3
    Technician Features 3
    Administrator Features 3
Technical Implementation 3
    Frontend Development 3
    Backend Development 5
    Connection Pooling 6
    Database Integration 7
    Server: Apache Tomcat 7
Conclusion 7
Future Enhancements 7

Executive Summary

This report outlines the development and implementation of a Procurement Portal designed for a public organization. The portal facilitates efficient procurement processes by enabling purchasers to request products, technicians to handle procurement activities, and administrators to manage user access. Leveraging modern web technologies, the project ensures a secure, user-friendly, and efficient system to enhance procurement operations within the organization.

Project Objectives

The primary objectives of the Procurement Portal are as follows:

- Develop a secure and efficient portal for managing procurement processes.
- Implement role-based access control for purchasers, technicians, and administrators.
- Provide a streamlined interface for requesting, approving, and managing procurement proposals.
- Ensure robust backend capabilities to handle requests, approvals, and user management.
- Maintain a comprehensive database to store all procurement-related data.

System Overview

The Procurement Portal is designed with three main user roles, each with specific functionalities:

- **Purchaser**: Requests products and approves or rejects procurement proposals.
- **Technician**: Manages new requests, approves requests, and submits procurement proposals.
- **Administrator**: Manages user accounts and ensures system integrity

System Features

USER AUTHENTICATION AND AUTHORIZATION

- **Login Requirement**: All users must log in to access the portal.
- **Session Management**: Secure and persistent sessions ensure user authenticity and security.

PURCHASER FEATURES

- **Submit Product Requests**: Purchasers can request products through the portal.
- **Manage Requests**: View and manage submitted requests.
- **Approve/Reject Proposals**: Approve or reject procurement proposals with reasons for rejection.

TECHNICIAN FEATURES

- **View Requests**: Access and manage new product requests.
- **Approve Requests**: Approve requests and create procurement proposals.
- **Review Rejections**: Review reasons for rejection and submit revised proposals.

ADMINISTRATOR FEATURES

- **User Management**: Create and manage user accounts.
- **Role Assignment**: Assign roles to users (Purchaser, Technician, Administrator).
- **System Monitoring**: Monitor system usage and ensure data security.

Technical Implementation

FRONTEND DEVELOPMENT

The frontend of the Procurement Portal was developed using a combination of HTML5, CSS, and JavaScript, ensuring a structured, maintainable, and dynamic user experience.

HTML5

HTML5 was employed to structure the content and layout of the web pages. Each user role (Admin, Purchaser, Technician) has its dedicated HTML files within their respective directories. This clear separation of concerns simplifies maintenance and updates.

CSS

A single, centralized CSS file (`style.css`) was used to maintain a consistent theme across the entire application. This centralized approach allows for easy customization and ensures that any design changes are applied uniformly. By leveraging modern CSS techniques, the design is both professional and responsive, adapting seamlessly to various screen sizes and devices.

JavaScript

JavaScript was used to add interactivity and client-side validation, enhancing the user experience with dynamic features such as form validations, data fetching, and interactive elements. The JavaScript files are organized by user role and include a global directory for common functionalities like login, logout, and notifications. This structure promotes code reuse and maintainability, ensuring that shared functionalities are easily accessible across different user roles.

The global JavaScript directory handles critical functions:

- **Login and Logout**: Ensures consistent authentication flows across all user interfaces.
- **Notifications**: Manages system-wide alerts and notifications, providing real-time feedback to users about their actions and system status.

Frontend Architecture

The frontend architecture is organized as follows:

- Site Root
    - CSS
    - HTML
        - Global
        - Admin
        - Technician
        - Purchaser
    - JavaScript
        - Global
        - Admin
        - Technician
        - Purchaser

BACKEND DEVELOPMENT

The backend logic was implemented using Java Servlets, which provided a robust and scalable solution for handling server-side operations. The backend architecture adheres to the Model-Controller (MC) pattern and utilizes the Data Access Object (DAO) pattern for database interactions.

Java Servlets (Controller Layer)

Java Servlets were employed as controllers to manage HTTP requests and responses. They handle user inputs, process business logic, and generate appropriate responses for the client-side. Each servlet corresponds to specific functionalities within the application, ensuring a modular and maintainable codebase.

DAO Pattern

The DAO pattern abstracts and encapsulates all access to the data source, providing a clear separation between the business logic and data access logic. The DAO interface defines the standard operations to be performed on the model objects, while the DAO implementation classes (DAOImp) handle the actual database interactions.

Model Layer

The model layer represents the data objects and business logic. Each model class corresponds to a table in the database and contains fields that map to the columns in the tables.

Additional Folders

Util :

Utility classes that provide common functionalities used across the application, such as helper methods for string manipulation.

Security :

Contains security-related classes, such as handling sessions.

Backend Architecture

The backend architecture is organized as follows:

- procurement_portal_back_end
  - Controller
  - Data
    - DAO
    - Model
  - Security
  - Utils

CONNECTION POOLING

To improve the performance and scalability of the database interactions, connection pooling is implemented. Connection pooling reduces the overhead of establishing a new database connection for every request by reusing a pool of established connections.

Benefits of Connection Pooling

1. **Performance Improvement**: Reusing existing connections reduces the time required to establish a new connection, making the application more responsive.
2. **Resource Management**: Limits the number of concurrent connections to the database, preventing resource exhaustion.
3. **Scalability**: Allows the application to handle a higher number of concurrent requests by efficiently managing database connections.

Implementation of Connection Pooling

Connection pooling is configured in the `context.xml` file of the Apache Tomcat server. This setup includes defining a resource for the database with parameters such as maximum total connections, idle connections, and connection timeout settings.

Centralized DBManager Class

A centralized `DBManager` class is used to manage database connections. This class accesses the DataSource configured in `context.xml` and provides connections to the DAO classes. By using a centralized approach, the connection management becomes streamlined and consistent across the application.

Using the DBManager in DAO Classes

DAO classes utilize the `DBManager` to obtain database connections from the connection pool. This allows the DAO classes to focus on data operations without worrying about connection management, promoting clean and maintainable code.

By configuring connection pooling through the `context.xml` file and managing connections using a centralized `DBManager` class, the application can efficiently handle database interactions, improving overall performance and scalability. This setup ensures that DAO classes can easily obtain and use database connections from the pool, maintaining a clean and maintainable codebase.

DATABASE INTEGRATION

A MySQL database was established to store and manage all procurement-related data:

- **Database Design**: The database schema was designed to store user data, procurement requests, proposals, and approval statuses. This included defining tables, relationships, and constraints to ensure data integrity and consistency.
- **Data Management**: CRUD (Create, Read, Update, Delete) operations were implemented to handle various data management tasks. These operations allowed for efficient data retrieval, insertion, updating, and deletion as required by the application.

SERVER: APACHE TOMCAT

- **Deployment**: The system is deployed on an Apache Tomcat server, ensuring a stable and reliable environment for running the web application.
- **HTTPS Configuration**: HTTPS was enabled to ensure secure communication between clients and the server. This involved obtaining and installing a security certificate, configuring the server to use HTTPS, and ensuring that all data transmitted was encrypted.

Conclusion

The Procurement Portal successfully meets the objectives of providing a secure, efficient, and user-friendly system for managing procurement processes within a public organization. The role-based access control ensures that each user can perform their specific tasks, enhancing overall productivity and transparency.

Future Enhancements

- **Advanced Analytics**: Integrate analytics to provide insights into procurement activities and trends.
- **Automated Notifications**: Implement email notifications for key actions and updates.
- **AI Integration**: Utilize artificial intelligence to predict procurement needs and optimize inventory management.