**Legislative System**

**Project Overview**

The **Legislation Process Management System** is a web-based application for managing legislative bills within the Federal Parliament of Canada. The system allows **Members of Parliament** to create and propose bills, **Reviewers** to suggest amendments, and **Administrators** to manage users and oversee the voting process.

**Table of Contents**

* Architecture
* Technologies Used
* Database Schema
* Modules
* User Roles and Permissions
* System Workflow
* API Endpoints
* Deployment
* Error Handling
* Testing
* Future Enhancements

**Architecture**

**1. Architecture Pattern**

* The project follows the **Model-View-Controller (MVC)** pattern:
  + **Model**: Manages the core data structure, interacts with the database using PDO, and represents business logic (e.g., Vote.php).
  + **View**: Provides the front-end interface for users, allowing them to interact with the system (e.g., views/bills/create.php).
  + **Controller**: Contains logic for handling user requests and linking models with views (e.g., VoteController.php).

**2. Technology Stack**

* **PHP**: Backend logic and business processes.
* **MySQL**: Database for storing user information, bills, votes, and amendments.
* **PDO (PHP Data Objects)**: For secure database connectivity and query handling.
* **Podman/Docker Compose**: For containerized deployment of the application and MySQL database.

**Technologies Used**

* **PHP**: Server-side scripting for handling backend logic.
* **MySQL**: Relational database to store legislative data and user information.
* **Tailwind CSS**: A modern utility-first CSS framework for styling.
* **JavaScript**: Used for client-side interactions.
* **Composer**: Dependency management for PHP.

**Prerequisites**

* **PHP** 8.2 or above.
* **MySQL Database**.
* **Apache Web Server** (XAMPP or similar).
* **Composer** for dependency management.

**Database Schema**

**1. Users Table**

Stores information about each user in the system, including MPs, Reviewers, and Administrators.

CREATE TABLE users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

role ENUM('MP', 'Reviewer', 'Admin') NOT NULL,

email VARCHAR(100) NOT NULL

);

**2. Bills Table**

Contains information about each bill, including its status and author.

CREATE TABLE bills (

bill\_id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

description TEXT,

author\_id INT,

status ENUM('Draft', 'Under Review', 'Amended', 'Voting', 'Passed', 'Rejected') NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (author\_id) REFERENCES users(user\_id)

);

**3. Votes Table**

Records each vote cast by MPs on bills in the voting session.

CREATE TABLE votes (

vote\_id INT AUTO\_INCREMENT PRIMARY KEY,

bill\_id INT,

voter\_id INT,

vote\_value ENUM('For', 'Against', 'Abstain') NOT NULL,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (bill\_id) REFERENCES bills(bill\_id),

FOREIGN KEY (voter\_id) REFERENCES users(user\_id)

);

**4. Amendments Table**

Stores suggested amendments for each bill during the review process.

CREATE TABLE amendments (

amendment\_id INT AUTO\_INCREMENT PRIMARY KEY,

bill\_id INT,

suggested\_changes TEXT NOT NULL,

reviewer\_id INT,

comments TEXT,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (bill\_id) REFERENCES bills(bill\_id),

FOREIGN KEY (reviewer\_id) REFERENCES users(user\_id)

);

**Modules**

**1. User Management**

* Handles user registration, login, and role-based access control.
* Provides separate permissions for MPs, Reviewers, and Admins.

**2. Bill Management**

* Enables MPs to create, edit, and submit bills.
* Allows status updates as bills move through the lifecycle (Draft → Under Review → Amended/Voting → Passed/Rejected).

**3. Voting System**

* Enables MPs to cast votes on bills during an active voting session.
* Calculates voting results and updates the bill’s status based on the majority of votes.

**4. Amendment Management**

* Allows reviewers to suggest amendments and provide feedback during the review process.

**User Roles and Permissions**

| **Role** | **Permissions** |
| --- | --- |
| **MP** | Create/edit bills, cast votes, view bill status and voting results. |
| **Reviewer** | Suggest amendments, comment on bills, view bills and voting results. |
| **Admin** | Manage users, initiate/close voting sessions, approve/reject bills based on voting results. |

**System Workflow**

1. **Bill Creation**:
   * MP drafts a bill (status: **Draft**).
2. **Bill Review**:
   * Reviewers provide feedback and suggest amendments (status changes to **Under Review**).
3. **Voting**:
   * Administrator initiates voting (status changes to **Voting**).
   * MPs cast their votes.
4. **Closing Voting**:
   * Administrator closes voting, results are calculated, and the bill status updates to either **Passed** or **Rejected**.

**API Endpoints**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /login | POST | Authenticates user credentials |
| /bills/create | POST | Creates a new bill (MP role only) |
| /bills/:id/vote | POST | Allows MP to cast a vote on a bill |
| /bills/:id/closeVoting | POST | Closes voting session (Admin role only) |
| /amendments/create | POST | Suggests an amendment for a bill (Reviewer only) |
| /bills/:id/results | GET | Retrieves voting results for a bill |

**Deployment**

1. **Clone the Repository**:

bash

Copy code

git clone <repository-url>

cd legislation-process-management-system

1. **Set Up Podman/Docker Compose**:
   * Add configuration for docker-compose.yml to define services for app (PHP) and mysql.
2. **Run Containers**:
   * Start the application with Podman:

bash

Copy code

podman-compose up --build

1. **Access the Application**:
   * Visit http://localhost:8080 in a web browser.

**Error Handling**

1. **Database Connection Errors**: Catch PDO exceptions and display custom error messages.
2. **User Authentication**: Handle invalid login attempts and unauthorized access with appropriate error responses.
3. **Input Validation**: Validate user inputs, especially for bill creation, voting, and amendments.

**Testing**

**1. Unit Tests**

* Test individual modules (User, Bill, Vote) to ensure that each component functions as expected.

**2. Integration Tests**

* Test the workflow from bill creation to voting results, ensuring each stage transitions correctly.

**3. Error Handling Tests**

* Test incorrect inputs and unauthorized access attempts to verify proper handling and error messages.

**4. Sample Test Cases**

| **Test Case** | **Expected Result** |
| --- | --- |
| User logs in with invalid password | Display error message |
| MP creates a new bill | Bill is added to database |
| Reviewer suggests an amendment | Amendment is linked to the bill |
| Administrator closes voting | Results are calculated and bill status is updated |

**Future Enhancements**

1. **Email Notifications**: Notify users when a voting session starts or closes.
2. **Role-Based Dashboards**: Customized dashboards for each role with relevant actions.
3. **Audit Logs**: Log significant actions for tracking and monitoring.

**Acknowledgments**

* This project was developed as part of a technical assignment at Humber College, with guidance from instructors and team collaboration.

**Screenshots**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated