Opis projektu "Library App"

1. Wprowadzenie

Celem projektu "Library App" jest stworzenie prostego, lecz funkcjonalnego systemu zarządzania biblioteką, obejmującego rejestrację i edycję zasobów (książek, autorów), obsługę kont czytelników oraz realizację procesu wypożyczeń i zwrotów z naliczaniem ewentualnych kar. Aplikacja została zbudowana w architekturze MVC (Model-View-Controller), co zapewnia przejrzysty podział odpowiedzialności, ułatwia utrzymanie kodu oraz jego późniejszą rozbudowę. Całość jest przygotowana do pracy w środowisku kontenerowym (Docker + Nginx + PHP + MySQL), co znacznie upraszcza wdrożenie.

2. Założenia funkcjonalne

1. Zarządzanie zasobami biblioteki: Tworzenie, edytowanie, usuwanie i przeglądanie rekordów książek i autorów. Każda książka posiada powiązanie z jednym autorem (relacja 1-N: jeden autor może mieć wiele książek).

2. Zarządzanie czytelnikami (members): Rejestracja nowych czytelników. Edycja danych czytelników oraz możliwość ich usunięcia. Walidacja podstawowych pól (np. imię, nazwisko, email) w kontrolerze.

3. Obsługa wypożyczeń (loans): Tworzenie wypożyczeń: czytelnik wybiera książkę, ustalana jest data wypożyczenia i przewidywana data zwrotu. Rejestracja zwrotu książki; w momencie zwrotu system oblicza ewentualne opóźnienie i nalicza karę (np. stała stawka za każdy dzień opóźnienia). Wyświetlanie historii wypożyczeń danego czytelnika oraz generowanie prostego raportu: obecnie wypożyczone książki i suma kar do zapłaty.

4. Warstwa prezentacji (interfejs webowy)

Widoki HTML/CSS umożliwiające wygodne przeglądanie list, formularze dodawania/edycji oraz strony szczegółowe. Połączone z kontrolerami za pomocą plików szablonów (views/).

3. Architektura i wzorce projektowe

Projekt został zrealizowany w oparciu o klasyczny wzorzec MVC:

- Model (katalog app/models/):

Każda encja (Book, Author, Member, Loan) jest reprezentowana przez osobną klasę modelu (Book.php, Author.php, Member.php, Loan.php). Modele zawierają metody CRUD: getAll(), getById(id), create(data), update(id, data), delete(id). Odpowiadają za bezpośrednią interakcję z bazą danych (zapytania SQL przygotowane w metodach).

- View (katalog app/views/): Dla każdej encji istnieje osobny podkatalog z widokami: books/, authors/, members/, loans. Widoki korzystają z prostych szablonów PHP, wstawiają dane z kontrolerów.

-Controller (katalog app/controllers/): Dla każdej encji (Book, Author, Member, Loan) istnieje dedykowany kontroler: BooksController.php, AuthorsController.php, MembersController.php, LoansController.php.

- Kontrolery realizują logikę biznesową: Pobierają parametry z URL lub formularza ($_GET / $_POST). Wywołują odpowiednie metody modelu. Przekazują dane do widoku (poprzez wczytanie pliku PHP z podaniem zmiennych). Obsługują przekierowania po operacjach (np. po dodaniu lub edycji). W kontrolerach zaimplementowano prostą walidację danych (sprawdzanie czy pola nie są puste; w razie błędów zwraca widok z informacją użytkownika).

- Front Controller / Routing: Plik public/index.php pełni funkcję jedynego punktu wejścia. Ładuje autoloader oraz plik routes.php, w którym definiowane są ścieżki URL i kojarzone metody kontrolerów. Dzięki temu każde żądanie trafia do odpowiedniej akcji kontrolera, co w pełni odpowiada wzorcowi Front Controller.

- Zasady SOLID

1. Single Responsibility Principle (SRP): Każda klasa ma jedną odpowiedzialność: model zajmuje się operacjami na danych, kontroler realizuje logikę żądań, widok odpowiada za prezentację, a plik routes.php – za kierowanie URL.

