Wyjaśnienia kodu:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Kod ten definiuje prosty kontroler API dla książek w ASP.NET Core MVC. Kontroler ten używa serwisu BookService do pobierania listy książek i zwraca ją w odpowiedzi na żądanie HTTP GET pod adresem /book/getBooks.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

 [ApiController] - Atrybut informujący, że klasa jest kontrolerem API.

 [Route("[controller]")] - Definiowanie domyślnej trasy dla kontrolera.

 public class BookLoanController : ControllerBase - Deklaracja klasy kontrolera dziedziczącej po ControllerBase.

 private readonly BookLoanService \_bookLoanService; - Prywatne pole do przechowywania instancji serwisu.

 public BookLoanController(BookLoanService bookLoanService) - Konstruktor z wstrzykiwaniem zależności (Dependency Injection).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

 [HttpPut("returnBook/{bookLoanId}")] - Deklaracja, że metoda obsługuje żądania HTTP PUT pod ścieżką returnBook/{bookLoanId}.

 public async Task<IActionResult> ReturnBook(int bookLoanId) - Definicja metody, która przyjmuje bookLoanId jako parametr.

 Tworzenie nowego obiektu TSORDERS i jego właściwości UNH, ustawianie parametrów komunikatu.

 Tworzenie Loop\_APR\_ORDERS i przypisywanie bookLoanId.

 Asynchroniczne wywołanie metody ReturnBook na serwisie \_bookLoanService z utworzonym ordersMessage.

 Zwracanie HTTP 200 (OK) jako odpowiedź.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

 [HttpPost("loanBook/{bookId}/{clientId}")] - Deklaracja, że metoda obsługuje żądania HTTP POST pod ścieżką loanBook/{bookId}/{clientId}.

 public IActionResult LoanBook(int bookId, int clientId) - Definicja metody, która przyjmuje bookId i clientId jako parametry.

 Tworzenie nowego obiektu TSORDERS i jego właściwości UNH, ustawianie parametrów komunikatu.

 Tworzenie Loop\_NAD\_ORDERS i przypisywanie clientId.

 Tworzenie Loop\_LIN\_ORDERS i przypisywanie bookId.

 Wywołanie metody Insert na \_bookLoanService z utworzonym ordersMessage.

 Zwracanie HTTP 200 (OK) jako odpowiedź.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

 [HttpPost("loanBook/{bookId}/{clientId}")] - Deklaracja, że metoda obsługuje żądania HTTP POST pod ścieżką loanBook/{bookId}/{clientId}.

 public IActionResult LoanBook(int bookId, int clientId) - Definicja metody, która przyjmuje bookId i clientId jako parametry.

 Tworzenie nowego obiektu TSORDERS i jego właściwości UNH, ustawianie parametrów komunikatu.

 Tworzenie Loop\_NAD\_ORDERS i przypisywanie clientId.

 Tworzenie Loop\_LIN\_ORDERS i przypisywanie bookId.

 Wywołanie metody Insert na \_bookLoanService z utworzonym ordersMessage.

 Zwracanie HTTP 200 (OK) jako odpowiedź.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Kontroler PaymentController obsługuje operacje związane z płatnościami użytkowników. Składa się z:

* Definicji przestrzeni nazw i kontrolera.
* Prywatnego pola PaymentService oraz konstruktora do jego inicjalizacji przy użyciu Dependency Injection.
* Metody GetPayments, która obsługuje żądania GET, wywołuje metodę serwisu GetForUser z userId jako parametrem i zwraca wynik jako odpowiedź HTTP 200 (OK).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Kontroler UserController obsługuje operacje związane z uwierzytelnianiem użytkowników. Składa się z:

* Definicji przestrzeni nazw i kontrolera.
* Prywatnego pola UserService oraz konstruktora do jego inicjalizacji przy użyciu Dependency Injection.
* Metody Authenticate, która obsługuje żądania POST, wywołuje metodę serwisu Authenticate z username i password jako parametrami i zwraca wynik jako odpowiedź HTTP 200 (OK).

