Advanced Programming Skills

Web API documentation

This document shows how to set up Web API in ASP.NET Core Framework and connect to the PostgreSQL database using Dapper ORM.

Project Requirements:

The software along with their versions used in this document are as follows:

- Microsoft Visual Studio 2017
- PostgreSQL 9.6
- pgAdmin 4.1
- Dapper 1.50.2
- Npgsql 3.2.2
- Postman
- Swashbuckle 1.0.0

These software are explained in detail as below:

Microsoft Visual Studio:

Microsoft Visual Studio is an IDE developed by Microsoft. Several computer programs such as windows console application, web applications, mobile applications, etc. can be developed using Microsoft Visual Studio IDE. It supports various programming languages such as C++, VB.NET, C#, F#, etc. Visual Studio 2017 can be downloaded from the link: https://www.visualstudio.com/downloads/

PostgreSQL:

PostgreSQL is a relational database which stores a collection of data. This relational database also stores relationships between the data which is stored in it. PostgreSQL database can be downloaded from the following link: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

pgAdmin:

pgAdmin is an open source GUI administration tool and powerful query tool for PostgreSQL database. pgAdmin can be downloaded from the following link: https://www.pgadmin.org/download/

Dapper:

Dapper is an open source object-relational mapping (ORM) which is provided in Microsoft .NET platform. It maps an object-oriented code to a relational database. It is compatible with any database that helps in implementing ADO.NET data providers for databases such as PostgreSQL,

MySQL, Oracle, SQL Server, etc. The steps for downloading Dapper in Visual Studio will be explained below in Step 2.

Npgsql:

Npgsql is one of the open-source ADO.NET provider that connects and provides access to PostgreSQL database. The steps for downloading Npgsql in Visual Studio will be explained below in Step 2.

Postman:

Postman is an HTTP tool which can be used to test the RESTful Web API. Postman can be downloaded from the following link: https://www.getpostman.com/

Swashbuckle:

Swagger documents can be generated with the help of an open source project called Swashbuckle. RESTful APIs can be designed, built, documented and consumed by using an open source framework called Swagger. The steps for downloading Swashbuckle in Visual Studio will be explained below in Step 2.

Step 1: Database setup

Create a new database in PostgreSQL "LibraryDB" by following the steps in pgAdmin tool:

Open pgAdmin > Servers > Click on the Server in left pane > Right Click > Click on Connect. While connecting to the database, a pop up may come to ask for the password to connect to the PostgreSQL database.

Click on Server in left pane > Right Click > Click on Create > Database.

A pop up comes and asks for the name of database to be created. Give the name of the database as "LibraryDB" and click on save.

Create the tables and indexes using the following SQL:

```
create table libraries
(
	libraryid serial not null constraint "PK_Library" primary key,
	libraryname text,
	address text
);

create index "IX_Library_LibraryID" on libraries(libraryid);

create table patrons
(
	patronid serial not null constraint "PK_Patron" primary key,
	fname text,
	lname text,
```

```
        email text
);

create index "IX_Patron_PatronId" on patrons (patronid);

create table books
(
        bookid serial not null constraint "PK_Books" primary key,
        publisheddate timestamp not null,
        title text,
        libraryid integer constraint FK_Books_Library_LibraryId references libraries on delete
            cascade on update set null,
        isavailable boolean,
        patronid integer constraint FK_Books_Patron_PatronId references patrons on delete
            cascade on update set null,
        duedate timestamp
);

create table authors
(
        authorid serial not null constraint "PK_Authors" primary key,
        fname text,
        lname text not null
);

create table bookauthor
(
        bookid integer not null constraint FK_BookAuthor_Books_BookId references books
             on delete cascade,
        authorid integer not null constraint FK_BookAuthor_Authors_AuthorId references authors
             on delete cascade,
        constraint "PK_BookAuthor" primary key (bookid, authorid)
);

create index "IX_BookAuthor_AuthorId" on bookauthor (authorid);
```

Step 2: Project Setup:

Web API project can be created by following the below steps:

Open Microsoft Visual Studio 2017 > Click on File > New > Project > .NET Core from left pane > ASP.NET Core Web Application (.NET Core) > Enter the name "**LibraryApplicationAPI**" and Location where you want to save your project > OK

Select "**Web API**" template > OK

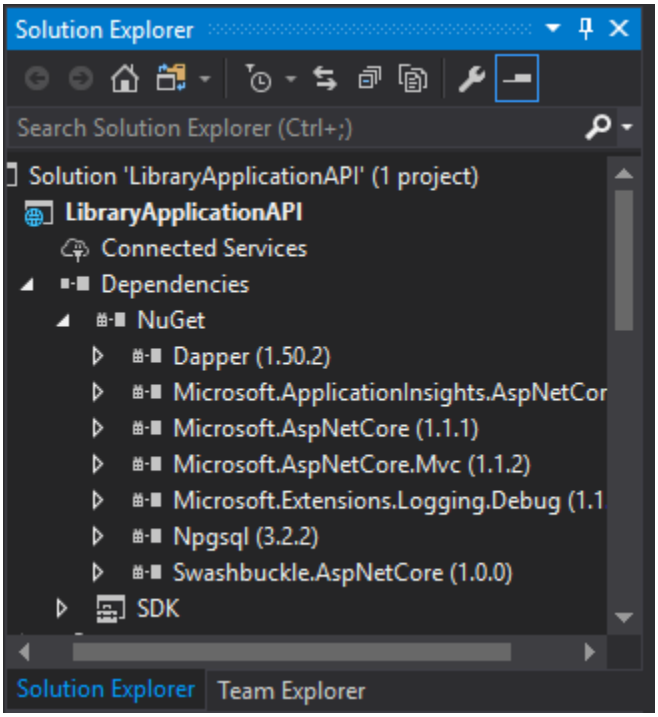
Web API project will be created.