2. Open/Closed Principle (OCP): Kod jest otwarty na rozszerzenia (można dodać nowe metody w modelach lub akcje w kontrolerach) bez modyfikowania istniejących elementów.

3. Liskov Substitution Principle (LSP): W projekcie nie zastosowano hierarchii klas ani dziedziczenia między modelami, zatem nie występuje problem podstawiania klas.

4. Interface Segregation Principle (ISP): Projekt nie zawiera interfejsów w tradycyjnym sensie; każda klasa modelu udostępnia tylko te metody, które są dla niej niezbędne.

5. Dependency Inversion Principle (DIP): W tym prostym projekcie zależności są wstrzykiwane ręcznie (kontroler tworzy model), co nie jest pełnym DIP. Aby w pełni zastosować DIP, można by dodać warstwę Repository lub Service Layer.

4. Struktura bazy danych

W katalogu głównym znajduje się skrypt init.sql, definiujący tabele: authors, book_author, books, loans, members. Relacje między tabelami zostały zdefiniowane przy pomocy kluczy obcych (FOREIGN KEY), co zapewnia spójność danych. Skrypt tworzy bazy, tabele oraz w razie potrzeby indeksy na kolumnach używanych w zapytaniach (np. isbn, email).

5. Opis poszczególnych modułów

1. Moduł zarządzania autorami (Authors)

Model Author.php: Metody: getAll(), getById(id), create(data), update(id, data), delete(id). Atrybuty odzwierciedlają strukturę tabeli authors.

Kontroler AuthorsController.php: index(): pobiera listę autorów i ładuje widok views/authors/index.php, create(): wyświetla formularz tworzenia, store(): waliduje dane, tworzy rekord w bazie i przekierowuje do listy, edit(id): pobiera dane wybranego autora, wyświetla formularz edycji, update(id): waliduje i aktualizuje dane, destroy(id): usuwa autora (jeśli nie ma powiązanych książek, inaczej zgłasza błąd).

2. Moduł zarządzania książkami (Books):

Model Book.php: Metody CRUD + dodatkowe: decreaseCopies(id), increaseCopies(id).

Kontroler BooksController.php: Działa analogicznie jak w module autorów, z uwzględnieniem wyboru autora z listy (lista autorów przekazywana do widoku).

3. Moduł zarządzania czytelnikami (Members):

Model Member.php: Podstawowe metody CRUD.

Kontroler MembersController.php: Zapewnia rejestrację (formularz), edycję, usuwanie czytelników. Walidacja: sprawdzanie poprawności emaila (proste sprawdzenie formatu), uniknięcie duplikatów.

4. Moduł wypożyczeń (Loans)

- Model Loan.php: createLoan(bookId, memberId): sprawdza dostępność książki (kopie > 0), zmniejsza liczbę dostępnych egzemplarzy (decreaseCopies() w Book), wpisuje rekord z datą wypożyczenia i planowanym terminem (np. 14 dni od dnia wypożyczenia), returnBook(loanId): ustawia return_date na dzisiaj, wywołuje calculateFine(), przywraca kopię książki (increaseCopies()), calculateFine(): oblicza różnicę między return_date a due_date, mnoży przez stałą stawkę (np. 1 PLN za dzień), getHistoryByMember(memberId): pobiera wszystkie wypożyczenia danego czytelnika (z kolumnami: tytuł książki, data wypożyczenia, data zwrotu, kara).

- Kontroler LoansController.php: index(): wyświetla listę wszystkich wypożyczeń (z możliwością filtrowania wg statusu: aktywne, zwrócone), create(): Pobiera listę dostępnych książek (takich, których copies_available > 0). Pobiera listę wszystkich czytelników. Wyświetla formularz wyboru książki i czytelnika, store(): Waliduje wybrane book_id i member_id. Wywołuje Loan::createLoan($bookId, $memberId). Przekierowuje do listy wypożyczeni, return($id): rejestruje zwrot książki (wywołuje Loan::returnBook($id)) i przekierowuje do historii wypożyczeń czytelnika, history($memberId): wyświetla historię wypożyczeń danego czytelnika (lista wypożyczeń z datami i karami).