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Klasa BiblioConnectDbContext jest kontekstem bazy danych w aplikacji ASP.NET Core korzystającej z Entity Framework Core. Obejmuje ona zestawy encji (DbSet), które reprezentują tabele w bazie danych, takie jak Books, Clients, BookLoans, Payments i Users. Konstruktor klasy przyjmuje opcje konfiguracji bazy danych i przekazuje je do konstruktora bazowego DbContext, co umożliwia łatwe konfigurowanie i zarządzanie połączeniem z bazą danych.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

Klasa MyJobs obsługuje zadania w tle w aplikacji ASP.NET Core przy użyciu Hangfire. Składa się z:

* Definicji przestrzeni nazw i klasy.
* Prywatnego pola EmailService oraz konstruktora do jego inicjalizacji przy użyciu Dependency Injection.
* Metody ReminderEmail, która wywołuje metodę SendReminder3DaysEmails z serwisu EmailService w celu wysłania przypomnień e-mail.

Hangfire umożliwia harmonogramowanie i zarządzanie zadaniami w tle, co jest przydatne do operacji takich jak wysyłanie e-maili przypominających. Klasa MyJobs pokazuje, jak można zdefiniować zadanie i wykorzystać wstrzykiwanie zależności do uzyskania dostępu do serwisu e-mail.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Te dwie klasy GetBookLoanForClientResponse i GetPaymentForClientResponse są używane do reprezentowania odpowiedzi z serwisów API zwracających dane o wypożyczonych książkach i płatnościach dla klientów. Każda klasa zawiera odpowiednie właściwości, które przechowują potrzebne informacje do odpowiedniego wyświetlenia danych klientowi.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie Obraz zawierający tekst, zrzut ekranu, wyświetlacz, oprogramowanie

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Te klasy encji definiują strukturę danych dla różnych obiektów w aplikacji:

* Book - Informacje o książkach.
* BookLoan - Informacje o wypożyczeniach książek.
* Client - Informacje o klientach.
* Payment - Informacje o płatnościach.
* User - Informacje o użytkownikach.

Każda klasa zawiera właściwości odpowiadające kolumnom w tabeli bazy danych, a atrybut [Key] wskazuje klucz główny każdej tabeli.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Serwis BookLoanService zarządza operacjami wypożyczeń książek, w tym pobieraniem wypożyczeń dla klienta, dodawaniem nowych wypożyczeń oraz obsługą zwrotów książek. Wszystkie operacje są realizowane poprzez kontekst bazy danych BiblioConnectDbContext, a operacje asynchroniczne wykorzystują serwis e-mail EmailService do wysyłania powiadomień.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

BookService umożliwia pobieranie wszystkich książek z bazy danych przy użyciu Entity Framework Core. Dzięki zastosowaniu Dependency Injection serwis może łatwo korzystać z kontekstu bazy danych BiblioConnectDbContext do wykonywania operacji CRUD.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

* Metoda SendReminder3DaysEmails wyszukuje wypożyczenia książek, których termin zwrotu mija za 3 dni.
* Pobiera dane klienta i książki dla każdego wypożyczenia.
* Jeśli klient i książka są znalezione, a adres e-mail klienta nie jest pusty, wysyła e-mail z przypomnieniem.Obraz zawierający tekst, zrzut ekranu, oprogramowanie

  Opis wygenerowany automatycznie

 Metoda PrepareInvoiceEmail tworzy fakturę za przetrzymywanie książki i wysyła ją do klienta.

 Pobiera dane klienta na podstawie wypożyczenia.

 Jeśli klient jest znaleziony i jego adres e-mail nie jest pusty, oblicza kwotę faktury i wysyła e-mail z fakturą.

 Dodaje nową płatność do bazy danych i zapisuje zmiany.

* Obraz zawierający tekst, zrzut ekranu, Czcionka

  Opis wygenerowany automatycznie  
  Metoda SendInvoiceEmail tworzy fakturę w formacie PDF i wysyła ją do klienta za pomocą usługi SendGrid.
* Tworzy dokument PDF zawierający fakturę.
* Inicjalizuje klienta SendGrid za pomocą klucza API.
* Tworzy wiadomość e-mail z podanym adresem odbiorcy, tematem i treścią, oraz dołącza fakturę PDF jako załącznik.
* Wysyła wiadomość e-mail asynchronicznie.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

 Metoda SendEmail wysyła e-mail przy użyciu usługi SendGrid.

 Inicjalizuje klienta SendGrid za pomocą klucza API.

 Tworzy wiadomość e-mail z podanym adresem odbiorcy, tematem i treścią.