*Install **Dapper** and **Npgsql** packages:*

Tools > NuGet Package Manager > Package Manager Console. Run the following commands to install **Dapper** and **Npgsql** packages.

```
Install-Package Dapper
Install-Package Npgsql
```

After installation, you should be able to see the dependencies in Solution Explorer like below:



Step 3: Connection setup

To setup the connection with the PostgreSQL database from Visual Studio, open **appsettings.json** file. By default, this file contains Logging settings. Connection String can be added after Logging settings as shown below:

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "DBInfo": {
```

```
  "Name": "librarydatabase",
  "ConnectionString": "User
ID=postgres;Password=xxxx;Host=localhost;Port=5432;Database=LibraryDB;Pooling=true;"
 }
}
```

Open **Startup.cs** file, add the following line in **ConfigureServices** method:

```
services.AddSingleton<IConfiguration>(Configuration);
```

Step 4: Swagger configuration

Install Swashbuckle:

Tools > NuGet Package Manager > Package Manager Console. Run the following command to install **Swashbuckle** package.

```
Install-Package Swashbuckle.AspNetCore
```

Open **Startup.cs** file

- Add the following lines in ConfigureServices method:

```
services.AddSwaggerGen(c =>
{
        c.SwaggerDoc("v1", new Info { Title = "Library API", Version = "v1" });
});
```

- Add the following lines in Configure method:

```
app.UseSwagger();
app.UseSwaggerUI(c =>
{
        c.SwaggerEndpoint("/swagger/v1/swagger.json", "Library API V1");
 });
```

For ease of use, different layers are added in the project:

Step 5: Models

Create **Models** folder in the project by following these steps:

Right click on LibraryApplicationAPI > Add > New Folder > Name it as **Models**

Add **BaseEntity.cs** class in Models folder by following these steps:

Right click on Models > Add > New Item > Class > Enter the name of class as **BaseEntity.cs**

```
namespace LibraryApplicationAPI.Models
{
   public abstract class BaseEntity
   {
   }
}
```

Add class **Book.cs** in Models folder by following these steps:

Right click on Models > Add > New Item > Class > Enter the name of class as **Book.cs**

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace LibraryApplicationAPI.Models
{
   public class Book : BaseEntity
   {
      [Key]
      public int bookid { get; set; }
      public string title { get; set; }
      public DateTime publisheddate { get; set; }
      public IEnumerable<BookAuthor> bookauthors { get; set; }
      public int libraryid { get; set; }
      public Library librarydetails { get; set; }
      public bool isavailable { get; set; }
      public int patronid { get; set; }
      public Patron patrondetails { get; set; }
      public DateTime duedate { get; set; }

      public Book()
      {
         bookauthors = new List<BookAuthor>();
      }
   }
}
```

This class stores the properties of variables that will be used when communicating with PostgreSQL database. The variables bookid, title, publisheddate, libraryid, isavailable, patronid, duedate are the attributes of the books table that was created in PostgreSQL database. The variable bookauthors displays the list of authors for a book. The variable librarydetails displays the details of library where a book is shelved. The variable patrondetails displays the details of patron who issued the book.

Note: To know the details of **Library** and **Patron** classes, refer to the code in the "LibraryApplicationAPI" project attached.

Add class **Author.cs** in Models folder by following these steps:

Right click on Models > Add > New Item > Class > Enter the name of class as **Author.cs**

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace LibraryApplicationAPI.Models
{
   public class Author : BaseEntity
   {
      public int authorid { get; set; }
      public string fname { get; set; }
      [Required]
      public string lname { get; set; }
      public IEnumerable<BookAuthor> bookauthors { get; set; }

      public Author()
      {
         bookauthors = new List<BookAuthor>();
      }
   }
}
```

This class stores the properties of variables that will be used when communicating with PostgreSQL database. The variables authorid, fname, lname are the attributes of the authors table that was created in PostgreSQL database. The variable bookauthors displays the list of books written by an author.

Add class **BookAuthor.cs** in Models folder by following these steps:

Right click on Models > Add > New Item > Class > Enter the name of class as **BookAuthor.cs**

```
namespace LibraryApplicationAPI.Models
{
   public class BookAuthor : BaseEntity
   {
      public int bookid { get; set; }
      public Book Book { get; set; }
      public int authorid { get; set; }
      public Author Author { get; set; }
   }
```

```
}
```

This class stores many-to-many relationship between book and author as a book may have multiple authors and an author may write many books.

Step 6: Repository:

Create **Repository** folder in the project by following these steps:

Right click on LibraryApplicationAPI > Add > New Folder > Name it as **Repository**

Add an interface named **IRepository.cs** class in Repository folder by following these steps:

Right click on Repository > Add > New Item > Class > Enter the name of class as **IRepository.cs**

```
using LibraryApplicationAPI.Models;
using System.Collections.Generic;

namespace LibraryApplicationAPI.Repository
{
   public interface IRepository<T> where T : BaseEntity
   {
      T Add(T item);
      void Remove(int id);
      T Update(T item);
      T FindByID(int id);
      IEnumerable<T> FindAll();
   }
}
```

Add class **BookRepository.cs** in Repository folder by following these steps:

Right click on Repository > Add > New Item > Class > Enter the name of class as **BookRepository.cs**

```
using Dapper;
using LibraryApplicationAPI.Models;
using Microsoft.Extensions.Configuration;
using Npgsql;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
```

```
/// <summary>
/// Reference: http://techbrij.com/asp-net-core-postgresql-dapper-crud
/// </summary>

namespace LibraryApplicationAPI.Repository
{
   public class BookRepository : IRepository<Book>
   {
      private string connectionString;
      public BookRepository(IConfiguration configuration)
      {
         connectionString = configuration.GetValue<string>("DBInfo:ConnectionString");
      }

      internal IDbConnection Connection
      {
         get
         {
            return new NpgsqlConnection(connectionString);
         }
      }

      /// <summary>
      /// Add a book
      /// </summary>
      /// <param name="book"></param>
      /// <returns></returns>
      public Book Add(Book book)
      {
         Book AddedBook;
         using (IDbConnection dbConnection = Connection)
         {
            dbConnection.Open();
            bool isAvailable = true;
            if (book.librarydetails.libraryid != 0)
            {
               var sql = "SELECT * FROM libraries where libraryid=" +
book.librarydetails.libraryid;
               List<Library> libraryList = Connection.Query<Library>(sql).ToList();
               if (libraryList.Count == 0)
               {
                  return null;
               }

               dbConnection.Execute("INSERT INTO books(title, publisheddate, libraryid,
isavailable) VALUES" +
```

```
                "(@title, @publisheddate, @libraryid, @isavailable)", new { title = book.title, publisheddate = book.publisheddate, libraryid = book.librarydetails.libraryid, isavailable = isAvailable });
            }
            else
            {
                dbConnection.Execute("INSERT INTO books(title, publisheddate, isavailable) VALUES" +
                "(@title, @publisheddate, @isavailable)", new { title = book.title, publisheddate = book.publisheddate, isavailable = isAvailable });
            }
            var BooksList = FindAll();
            Book bookElement = BooksList.Last();
            int addedBookID = bookElement.bookid;
            foreach (var author in book.bookauthors)
            {
                var getAuthor = "select * from authors where authorid = " + author.authorid;
                List<BookAuthor> authorsList = Connection.Query<BookAuthor>(getAuthor).ToList();
                if(authorsList.Count == 0)
                {
                    dbConnection.Execute("DELETE FROM books WHERE bookid=@bookid", new { bookid = addedBookID });
                    return null;
                }
                dbConnection.Execute("INSERT INTO bookauthor(bookid, authorid) VALUES (@bookid, @authorid)", new { bookid = addedBookID, authorid = author.authorid });
            }
            AddedBook = FindByID(addedBookID);
            dbConnection.Close();
        }
        return AddedBook;
    }

    /// <summary>
    /// Gets all the books
    /// </summary>
    /// <returns></returns>
    public IEnumerable<Book> FindAll()
    {
        var sql = "SELECT * FROM books";
        List<Book> booksList = Connection.Query<Book>(sql).ToList();

        foreach (Book book in booksList)
        {
            var getLibraryDetails = "select * from libraries where libraryid = " + book.libraryid;
```

```
            Library libraryDetailsList = Connection.Query<Library>(getLibraryDetails).FirstOrDefault();

            Library library = new Library();
            if(libraryDetailsList != null)
            {
               library.libraryid = book.libraryid;
               library.libraryname = libraryDetailsList.libraryname;
               library.address = libraryDetailsList.address;
            }

            var getPatronDetails = "select * from patrons where patronid = " + book.patronid;
            Patron patronDetails = Connection.Query<Patron>(getPatronDetails).FirstOrDefault();

            var getBookAuthors = "select authorid, bookid from bookauthor where bookid = " + book.bookid;
            List<BookAuthor> bookAuthorsList = Connection.Query<BookAuthor>(getBookAuthors).ToList();
            List<BookAuthor> bookAuthors = new List<BookAuthor>();
            foreach (BookAuthor bookAuthor in bookAuthorsList)
            {
               var getAuthorID = bookAuthor.authorid;
               var getBooksDetails = "select * from authors where authorid=" + getAuthorID;
               List<Author> authorsList = Connection.Query<Author>(getBooksDetails).ToList();
               foreach (Author author in authorsList)
               {
                  bookAuthor.authorid = author.authorid;
                  bookAuthor.Author = author;
               }
               bookAuthors.Add(bookAuthor);
            }
            book.librarydetails = library;
            book.bookauthors = bookAuthors;
            book.patrondetails = patronDetails;
         }
         return booksList;
      }


      /// <summary>
      /// Gets the book by ID
      /// </summary>
      /// <param name="id"></param>
      /// <returns></returns>
      public Book FindByID(int id)
      {
```

```
        Book book;
        IDbConnection dbConnection = Connection;
        dbConnection.Open();
        book = dbConnection.Query<Book>("SELECT * FROM books WHERE bookid = @bookid", new { bookid = id }).FirstOrDefault();
        if(book != null)
        {
            var getPatronDetails = "select * from patrons where patronid = " + book.patronid;
            Patron patronDetails = Connection.Query<Patron>(getPatronDetails).FirstOrDefault();
            var getLibraryDetails = "select * from libraries where libraryid = " + book.libraryid;
            Library libraryDetailsList = Connection.Query<Library>(getLibraryDetails).FirstOrDefault();
            Library library = new Library();
            if (libraryDetailsList != null)
            {
                library.libraryid = book.libraryid;
                library.libraryname = libraryDetailsList.libraryname;
                library.address = libraryDetailsList.address;
            }
            var getBookAuthors = "select authorid, bookid from bookauthor where bookid = " + book.bookid;
            List<BookAuthor> bookAuthorsList = Connection.Query<BookAuthor>(getBookAuthors).ToList();
            List<BookAuthor> bookAuthors = new List<BookAuthor>();
            foreach (BookAuthor bookAuthor in bookAuthorsList)
            {
                var getAuthorID = bookAuthor.authorid;
                var getBooksDetails = "select * from authors where authorid=" + getAuthorID;
                List<Author> authorsList = Connection.Query<Author>(getBooksDetails).ToList();
                foreach (Author author in authorsList)
                {
                    bookAuthor.authorid = author.authorid;
                    bookAuthor.Author = author;
                }
                bookAuthors.Add(bookAuthor);
            }
            book.librarydetails = library;
            book.bookauthors = bookAuthors;
            book.patrondetails = patronDetails;
            dbConnection.Close();
        }
        return book;
    }