 Wysyła wiadomość e-mail asynchronicznie.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Klasa PaymentService zarządza operacjami związanymi z płatnościami w aplikacji. Składa się z:

* Definicji przestrzeni nazw i klasy.
* Prywatnego pola \_dbContext oraz konstruktora do jego inicjalizacji przy użyciu Dependency Injection.
* Metody GetForUser, która pobiera wszystkie płatności dla danego użytkownika na podstawie jego ID, a następnie dla każdej płatności pobiera szczegóły wypożyczenia i książki, po czym zwraca listę odpowiedzi zawierających te dane.

Implementacja serwisu PaymentService umożliwia łatwe zarządzanie płatnościami użytkowników, pobieranie powiązanych informacji o książkach i wypożyczeniach oraz zwracanie tych informacji jako listy obiektów GetPaymentForClientResponse.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Klasa UserService zarządza operacjami związanymi z uwierzytelnianiem użytkowników w aplikacji. Składa się z:

* Definicji przestrzeni nazw i klasy.
* Prywatnego pola \_dbContext oraz konstruktora do jego inicjalizacji przy użyciu Dependency Injection.
* Metody Authenticate, która uwierzytelnia użytkownika na podstawie nazwy użytkownika i hasła, a następnie zwraca identyfikator klienta powiązanego z tym użytkownikiem, jeśli uwierzytelnianie zakończy się pomyślnie. Jeśli uwierzytelnianie nie powiedzie się, zwraca 0.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Konfiguracja aplikacji ASP.NET Core obejmuje rejestrację i konfigurację wielu usług, w tym:

* **Entity Framework Core** do pracy z bazą danych SQL Server.
* **Hangfire** do zarządzania zadaniami w tle.
* **Swagger** do dokumentacji API.
* **CORS** do zezwalania na żądania z określonego źródła.
* **Lokalizacja** do ustawiania domyślnej kultury aplikacji.

Angular:

"$schema": "./node\_modules/@angular/cli/lib/config/schema.json",

"version": 1,

"newProjectRoot": "projects",

"projects": {

"biblio-connect": {

"projectType": "application",

"schematics": {},

"root": "",

"sourceRoot": "src",

"prefix": "app",

"architect": {

"build": {

"builder": "@angular-devkit/build-angular:application",

"options": {

"outputPath": "dist/biblio-connect",

"index": "src/index.html",

"browser": "src/main.ts",

"polyfills": [

"zone.js"

],

"tsConfig": "tsconfig.app.json",

"assets": [

"src/favicon.ico",

"src/assets"

],

"styles": [

"src/styles.css"

],

"scripts": []

},

"configurations": {

"production": {

"budgets": [

{

"type": "initial",

"maximumWarning": "500kb",

"maximumError": "1mb"

},

{

"type": "anyComponentStyle",

"maximumWarning": "2kb",

"maximumError": "4kb"

}

],

"outputHashing": "all"

},

"development": {

"optimization": false,

"extractLicenses": false,

"sourceMap": true

}

},

"defaultConfiguration": "production"

},

"serve": {

"builder": "@angular-devkit/build-angular:dev-server",

"configurations": {

"production": {

"buildTarget": "biblio-connect:build:production"

},

"development": {

"buildTarget": "biblio-connect:build:development"

}

},

"defaultConfiguration": "development"

},

"extract-i18n": {

"builder": "@angular-devkit/build-angular:extract-i18n",

"options": {

"buildTarget": "biblio-connect:build"

}

},

"test": {

"builder": "@angular-devkit/build-angular:karma",

"options": {

"polyfills": [

"zone.js",

"zone.js/testing"

],

"tsConfig": "tsconfig.spec.json",

"assets": [

"src/favicon.ico",

"src/assets"

],

"styles": [

"src/styles.css"

],

"scripts": []

}

}

}

}

}

}

Plik angular.json definiuje konfigurację projektu Angular biblio-connect. Zawiera informacje o strukturze katalogów, głównych plikach projektu, stylach, zasobach, a także szczegółowe ustawienia dla różnych zadań, takich jak budowanie, uruchamianie serwera deweloperskiego, wyodrębnianie tłumaczeń i uruchamianie testów. Konfiguracja zawiera również różne opcje dla środowisk produkcyjnych i deweloperskich.