    /// <summary>
    /// Delete the book
```

```
        /// </summary>
        /// <param name="id"></param>
        public void Remove(int id)
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                dbConnection.Execute("DELETE FROM books WHERE bookid=@bookid", new { bookid = id });
                dbConnection.Close();
            }
        }


        /// <summary>
        /// Delete the book
        /// </summary>
        /// <param name="book"></param>
        /// <returns></returns>
        public Book Update(Book book)
        {
            using (IDbConnection dbConnection = Connection)
            {
                dbConnection.Open();
                var getBookDetails = "select * from books where bookid = " + book.bookid;
                Book bookDetails = Connection.Query<Book>(getBookDetails).FirstOrDefault();
                dbConnection.Query("UPDATE books SET title = @title, publisheddate = @publisheddate WHERE bookid = @bookid", book);
                foreach (BookAuthor bookAuthor in book.bookauthors)
                {
                    var getBookAuthors = "select authorid, bookid from bookauthor where authorid = " + bookAuthor.authorid + "and bookid =" + book.bookid;
                    List<BookAuthor> bookAuthorsList = Connection.Query<BookAuthor>(getBookAuthors).ToList();
                    if (bookAuthorsList.Count == 0)
                    {
                        var getAuthor = "select * from authors where authorid = " + bookAuthor.authorid;
                        List<BookAuthor> authorsList = Connection.Query<BookAuthor>(getAuthor).ToList();
                        if(authorsList.Count == 0)
                        {
                            dbConnection.Query("UPDATE books SET title = @title, publisheddate = @publisheddate WHERE bookid = @bookid", new { title = bookDetails.title, publisheddate = bookDetails.publisheddate, bookid = book.bookid});
                            return null;
                        }
```

```
                dbConnection.Execute("INSERT INTO bookauthor (bookid, authorid) VALUES(@bookid, @authorid)", new { bookid = book.bookid, authorid = bookAuthor.authorid });
            }
        }
        Book UpdatedBook = FindByID(book.bookid);
        dbConnection.Close();
        return UpdatedBook;
      }
    }

    /// <summary>
    /// Get books by author
    /// </summary>
    /// <param name="searchString"></param>
    /// <returns></returns>
    public IEnumerable<Book> GetByAuthor(string searchString)
    {
      var sql = "select * from authors where fname ilike '%" + searchString + "%' or lname ilike '%" + searchString + "%'";
      List<Author> authorsList = Connection.Query<Author>(sql).ToList();
      List<Book> booksList = new List<Book>();
      foreach(Author author in authorsList)
      {
        var getBooksByAuthors = "select bookid from bookauthor where authorid = " + author.authorid;
        List<BookAuthor> booksByAuthorsList = Connection.Query<BookAuthor>(getBooksByAuthors).ToList();
        foreach (BookAuthor bookAuthor in booksByAuthorsList)
        {
          var getAuthorID = bookAuthor.authorid;
          var getBookID = bookAuthor.bookid;
          var getBooksDetails = "select * from books where bookid=" + getBookID;
          List<Book> books = Connection.Query<Book>(getBooksDetails).ToList();
          foreach (Book b in books)
          {
            var getLibraryDetails = "select * from libraries where libraryid = " + b.libraryid;
            Library libraryDetailsList = Connection.Query<Library>(getLibraryDetails).FirstOrDefault();
            Library library = new Library();
            if(b.libraryid != 0)
            {
              library.libraryid = b.libraryid;
              library.libraryname = libraryDetailsList.libraryname;
              library.address = libraryDetailsList.address;
            }
```

```
                var getPatronDetails = "select * from patrons where patronid = " + b.patronid;
                Patron patronDetails = Connection.Query<Patron>(getPatronDetails).FirstOrDefault();

                var getBookAuthors = "select authorid, bookid from bookauthor where bookid = " + b.bookid;
                List<BookAuthor> bookAuthorsList = Connection.Query<BookAuthor>(getBookAuthors).ToList();
                List<BookAuthor> bookAuthors = new List<BookAuthor>();
                foreach (BookAuthor ba in bookAuthorsList)
                {
                   var authorID = ba.authorid;
                   var getAuthors = "select * from authors where authorid=" + authorID;
                   List<Author> listAuthors = Connection.Query<Author>(getAuthors).ToList();
                   foreach (Author a in listAuthors)
                   {
                      ba.authorid = a.authorid;
                      ba.Author = a;
                   }
                   bookAuthors.Add(ba);
                }
                b.librarydetails = library;
                b.bookauthors = bookAuthors;
                b.patrondetails = patronDetails;
                booksList.Add(b);
             }
          }
       }
       return booksList;
    }

    /// <summary>
    /// Gets the book that are due
    /// </summary>
    /// <returns></returns>
    public IEnumerable<Book> BooksDue()
    {
       String bookDueDate = DateTime.Now.ToString("yyyy-MM-dd");
       var sql = "SELECT * FROM books where duedate < '" + bookDueDate + "'";
       List<Book> booksList = Connection.Query<Book>(sql).ToList();

       foreach (Book book in booksList)
       {
          var getLibraryDetails = "select * from libraries where libraryid = " + book.libraryid;
```

```
            Library libraryDetailsList = Connection.Query<Library>(getLibraryDetails).FirstOrDefault();
            Library library = new Library();
            library.libraryid = book.libraryid;
            library.libraryname = libraryDetailsList.libraryname;
            library.address = libraryDetailsList.address;

            var getPatronDetails = "select * from patrons where patronid = " + book.patronid;
            Patron patronDetails = Connection.Query<Patron>(getPatronDetails).FirstOrDefault();

            var getBookAuthors = "select authorid, bookid from bookauthor where bookid = " + book.bookid;
            List<BookAuthor> bookAuthorsList = Connection.Query<BookAuthor>(getBookAuthors).ToList();
            List<BookAuthor> bookAuthors = new List<BookAuthor>();
            foreach (BookAuthor bookAuthor in bookAuthorsList)
            {
               var getAuthorID = bookAuthor.authorid;
               var getBooksDetails = "select * from authors where authorid=" + getAuthorID;
               List<Author> authorsList = Connection.Query<Author>(getBooksDetails).ToList();
               foreach (Author author in authorsList)
               {
                  bookAuthor.authorid = author.authorid;
                  bookAuthor.Author = author;
               }
               bookAuthors.Add(bookAuthor);
            }
            book.librarydetails = library;
            book.bookauthors = bookAuthors;
            book.patrondetails = patronDetails;
         }
         return booksList;
      }

      /// <summary>
      /// Transfers the book to another library
      /// </summary>
      /// <param name="b"></param>
      /// <returns></returns>
      public int TransferBook(Book b)
      {
         using (IDbConnection dbConnection = Connection)
         {
            dbConnection.Open();
            var sql = "SELECT * from books where bookid = " + b.bookid;
            Book book = dbConnection.Query<Book>(sql).FirstOrDefault();
```

```
            if(book.isavailable == false)
            {
               return 0;
            }
            else if(book.libraryid == b.libraryid)
            {
               return 1;
            }
            else
            {
               var getLibrary = "select * from libraries where libraryid = " + b.libraryid;
               List<Library> libraryList = Connection.Query<Library>(getLibrary).ToList();
               if (libraryList.Count == 0)
               {
                  return 3;
               }
               dbConnection.Query("UPDATE books SET libraryid = @libraryid WHERE bookid
= @bookid", new { libraryid = b.libraryid, bookid = b.bookid });
               return 2;
            }
         }
      }
   }
}
```

Note: Kindly refer to the project attachment for more functions implemented in BookRepository.cs file.
As shown in the above code, the methods of IRepository has been implemented and Dapper is used to perform all the operations. The connection string is retrieved by the following line of code: connectionString = configuration.GetValue<string>("DBInfo:ConnectionString");

DBInfo:ConnectionString had been defined in appsettings.json in **step 3**

Many-to-Many relationship solution: Ideally, when there is many-to-many relationship, the circular reference occurs in the JSON response while getting all the books along with its authors. To avoid that circular reference, I have stored the SQL queries response in the various objects. I am modifying the BookAuthor object to store the corresponding objects. As if you see in the above code, in FindAll() method, I have written a SQL code as (`"SELECT * FROM books"`) to get all the books. For each book, I am fetching the authors from bookauthors table by writing the SQL code as (`"select authorid, bookid from bookauthor where bookid = " + book.bookid;`). For each authorid retrieved from the bookauthors table, I have stored the author details in the Author object by SQL query (`"select * from authors where authorid=" + getAuthorID;`). For each author object retrieved, I am modifying the object of BookAuthor. BookAuthor object contains bookid, Book object, authorid, Author object. However, book details have already been defined so Book object will be kept null to avoid circular references. authorid will be modified by getting the authorid

from bookauthors table. Author object will be modified by overwriting it with the author details that was retrieved for each author of bookauthors table.

Note: Other Repository class files have been generated which can be referred in the project attachment.

Step 7: Controllers

Create **Controllers** folder in the project by following these steps:

Right click on LibraryApplicationAPI > Add > New Folder > Name it as **Controllers**

Add **BookController.cs** class in Controllers folder by following these steps:

Right click on Controllers > Add > New Item > Class > Enter the name of class as **BookController.cs**

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using LibraryApplicationAPI.Repository;
using Microsoft.Extensions.Configuration;
using LibraryApplicationAPI.Models;

/// <summary>
/// A book controller that handles the routes
/// Reference: http://techbrij.com/asp-net-core-postgresql-dapper-crud
/// </summary>
namespace LibraryApplicationAPI.Controllers
{
   [Route("api/[controller]")]
   public class BookController : Controller
   {
      public readonly BookRepository _bookRepository;

      public BookController(IConfiguration configuration)
      {
         _bookRepository = new BookRepository(configuration);
      }

      [HttpGet]
      public IEnumerable<Book> GetAll()
      {
         return _bookRepository.FindAll();
      }

      [HttpGet("{id}", Name = "GetBook")]
```

```
public IActionResult GetById(int id)
{
    var book = _bookRepository.FindByID(id);
    if (book == null)
    {
        return Ok("Book doesn't exist!");
    }
    return Ok(book);
}

[HttpPost]
public IActionResult Create([FromBody] Book item)
{
    if (item == null)
    {
        return BadRequest();
    }

    Library librarydetails = new Library();
    if(item.librarydetails != null)
    {
        librarydetails.libraryid = item.librarydetails.libraryid;
    }
    Book book = new Book();
    book.title = item.title;
    book.publisheddate = item.publisheddate;
    book.librarydetails = librarydetails;
    book.bookauthors = item.bookauthors;

    var createdBook = _bookRepository.Add(book);

    if(createdBook == null)
    {
        return Ok("Author doesn't exist!");
    }

    return CreatedAtRoute("GetBook", new { id = createdBook.bookid }, createdBook);
}

[HttpPut("{id}")]
public IActionResult Update(int id, [FromBody] Book item)
{
    if (item == null || item.bookid != id)
    {
        return BadRequest();
    }
```

```
    var book = _bookRepository.FindByID(id);
    if (book == null)
    {
        return Ok("Book doesn't exist!");
    }

    book.title = item.title;
    book.publisheddate = item.publisheddate;
    book.bookauthors = item.bookauthors;

    var updatedBook = _bookRepository.Update(book);
    if(updatedBook == null)
    {
        return Ok("Author doesn't exist!");
    }
    return Ok(updatedBook);
}

[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    var book = _bookRepository.FindByID(id);
    if (book == null)
    {
        return Ok("Book doesn't exist!");
    }

    _bookRepository.Remove(id);
    return Ok("Book successfully deleted!");
}

[HttpGet("getByAuthor/{searchString}")]
public IActionResult GetByAuthor(string searchString)
{
    var book = _bookRepository.GetByAuthor(searchString);
    return Ok(book);
}

[HttpGet("getBooksDue")]
public IEnumerable<Book> GetBooksDue()
{
    return _bookRepository.BooksDue();
}

[HttpPut("transferBook")]
```

```
        public IActionResult TransferBook([FromBody] Book item)
        {
            if (item == null)
            {
                return BadRequest();
            }

            var book = _bookRepository.FindByID(item.bookid);
            if (book == null)
            {
                return Ok("Book doesn't exist!");
            }

            book.bookid = item.bookid;
            book.libraryid = item.libraryid;

            var transferToOtherLibrary = _bookRepository.TransferBook(book);
            if (transferToOtherLibrary == 0)
            {
                return Ok("Can't transfer book to other library as it is already issued! Try again when
the book is returned");
            }
            else if(transferToOtherLibrary == 1)
            {
                return Ok("Can't transfer book as it is in the same library!");
            }
            else if(transferToOtherLibrary == 3)
            {
                return Ok("Library doesn't exist!");
            }
            var transferredBook = _bookRepository.FindByID(book.bookid);
            return Ok(transferredBook);
        }
    }
}
```

I have created a BookRepository object in the constructor.

Note: Kindly refer to the project attachment for more functions implemented in BookRepository.cs file.

Following are the different HTTP methods:
HttpGet: Retrieves the books details from the database.
HttpPost: Inserts the book into the database. The details of the book to be inserted is placed in request body.
HttpPut: Updates the book details in the database.

HttpDelete: Deletes the book from the database.

Note: Other Controller class files have been generated which can be referred in the project attachment.

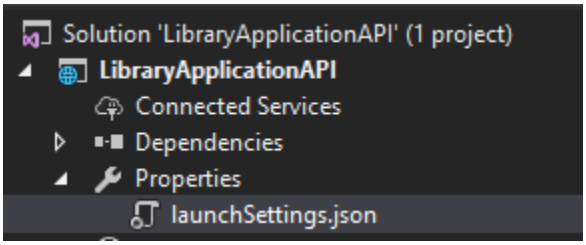
Step 8: Using Postman

The routes are created as shown in the code of BookController as [Route("api/[controller]")].
The [controller] needs to be replaced with the name of controller class minus Controller suffix.
So, the route becomes api/book

For books API:

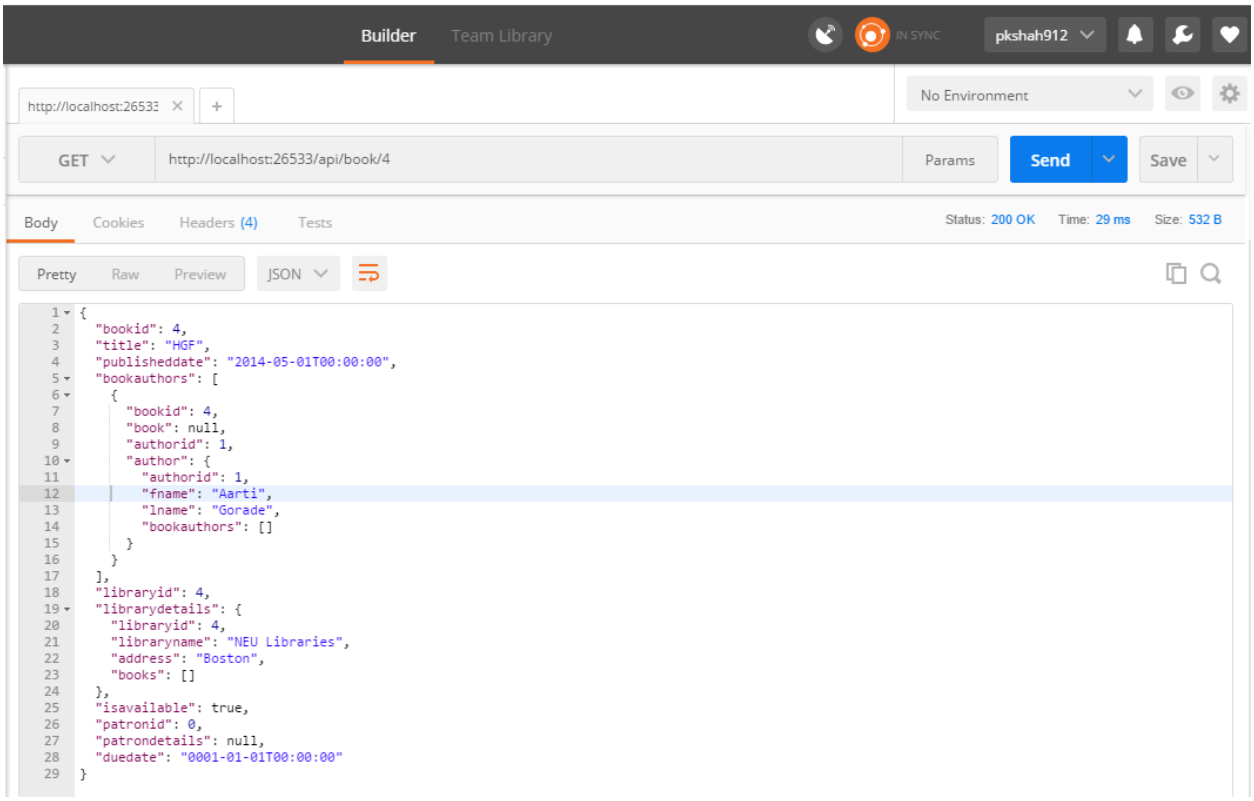
API	Description	Request body	Response body
GET /api/book	Gets all the books and its details	None	A list of books along with its details
GET /api/book/{id}	Gets a book along with its details by its ID	None	A book along with its details
POST /api/book	Add a new book	Book details	Book ID along with its details
PUT /api/book/{id}	Update an existing book	Book details	Updated book details
DELETE /api/book/{id}	Delete a book	None	None

The port number can be seen from Solution Explorer in Properties > launchSettings.json as shown in the figure:



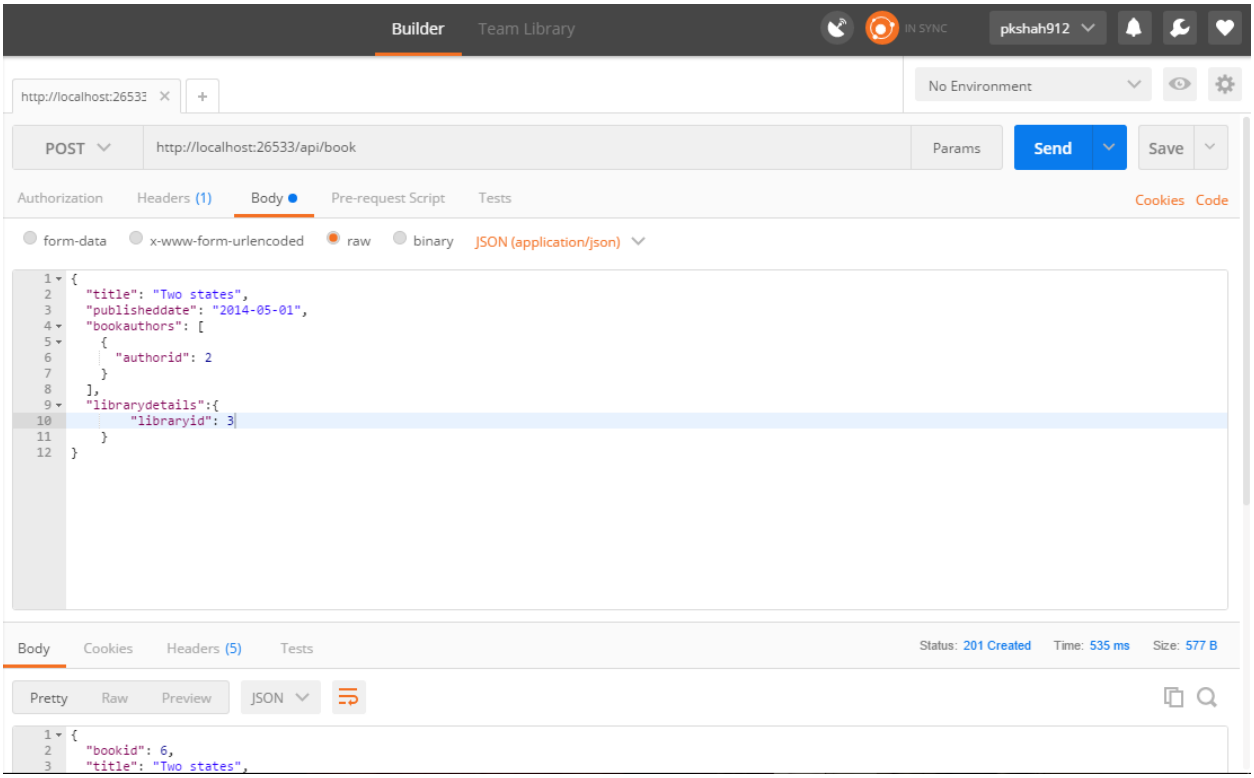
Below is the screenshot of how Postman looks like:

For GET method to get book by ID 4:



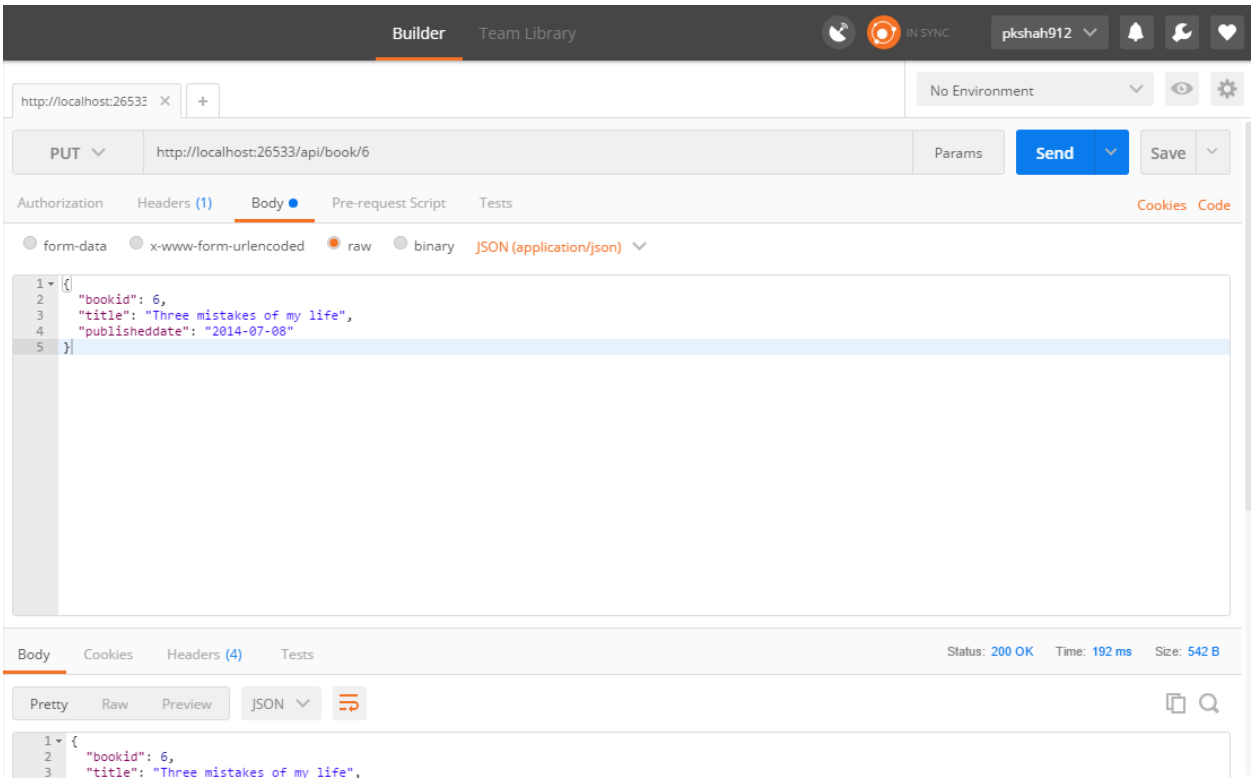
For POST method to insert a book:

Click on Body > raw > JSON (application/json) from dropdown > Enter the details of book to be inserted in the text area.

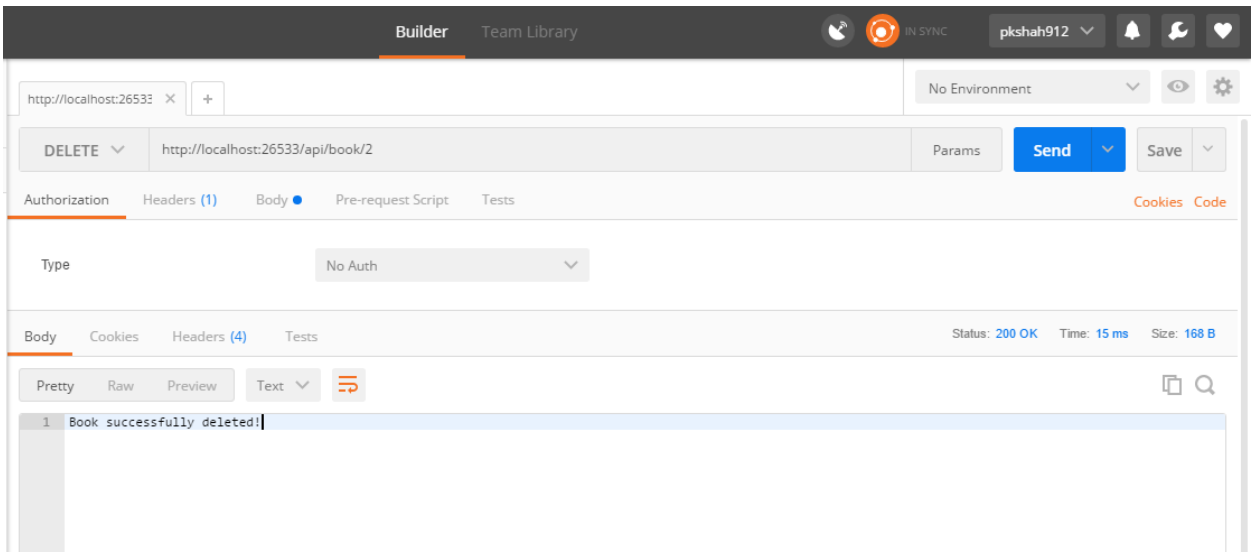


For PUT method to update the book ID 6:

Click on Body > raw > JSON (application/json) from dropdown > Enter the details of book to be updated in the text area.



For DELETE method to delete the book 2:



Step 9: Swagger (Optional)

To see the document generated that describes the endpoints, press F5 to launch the application. Enter the URL: http://localhost:port/swagger/ on the browser. Below is the screenshot of how swagger looks like: