ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

Курсовой проект

по дисциплине «Компьютерные технологии управления в технических системах»

на тему «Приложение с базой данных»

Выполнил: ст. гр. ТУУ–151

Николаев А.Ю., Куликова К.И., Гришина М.А.

Проверил: доц. к.т.н. Васильева М.А.

Москва – 2020 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc62386636)

[1. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ 3](#_Toc62386637)

[1.1. ИНФОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ 3](#_Toc62386638)

[1.1.1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 3](#_Toc62386639)

[1.1.2. АНАЛИЗ ИНФОРМАЦИОННЫХ ЗАДАЧ И КРУГА ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ 5](#_Toc62386640)

[1.2. ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД 6](#_Toc62386641)

[1.2.1. ПРЕОБРАЗОВАНИЕ ER-ДИАГРАММЫ В СХЕМУ БАЗЫ ДАННЫХ 6](#_Toc62386642)

[1.2.2. СОСТАВЛЕНИЕ РЕЛЯЦИОННЫХ ОТНОШЕНИЙ 8](#_Toc62386643)

[1.2.3. НОРМАЛИЗАЦИЯ ПОЛУЧЕННЫХ ОТНОШЕНИЙ 9](#_Toc62386644)

[1.2. ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БД 10](#_Toc62386645)

[1.2.1. РАЗРАБОТКА СКРИПТОВ НА СОЗДАНИЕ БАЗЫ ДАННЫХ И ТАБЛИЦ 10](#_Toc62386646)

[2. РАЗРАБОТКА ПРИЛОЖЕНИЯ 15](#_Toc62386647)

[2.1. NHIBERNATE 15](#_Toc62386648)

[2.2. ORM, репозитории, сессии, сервисы. 16](#_Toc62386649)

[2.3. СЕССИИ 17](#_Toc62386650)

[2.4. ФАБРИКА СЕССИЙ (SESSION FACTORY) 18](#_Toc62386651)

[2.5. ТЕСТИРОВАНИЕ МАППИНГОВ 20](#_Toc62386652)

[2.6. ПРИМЕРЫ ЗАПРОСОВ 21](#_Toc62386653)

[2.7. ПРИНЦИПЫ REST 27](#_Toc62386654)

[СПИСОК ЛИТЕРАТУРЫ 32](#_Toc62386655)

[ПРИЛОЖЕНИЕ 33](#_Toc62386656)

ВВЕДЕНИЕ

Сегодня любая IT-разработка очень тесно связана с использованием баз данных. Чаще всего приходится работать с реляционными СУБД. Поэтому приходится использовать различные драйвера к базам данных для необходимых языков программирования, что приводит к постоянному написанию SQL-запросов внутри кода программы, к использованию большого количества необходимых классов, чтобы выполнить достаточно простые операции получения/изменения информации внутри базы.

Таким образом, появляется задача обеспечения работы с данными в терминах классов/объектов, а не таблиц. Необходимо преобразовать термины и данные классов в данные, пригодные для хранения в СУБД, а также обеспечить интерфейс для операций создания, получения, обновления, удаления (create, read, update, delete – CRUD) данных, и в целом избавиться от написания SQL-запросов в коде программы для взаимодействия с СУБД.

Решением проблемы является объектно-реляционное отображение (object relational mapping – ORM) – технология, связывающая объекты в рамках понятий баз данных и объекты в рамках объектно-ориентированного программирования. Применение объектно-реляционного отображения в настоящее время является общим средством в процессе разработки сложных систем, позволяющим объединить объектно-ориентированную модель представления данных с реляционной.

ORM является промежуточным слоем между базой данных и кодом программы, которая позволяет выполнять CRUD-операции в базе данных, в частности, в таблицах, с помощью использования специального API и ООП-объектов.

Целью данной работы является реализация подобного отображения, которое можно будет использовать на практике.

1. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ
   1. ИНФОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ
      1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

БД создается для домашнего обслуживания библиотеки. Библиотека содержит книги разных авторов, изданий, тематик. БД предназначена для поиска выбранной книги на полке.

ER-диаграмма БД представлена на Рисунке 1.

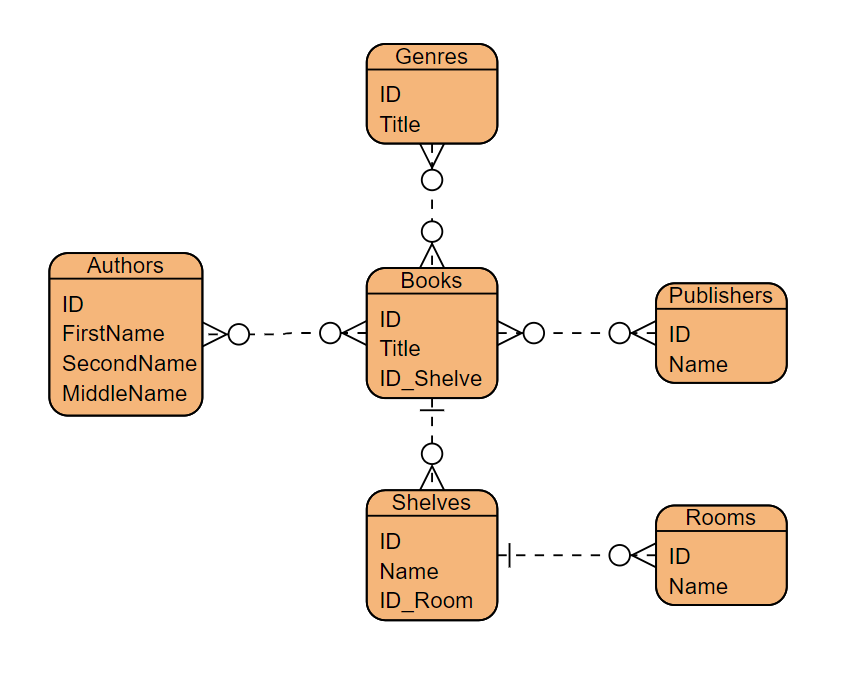


Рисунок 1 - ER-диаграмма

В полученной ER-диаграмме присутствует связь многие - ко - многим, поэтому вводится специальная вспомогательная сущность, которая является соединением первичных ключей соответствующих сущностей. Таким образом разбиваются все связи типа n:m. Затем строится уточненная ER-диаграмма (Рисунок 2).

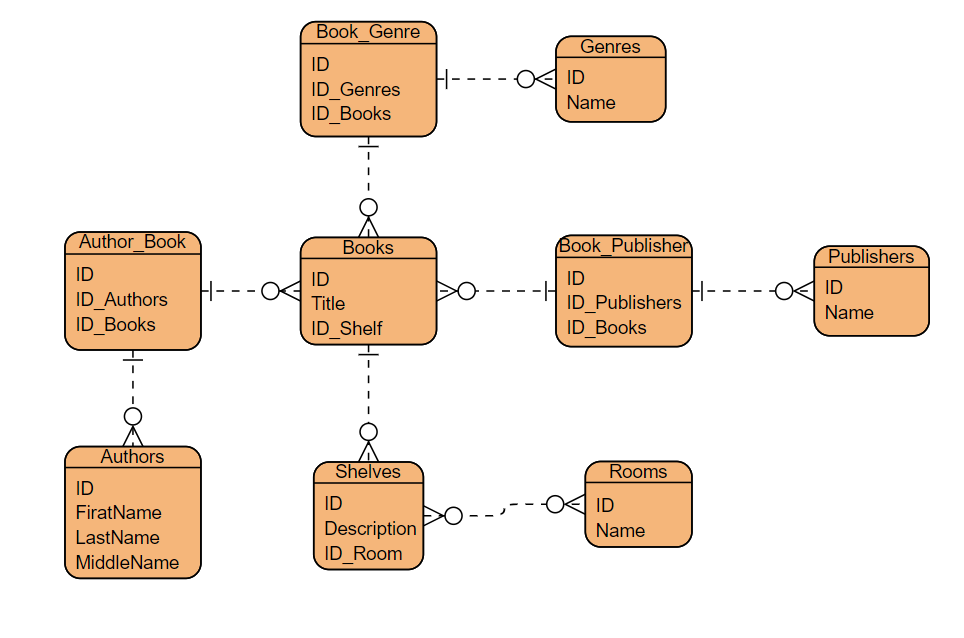


Рисунок 2 - Уточненная ER-диаграмма

* + 1. АНАЛИЗ ИНФОРМАЦИОННЫХ ЗАДАЧ И КРУГА ПОЛЬЗОВАТЕЛЕЙ СИСТЕМЫ

Система создаётся для домашнего обслуживания библиотеки.

Определим границы информационной поддержки пользователей:

1) Функциональные возможности:

1. Ведение БД (запись, чтение, модификация, удаление в архив)

2. Обеспечение логической непротиворечивости БД

3. Обеспечение защиты данных от несанкционированного или случайного доступа (определение прав доступа)

4. Реализация наиболее часто встречающихся запросов в готовом виде.

2) Готовые запросы:

1. Выдавать список книг по названию.

2. Выдавать список трудов данного автора (учитывать труды, выполненные в соавторстве).

3. Выдавать список книг по данной тематике.

4. Выдавать список книг данного издательства.

5. Выдавать местонахождение данной книги.

1.2. ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БД

1.2.1. ПРЕОБРАЗОВАНИЕ ER-ДИАГРАММЫ В СХЕМУ БАЗЫ ДАННЫХ

База данных создаётся на основании схемы базы данных. Преобразование ER-диаграммы в схему БД выполняется путем сопоставления каждой сущности и каждой связи, имеющей атрибуты, отношения (таблицы БД). На схеме (Рисунок 3) введены обозначения:

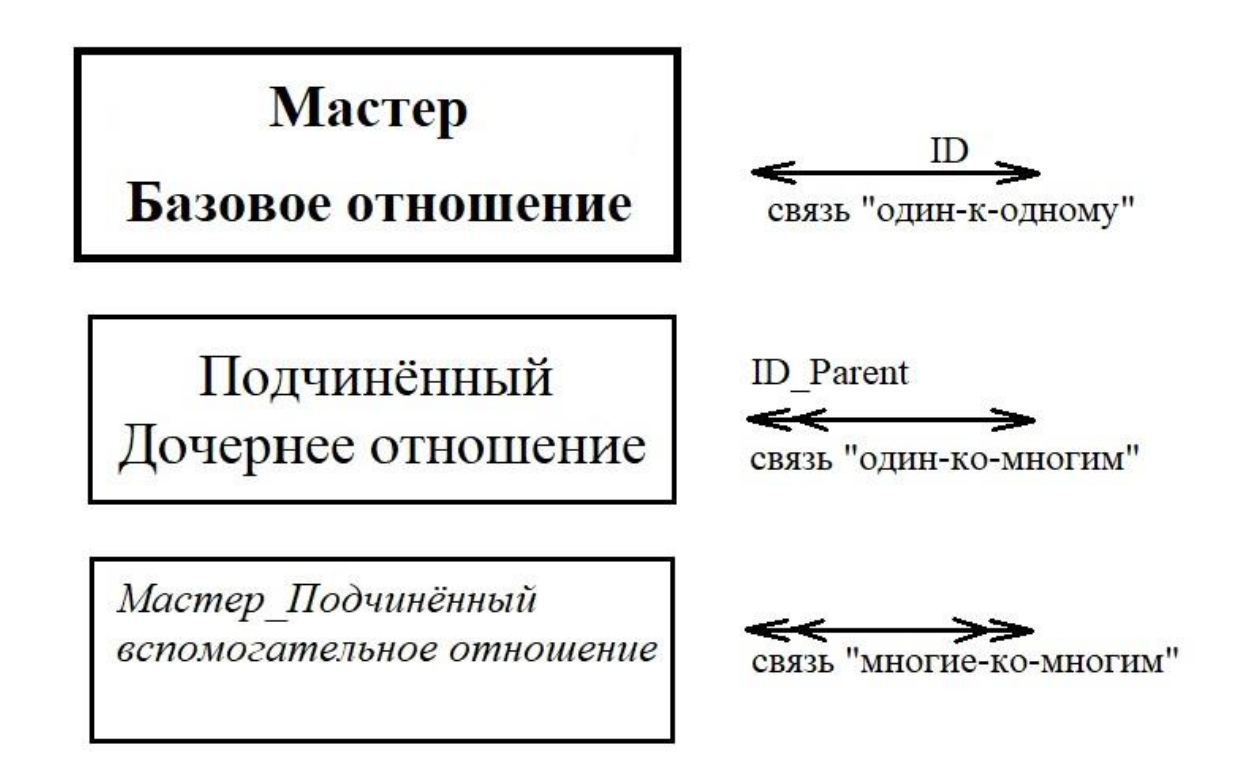


Рисунок 3 - Обозначения, используемые на схеме базы данных

Полученная схема реляционной базы данных (РБД) «Библиотека» приведена на Рисунке 4.

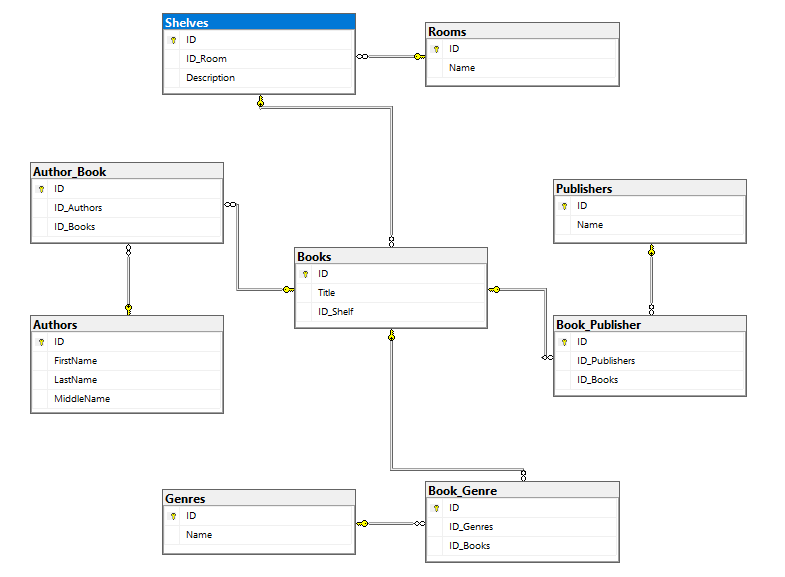


Рисунок 4 - Диаграмма базы данных

1.2.2. СОСТАВЛЕНИЕ РЕЛЯЦИОННЫХ ОТНОШЕНИЙ

Каждое реляционное отношение соответствует одной сущности (объекту ПрО) и в него вносятся все атрибуты сущности. Для каждого отношения необходимо определить первичный ключ и внешние ключи (если они есть). В том случае, если базовое отношение не имеет потенциальных ключей, вводится суррогатный первичный ключ, который не несёт с

Каждая сущность имеет связь многое-ко-многим, как показано на рисунке 1, где главной сущностью является “Books” — может принадлежать много авторов, жанров, одно издательство, находиться на одной полке в одной комнате. Но при этом много книг может быть в одной комнате. Автор может написать или участвовать во многих книгах.

1.2.3. НОРМАЛИЗАЦИЯ ПОЛУЧЕННЫХ ОТНОШЕНИЙ

1НФ. Наша база данных соответсвует 1НФ. Т.к все данные в полях атомарны, т.е одно поле – одно значение.

2НФ. Все отношения находятся во 2НФ, так как для каждого из них введен уникальный ключ и все записи от него зависят.

3НФ. Отношения в БД находится в 3НФ. Т.к третья нормальная форма исключает зависимость неключевых полей от других неключевых полей.

* 1. ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БД
     1. РАЗРАБОТКА СКРИПТОВ НА СОЗДАНИЕ БАЗЫ ДАННЫХ И ТАБЛИЦ

БД «Библиотека» проектируется в среде Microsoft SQL Server Management Studio.

Создание базы данных «Библиотека»:

create database DBLibraryHome

go

Создание таблицы «Authors» :

use DBLibraryHome

go

CREATE TABLE [dbo].[Authors]

(

[ID] int primary key

, [FirstName] nvarchar(255) not null

, [LastName] nvarchar(255) not null

, [MiddleName] nvarchar(255)

);

Создание таблицы «Genres»:

use DBLibraryHome

go

CREATE TABLE [dbo].[Genres]

(

[ID] int primary key

, [Name] nvarchar(255) not null

);

Создание таблицы «Publishers» :

use DBLibraryHome

CREATE TABLE [dbo].[Publishers]

(

[ID] int primary key

, Name nvarchar(255) not null

);

Создание таблицы «Shelves»:

CREATE TABLE [dbo].[Shelves]

(

[ID] int primary key

, Description nvarchar(255) not null

, FOREIGN KEY ([ID\_Room])

REFERENCES Rooms ([ID])

);

Создание таблицы «Rooms»:

CREATE TABLE [dbo].[Rooms]

(

[ID] int primary key

, Name nvarchar(255) not null

);

Создание таблицы «Books»:

CREATE TABLE [dbo].[Books]

(

[ID] int primary key

, Title nvarchar(255) not null

, [ID\_Shelve] int

, foreign key ([ID\_Shelve])

references Shelves([ID])

);

Создание таблицы «Author\_Book»:

use DBLibraryHome

CREATE TABLE [dbo].[Author\_Book]

(

[ID] int primary key

, [ID\_Authors] int not null

, [ID\_Books] int not null

, FOREIGN KEY ([ID\_Authors])

REFERENCES Authors ([ID])

, FOREIGN KEY ([ID\_Books])

REFERENCES Books ([ID])

);

Создание таблицы «Book\_Genre»:

use DBLibraryHome

CREATE TABLE [dbo].[Book\_Genre]

(

[ID] int primary key

, [ID\_Genres] int not null

, [ID\_Books] int not null

, FOREIGN KEY ([ID\_Genres])

REFERENCES Genres ([ID])

, FOREIGN KEY ([ID\_Books])

REFERENCES Books ([ID])

);

Создание таблицы «Book\_Publisher»:

use DBLibraryHome

CREATE TABLE [dbo].[Book\_Publisher]

(

[ID] int primary key

, [ID\_Publishers] int not null

, [ID\_Books] int not null

, FOREIGN KEY ([ID\_Publishers])

REFERENCES Publishers ([ID])

, FOREIGN KEY ([ID\_Books])

REFERENCES Books ([ID])

);

* + 1. РАЗРАБОТКА СКРИПТОВ НА ДОБАВЛЕНИЕ ДАННЫХ В ТАБЛИЦЫ

Заполнение таблицы «Authors»:

insert into Authors

values (1, 'Александр', 'Пушкин', 'Сергеевич')

, (2, 'Лев', 'Толстой', 'Николаевич')

, (3, 'Фёдор', 'Достоевский', 'Михайлович')

, (4, 'Денис', 'Фонвизин', 'Иванович')

, (5, 'Василь', 'Быков', 'Владимирович');

Заполнение таблицы «Books»:

insert into Books

values (1, 'Капитанская дочка', 1)

, (2, 'Война и Мир', 1)

, (3, 'Преступление и наказание', 1)

, (4, 'Недоросль', 2)

, (5, 'Сотников', 3);

Заполнение таблицы «Genres»:

insert into Genres

values (1, 'Роман')

, (2, 'Повесть')

, (3, 'Комедия');

Заполнение таблицы «Publishers»:

insert into Publishers

values (1, 'Эксмо')

, (2, 'Юрайт')

, (3, 'Рипол классик');

Заполнение таблицы «Rooms»:

insert into Rooms

values (1, 'Зал')

, (2, 'Спальня')

, (3, 'Коридор');

Заполнение таблицы «Shelves»:

insert into Shelves

values (1, 2, 'Верхняя полка')

, (2, 1, 'Нижняя полка')

, (3, 3, 'Средняя полка');

Заполнение таблицы «Author\_Book»:

insert into AuthorBook

values (1, 1, 1)

, (2, 2, 2)

, (3, 3, 3)

, (4, 4, 4)

, (5, 5, 5);

Заполнение таблицы «Book\_Publisher»:

insert into Book\_Publisher

values (1, 1, 1)

, (2, 2, 2)

, (3, 3, 3)

, (4, 1, 4)

, (5, 3, 5);

Заполнение таблицы «Book\_Genre»:

insert into Book\_Genre

values (1, 1, 1)

, (2, 2, 2)

, (3, 3, 3)

, (4, 2, 4)

, (5, 3, 5);

1. РАЗРАБОТКА ПРИЛОЖЕНИЯ
   1. NHIBERNATE

NHibernate — ORM-решение для платформы Microsoft .NET, портированное с Java. Это бесплатная библиотека с открытым кодом, распространяется под лицензией GNU Lesser General Public License.

NHibernate позволяет отображать объекты бизнес-логики на реляционную базу данных. По заданному XML-описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов.

* 1. ORM, репозитории, сессии, сервисы.

ORM – это аббревиатура для Object Relational Mapping (Объектно-реляционное отображение). Техника программирования, предназначенная для преобразования между несовместимыми типами данных в объектно-ориентированных языках программирования. Суть определения заключается в том, что создается абстракция - “виртуальная объектная база”, запросы к которой, преобразуются в SQL команды.

М. Фаулер даёт следующую формулировку архитектурному шаблону проектирования отображение данных (Data Mapper): слой отображателей (Mapper), который осуществляет передачу данных между объектами и БД, сохраняя последние независимыми друг от друга и от самого отображателя.

Типы отображения:

* Отображение на основе XML.
* Отображение на основе атрибутов.
* Текучее отображение (fluent mapping).
* Отображение на основе соглашений, иногда называемое автоотображением (auto-mapping).
  1. СЕССИИ

NHibernate сессию можно представить как абстрактный или виртуальный канал для БД. Раньше вам надо было создавать ADOConnection, открывать Connection, передавать Connection к объекту Command, создавать DataReader из объекта Connection и так далее.

С NHibernate мы просим sessionFactory объект Session, и все. NHibernate поддерживает все «реальные» сессии с БД (соединение, пулы и т.д.). Мы используем преимущества соединения с БД без знания всех тонкостей, лежащих в его основе.

Через объект сессии мы можем добавить новые данные в БД, изменить или удалить существующие данные, а также считывать данные из БД. Все эти операции могут быть выполнены в стиле ООП, без знания SQL и других специфических тонкостей БД. Объект сессии позволяет нам взаимодействовать с данными, хранящимися в БД, и не зависеть от выбранной СУБД (SQL Server, MySQL, Oracle и т.д.). NHibernate полностью абстрагирует эти детали от нас.

* 1. ФАБРИКА СЕССИЙ (SESSION FACTORY)

NHibernate использует объект фабрика, чтобы создать сущности сессии. Один объект фабрики сессии может создавать множество объектов сессии. Создание новой сессии очень дешёвая операция, в отличие от трудозатратной операции – создание фабрики сессий. В зависимости от сложности системы, на создание объекта фабрики сессий тратится значительное количество времени. По этой причине мы должны создавать фабрику сессий единожды в течении жизненного цикла приложения.

Фабрика сессий специфична к БД. Если наше приложение нуждается только в одной БД, нам необходима только одна фабрика. Если наше приложение взаимодействует с несколькими разными БД, тогда нам необходим объект фабрики сессий для каждой БД. В более развитых сценариях может случится так, что понадобится более, чем один объект фабрики сессий, даже, если мы имеем доступ к единственной БД. Причиной для этого может служить то, что наша БД имеет несколько разных схем, и мы хотим иметь для каждой схемы свой объект фабрики сессий. Другой причиной, по которой может существовать более одного объекта фабрики сессий для единственной БД, является существование нескольких предметных областей (часто называемый ограниченный контекст ), и мы хотим иметь доступ к БД из каждой подобласти через сессии, созданные фабрикой сессий ограниченного контекста.

Фабрика сессий потокобезопасна. Код, запущенный в разных потоках, может использовать тот же самый объект фабрики сессий для создания новых объектов сессии. Это контрастирует с тем фактом, что объект сессии может быть использован только в единственном потоке. Другими словами, объект сессии не потокобезопасен.

Важные интерфейсы Hibernate:

* SessionFactory (org.hibernate.SessionFactory) – неизменяемый потокобезопасный объект с компилированным маппингом для одной базы данных. Необходимо инициализировать SessionFactory всего один раз. Экземпляр SessionFactory используется для получения объектов Session, которые используются для операций с базами данных.
* Session (org.hibernate.Session) – однопоточный короткоживущий объект, который предоставляет связь между объектами приложения и базой данных. Он оборачивает JDBC java.sql.Connection и работает как фабрика для org.hibernate.Transaction. Разработчик должен открывать сессию по необходимости и закрывать ее сразу после использования. Экземпляр Session является интерфейсом между кодом в java приложении и hibernate framework и предоставляет методы для операций CRUD.
* Transaction (org.hibernate.Transaction) – однопоточный короткоживущий объект, используемый для атомарных операций. Это абстракция приложения от основных JDBC или JTA транзакций. org.hibernate.Session может занимать несколько org.hibernate.Transaction в определенных случаях.
  1. ТЕСТИРОВАНИЕ МАППИНГОВ

Для тестирования отображений FluentNHibernate предоставляет Persistence specification testing framework. Необходимо создать новую сущность PersistenceSpecification с универсальным типом сущности, которую нужно протестировать. Для каждого свойства вызывается метод CheckProperty, для ссылок вызывается CheckReference, для коллекций в связи многие-ко-многим – CheckList. Далее назначаются тестовые значения и параметр для сравнения. После вызывается VerifyTheMappings, который выполняет основные операции CRUD.

В тесте не важно, установлено ли какое-то конкретное или случайное значение. Важно то, что свойство может быть записано в базу данных и получено из нее.

* 1. ПРИМЕРЫ ЗАПРОСОВ
* Метод GET

/api/author - список всех авторов из БД . Результат запроса представлен на Рисунке 5.

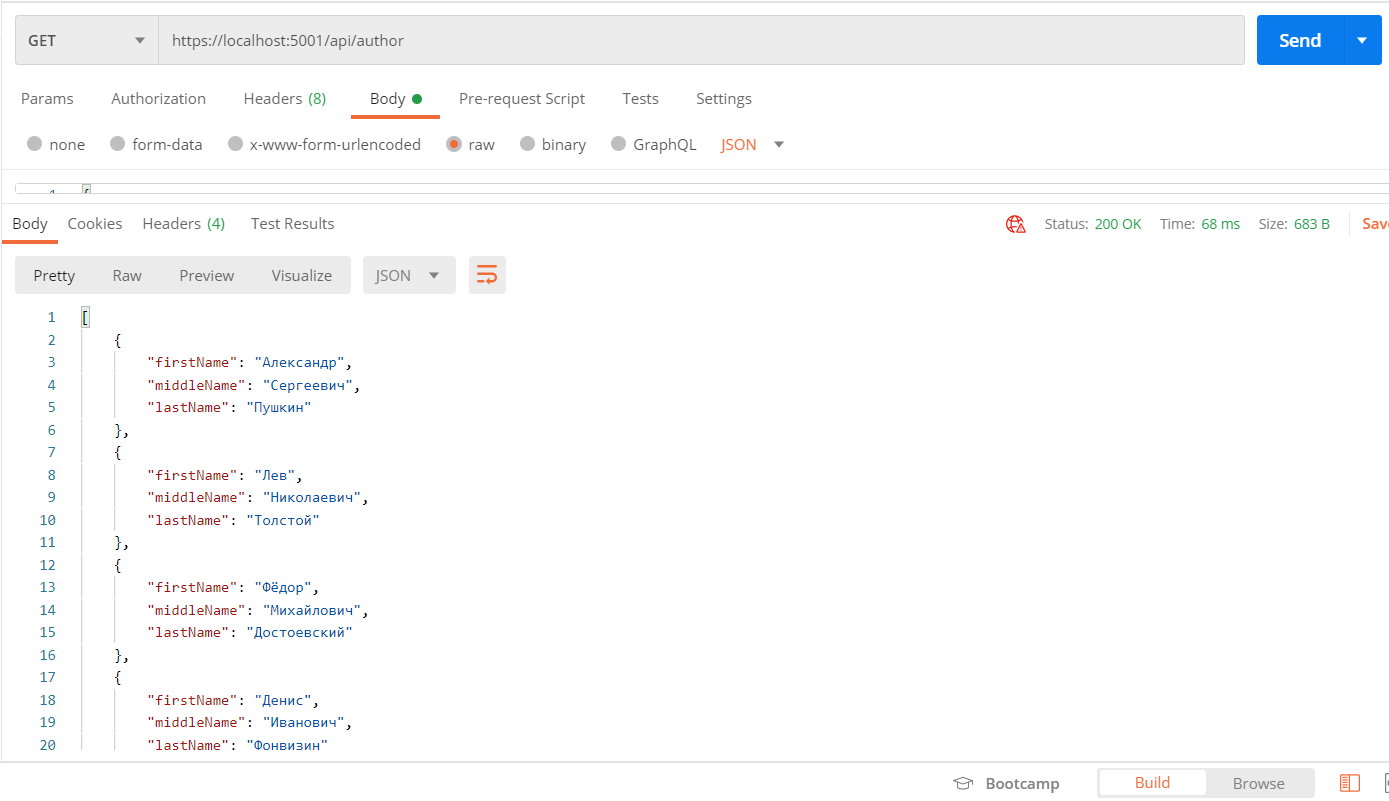


Рисунок 5 – Получение всех авторов из БД

/api/author/{id} - автор определенного id. Результат запроса представлен на Рисунке 6.

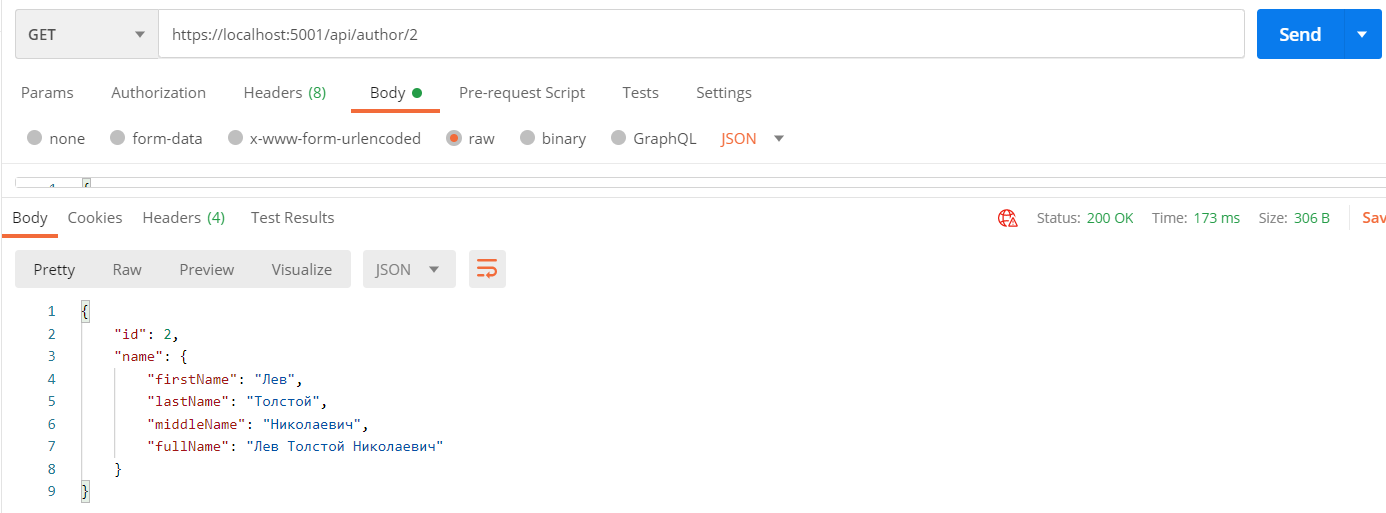


Рисунок 6 - Получение автора с определенным id

/api/book - список всех книг библиотеки. Результат запроса представлен на Рисунках 7-8.

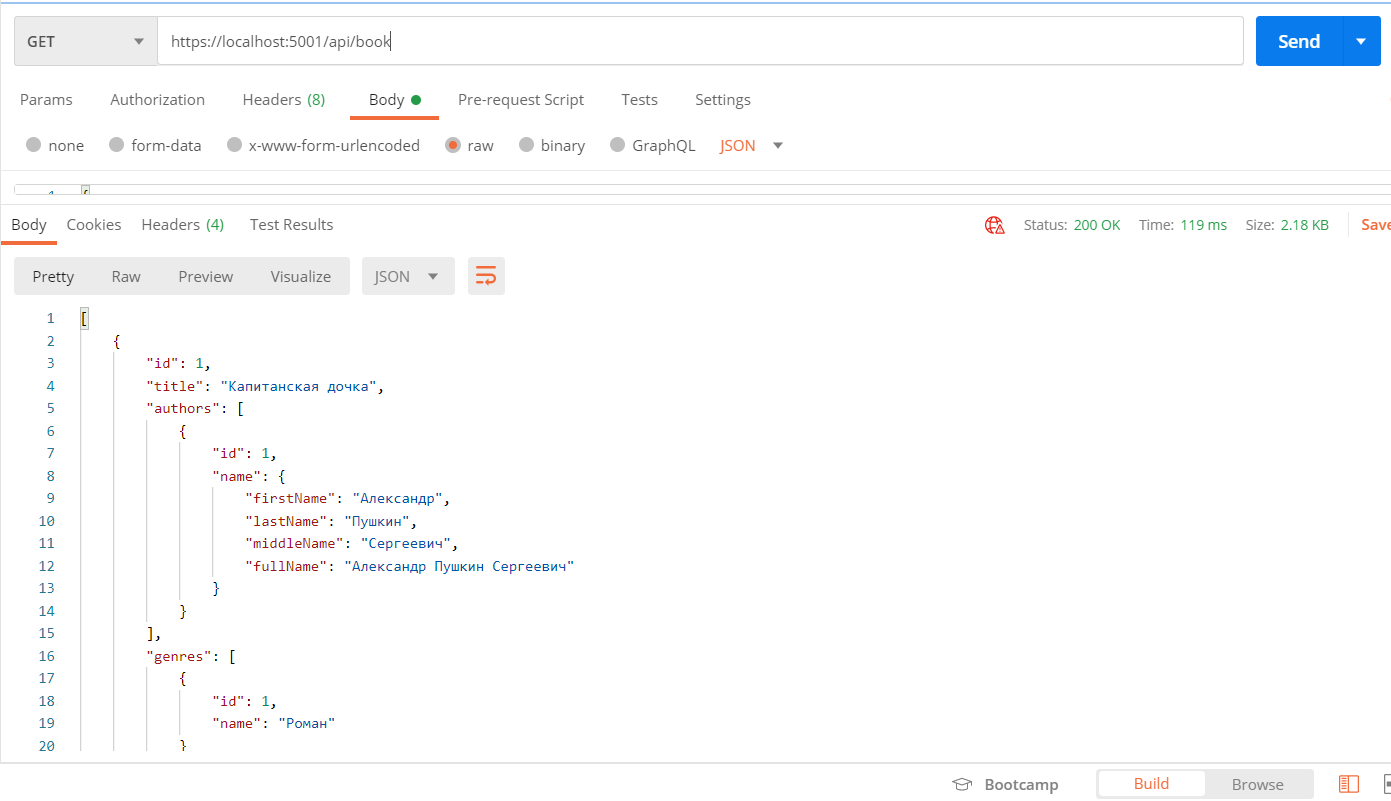


Рисунок 7 - Получение всех книг библиотеки



Рисунок 8 - Получение всех книг библиотеки

/api/book/{id} - книга определенного id. Результат запроса представлен на Рисунке 9.

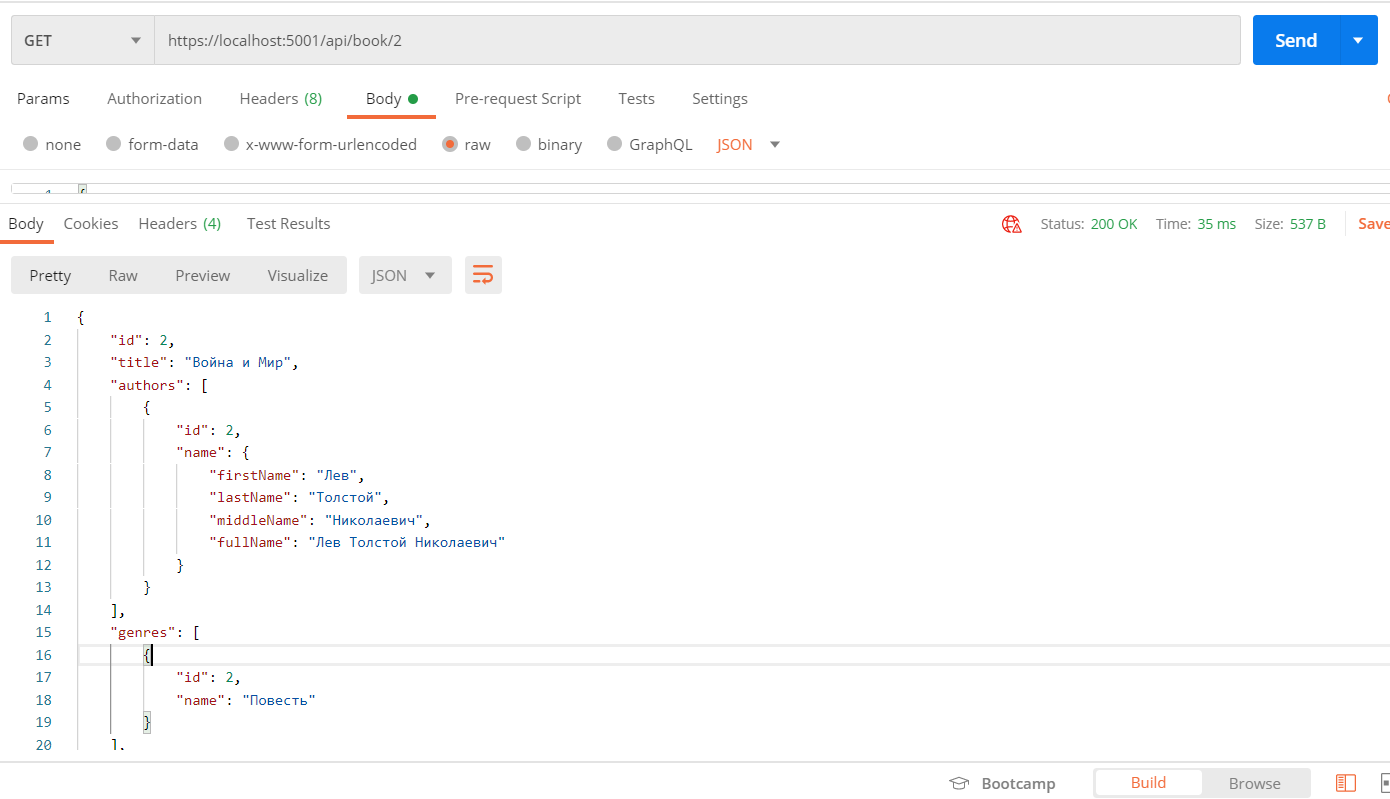


Рисунок 9 - Получение книги по id

/api/book/title/{str} – поиск книги по названию. Результат запроса представлен на Рисунке 10.

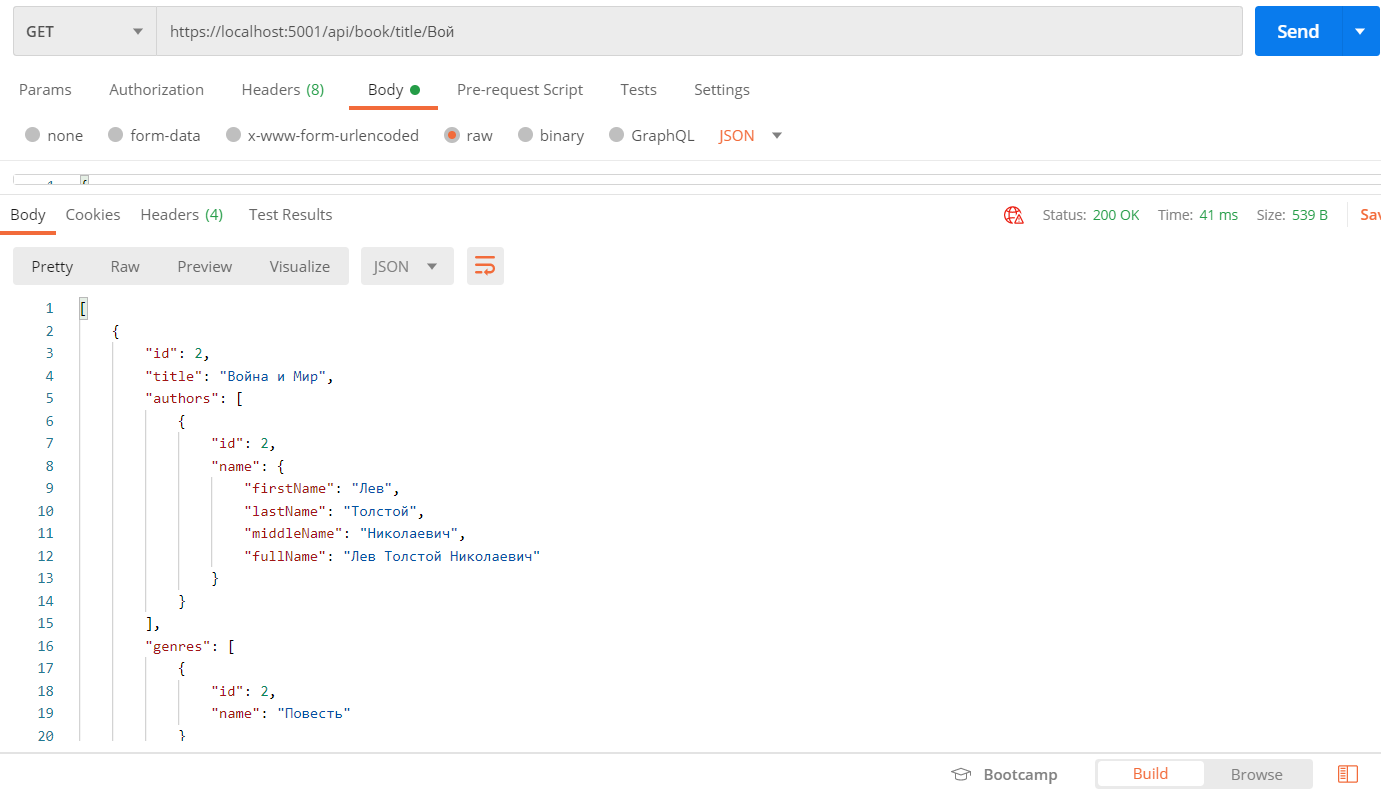


Рисунок 10 - Получение книги по названию

/api/author/for-book/{id} - список авторов книги по её id. Результат запроса представлен на Рисунке 11.

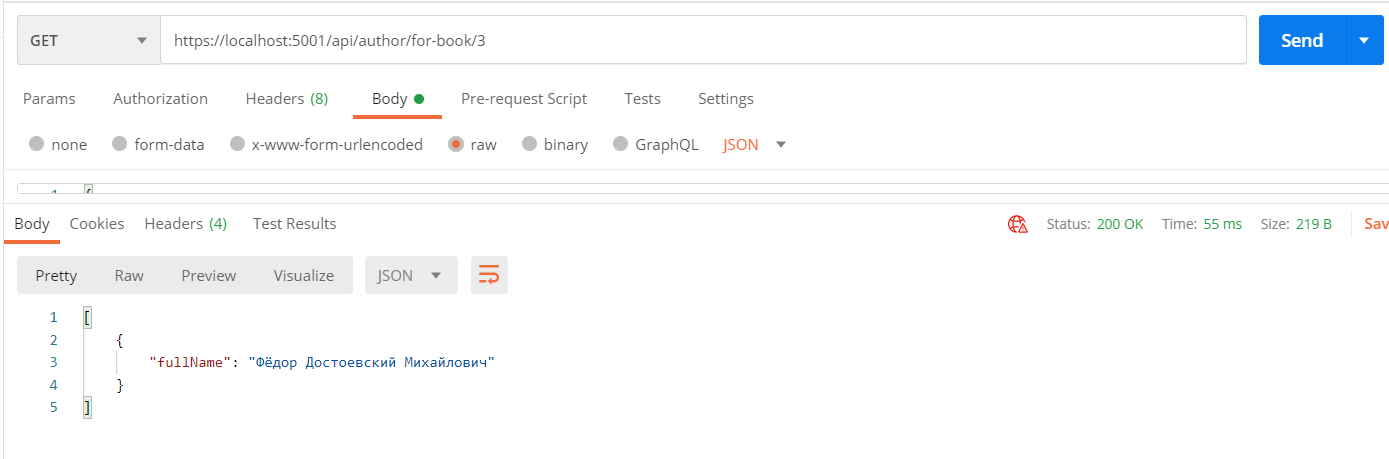


Рисунок 11 - Получение автора по id

* Метод POST:

/api/author - добавление нового автора в БД. Результат запроса представлен на Рисунке 12.

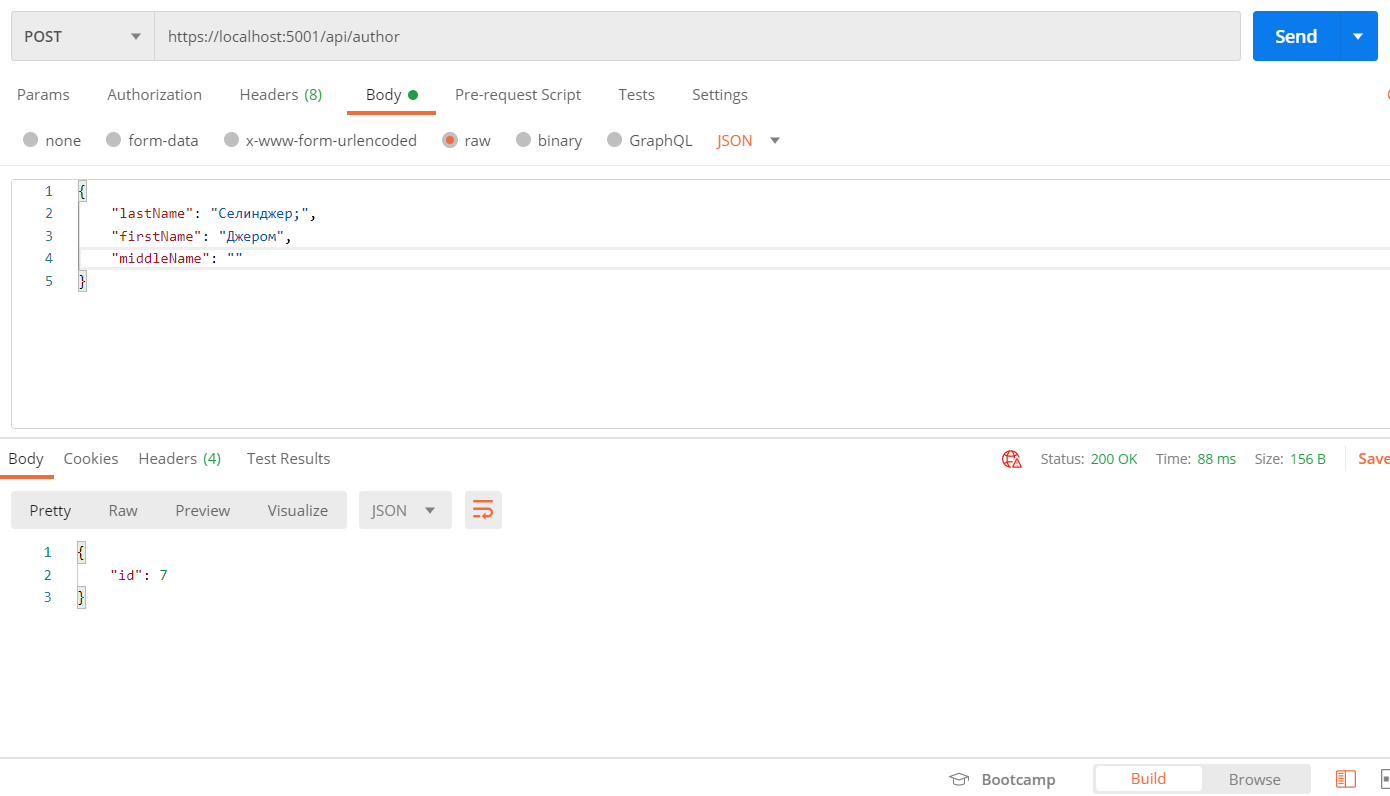


Рисунок 12 - Добавление нового автора в БД

* Метод DELETE

/api/author/{id} - удаление автора по id. Результат запроса представлен на Рисунке 13.

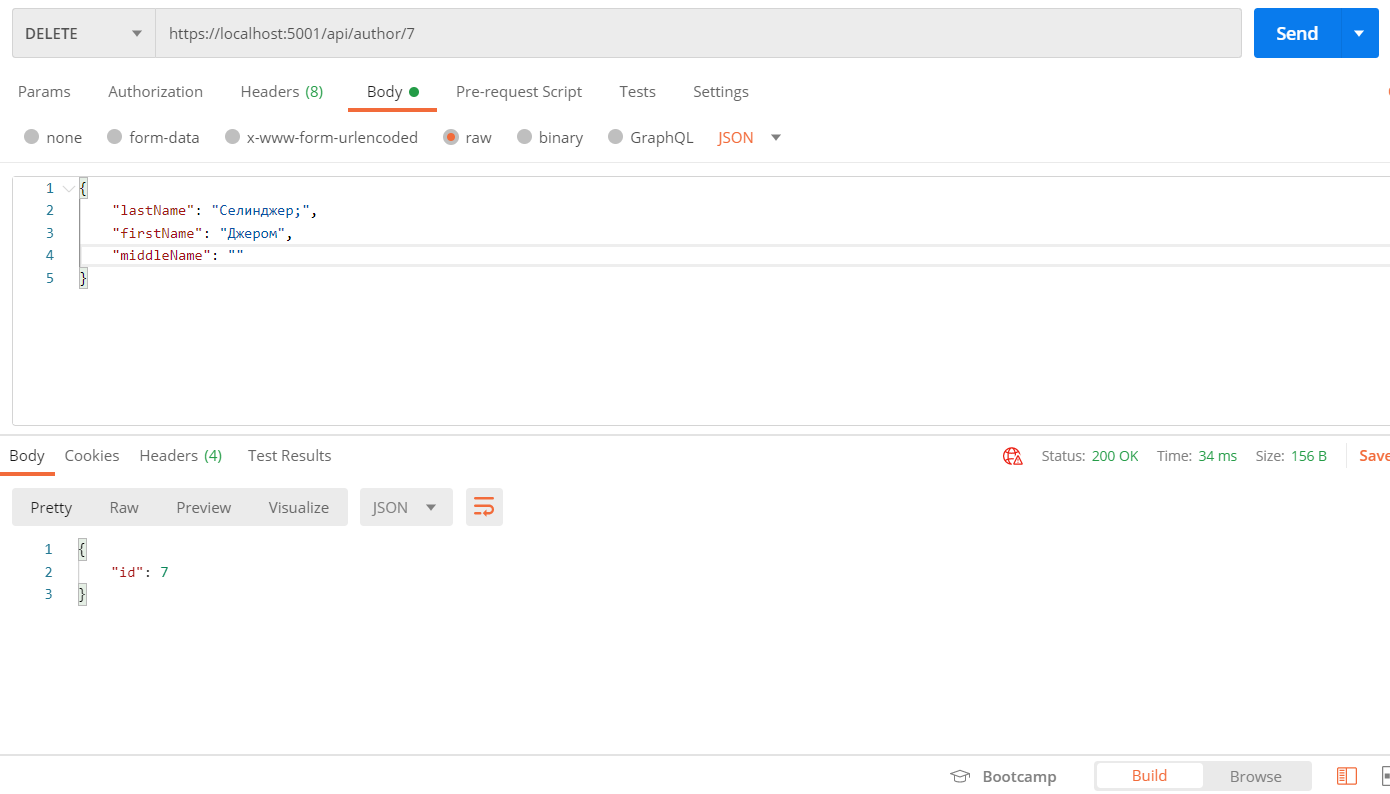


Рисунок 13 - Удаление автора по id

* 1. ПРИНЦИПЫ REST

REST, или Representational State Transfer (англ. «передача состояния представления») — набор принципов, которых рекомендовано придерживаться при создании API. Если API сделан по этим принципам, его называют REST API.

Раньше клиент и сервер были тесно связаны. Например, сервер собирал веб-страницу и отдавал клиенту. Таким образом, серверу нужно было знать, какие технологии работают на стороне клиента, чтобы тот смог открыть вернувшуюся веб-страницу.

Принципы REST позволили отделить клиента от сервера. Благодаря этому:

* стало проще переносить веб-приложение на другие платформы;
* появилась возможность делать открытые API;
* разрабатывать и тестировать серверное ПО стало проще и быстрее.

Поэтому принципы REST важно понимать для программирования сервера.

1. Клиент-сервер

Сервер и клиент отвечают за разные вещи. Ответственность клиента — пользовательский интерфейс, а ответственность сервера — данные. Если API возвращает HTML-страницу, его нельзя назвать REST API: ведь при этом сервер берёт на себя ответственность за интерфейс.

В REST API сервер возвращает данные в каком-то формате — обычно в JSON. Именно этот принцип делает возможным точечную отрисовку страницы и существование единого API для браузера и мобильного приложения.

2. Отсутствие состояния (англ. "Stateless")

Запрос клиента к серверу должен содержать всю информацию, необходимую для обработки этого запроса. В проекте Mesto вы отправляли на сервер токен, чтобы получить карточки. Этот токен сообщал системе, что вы — это вы: запрос содержит информацию о том, ктó запрашивает данные и какие.

Раньше серверы программировали иначе. При аутентификации сервер создавал сессию, то есть запоминал, что пользователь совершил вход. Сейчас такой подход применяется редко.

При отправке токена создавать сессию не нужно. Так что сервер не хранит информацию о состоянии пользователя. Поэтому принцип и называется «отсутствие состояния».

3. Единый интерфейс (англ. "Uniform Interface")

Интерфейс обращения к серверу не зависит от клиента. Он одинаковый для всех.

Таким образом запрос к карточке может быть сформирован из браузера, мобильного приложения и с умного чайника — для всех интерфейс един.

4. Многоуровневость

Первый принцип гласит, что в коммуникации участвуют двое: клиент и сервер. Но мы всё равно можем строить более сложные системы, не нарушая этого принципа. Благодаря многоуровневости.

API сервиса Яндекс.Такси может использовать API Яндекс.Навигатора. Вы как клиент взаимодействуете только с API Яндекс.Такси, а он, в свою очередь, является клиентом навигатора.

5. Кешируемость

Данные ответа могут быть закешированы. Это значит, что мы можем сохранить данные на клиенте и при идентичном запросе взять их из памяти клиента — кеша.

6. Код по запросу (англ. "Code on demand")

Этот принцип необязательный. Он гласит, что функциональность клиента может быть расширена кодом, приходящим с сервера. Сейчас такое можно встретить повсеместно: мы получаем с сервера JS-файлы и исполняем их в браузере. Но принципы формулировались в 2000 году — тогда с сервера код возвращали редко. Потому и выделили это в отдельный пункт.

В REST применяются 5 основных методов:

* GET получает ресурсы;
* POST создаёт ресурс;
* PUT заменяет существующий ресурс целиком;
* PATCH частично изменяет существующий ресурс;
* DELETE удаляет ресурс.

Реже применяют ещё 2 метода:

* HEAD позволяет получить только заголовки ответа. HEAD похож на GET, но у его ответа нет тела;
* OPTIONS — узнать, какие HTTP-методы поддерживает сервер.

Сам по себе метод не определяет логику работы сервера. Можно закодить API так, чтобы POST удалял ресурс, а DELETE — добавлял. Но такой подход противоречил бы заложенной в REST идее, что методы должны применяться по назначению.

Ответ от сервера всегда содержит статус. Правильно выставленный статус делает API более понятным. В этом уроке разберём основные статусы ответов сервера.

Статусы делят на 5 категорий. Запросы каждой из категорий начинаются с разных цифр:

1xx: Informational. Информационный ответ. Такие ответы приходят только техническим системам, вам их отправлять не придётся.

2xx: Success. Запрос прошёл успешно.

3xx Redirection. Запрос не завершён, и клиенту нужно предпринять какие-то действия, чтобы запрос успешно прошёл.

4xx: Client Error. Ошибка на стороне клиента: запрос сформирован неверно или у клиента нет нужных прав доступа.

5xx: Server Error. Ошибка на стороне сервера: что-то сломалось или сервер перегружен.

Самые частые статусы

* 200 OK.

Запрос прошёл успешно. Ответ с таким статусом должен содержать тело. Чаще всего этот статус используют при ответе на GET-запрос ресурса.

* 201 Created.

Ресурс был создан на сервере. Подходящий ответ при создании нового поста в блоге, например.

* 202 Accepted.

Сервер начал работу по удовлетворению запроса, но ещё не закончил. Этот статус применяют для ответа на запросы, обработка которых занимает много времени, например, при обработке большого объёма данных.

* 301 Moved Permanently.

API переделали, а ресурс переместили в другое место. Новый URL указывают в заголовке Location ответа сервера.

* 302 Found.

Запрос должен быть перенаправлен на другой URL. В заголовке Location сервер должен отправить новый URL. При получении запроса браузер автоматически отправит запрос на новый URL.

* 400 Bad Request.

Ошибка на стороне клиента. Например, запрос был неправильно сформирован. Это общий статус: его отправляют в том случае, когда ни один другой 4xx статус не подходит.

* 401 Unauthorized.

Запрос требует авторизации, но соответствующие авторизационные заголовки отсутствуют или сформированы неправильно.

* 403 Forbidden.

Запрос сформирован правильно, но у клиента недостаточно прав, чтобы запрос прошёл успешно. Например, клиент пытается удалить чужой пост.

* 404 Not Found.

Ресурс не найден. Например, пользователя с запрошенным id не существует.

* 405 Method Not Allowed.

Ресурс запроса не поддерживает HTTP-метод, которым сделан запрос.

* 500 Internal Server Error.

Общий статус для ошибок на стороне сервера. Это не ошибка клиента.

* 501 Not Implemented.

Ресурс есть на сервере, но способ обращения к нему пока не реализован.

СПИСОК ЛИТЕРАТУРЫ

J. Dentler, NHibernate 3.0 Cookbook, Birmingham: Packt Publishing Ltd., 2010.

J. Nilsson, «The Cost of GUIDs as Primary Keys,» PEARSON, 8 March 2002. [В Интернете]. Available: http://mng.bz/4q49. [Дата обращения: 22 Июнь 2016].

А. Кондуфоров, «Немного о проектировании: паттерны из мира ORM,» [В Интернете]. Available: http://merle-amber.blogspot.ru/2009/02/orm.html. [Дата обращения: .2016 12 7].

Д. Палермо, «Конфигурационная инфраструктура приложения в NHibernate,» в ASP.NET MVC 4 в Действии, Москва, Manning, 2012.

М. Фаулер, Шаблоны корпоративных приложений, Москва: Вильямс, 2010.

В. Вернон, Реализация методов предметно-ориентированного проектирования, Москва: Вильямс, 2016.

Э. Троелсен и Ф. Джепикс, Язык программирования C# 6.0 и платформа .NET 4.6, Москва: Издательсткй дом "Вильямс", 2016.

ПРИЛОЖЕНИЕ

1. Infrastructure:
2. MyForeignKeyConvention.cs:

namespace Infrastructure.Conventions

{

using System;

using FluentNHibernate;

using FluentNHibernate.Conventions;

/// <summary>

/// Собственная настройка договорённости (конвенции) о формате вторичных (внешних) ключей.

/// </summary>

internal class MyForeignKeyConvention : ForeignKeyConvention

{

/// <summary>

/// Метод, получающий имя вторичного (внешнего) ключа по имения свойства или имени типа свойства.

/// </summary>

/// <param name="property"> Свойство. </param>

/// <param name="type"> Тип свойства. </param>

/// <returns> Имя вторичного (внешнего) ключа. </returns>

protected override string GetKeyName(Member property, Type type) => $"ID\_{property?.Name ?? type.Name}s";

}

}

1. MyIdConvention.cs:

namespace Infrastructure.Conventions

{

using FluentNHibernate.Conventions;

using FluentNHibernate.Conventions.Instances;

/// <summary>

/// Собственная настройка договорённости (конвенции) о формате первичных ключей.

/// </summary>

internal class MyIdConvention : IIdConvention

{

/// <summary>

/// Метод, применяющий правила порождения значения ключа на стороне приложения.

/// </summary>

/// <param name="instance"> Объект типа. </param>

public void Apply(IIdentityInstance instance) => instance.GeneratedBy.Increment();

}

}

1. MyManyToManyTableNameConvention.cs:

using System;

namespace Infrastructure.Conventions

{

using FluentNHibernate.Conventions;

using FluentNHibernate.Conventions.Inspections;

/// <summary>

/// Собственная конвенция по наименованию промежуточной сущности в связи многие-ко-многим.

/// </summary>

internal class MyManyToManyTableNameConvention : ManyToManyTableNameConvention

{

/// <inheritdoc />

protected override string GetBiDirectionalTableName(

IManyToManyCollectionInspector collection,

IManyToManyCollectionInspector otherSide)

{

Console.WriteLine($"{collection.EntityType.Name}\_{otherSide.EntityType.Name}");

return $"{collection.EntityType.Name}\_{otherSide.EntityType.Name}";

}

/// <inheritdoc />

protected override string GetUniDirectionalTableName(IManyToManyCollectionInspector collection)

{

return $"{collection.ChildType.Name}\_{collection.OtherSide.ChildType.Name}";

}

}

}

1. StringExtentions.cs:

namespace Infrastructure.Extensions

{

using System.Collections.Generic;

public static class StringExtentions

{

/// <summary>

/// Метод, проверяющий входной параметр на null или пустое значение.

/// </summary>

/// <param name="str">Входная строка</param>

public static bool IsNullOrEmpty(this string str) => string.IsNullOrEmpty(str);

/// <summary>

/// Метод, проверяющий входной параметр на null или пустое значение. Удаляет пробелы.

/// </summary>

/// <param name="str">Входная строка</param>

/// <returns></returns>

public static bool IsNullOrWhiteSpace(this string str) => string.IsNullOrEmpty(str?.Trim());

/// <summary>

/// Метод, возвращающий строку null или str. Проверка IsNullOrEmpty()

/// </summary>

/// <param name="str">Входная строка</param>

/// <returns></returns>

public static string NullIfNullOrEmpty(this string str) => str.IsNullOrEmpty() ? null : str;

/// <summary>

/// Метод, возвращающий строку null или str. Проверка IsNullOrWhitespace()

/// </summary>

/// <param name="str">Входная строка</param>

/// <returns></returns>

public static string NullIfNullOrWhiteSpace(this string str) => str.IsNullOrWhiteSpace() ? null : str;

/// <summary>

/// Метод, проверяющий на пустое значение. Удаляет пробелы у строки.

/// Сделано для MiddleName, так как данный параметр может иметь пустую строку, а два других нет.

/// </summary>

/// <param name="str"></param>

/// <returns></returns>

public static string NullIfNullOrWhitespaceTrim(this string str) =>

str.IsNullOrWhiteSpace() ? null : str.Trim();

/// <summary>

/// Метод формирующий строку из коллекции.

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="source">Коллекция</param>

/// <param name="separator">Разделитель</param>

/// <param name="defaultResult">Значение по умолчанию</param>

/// <returns></returns>

public static string Join<T>(this IEnumerable<T> source, string separator = ", ", string defaultResult = "")

{

return source is null ? defaultResult : string.Join(separator, source);

}

}

}

1. NHibernateConfigurator.cs

namespace Infrastructure.Helpers

{

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using Infrastructure.Conventions;

using NHibernate;

public class NHibernateConfigurator

{

public static FluentConfiguration config;

public static string DataSourceLocation = @"DESKTOP-V49F727\SQLEXPRESS";

//@"DESKTOP-9RIQ0RR\SQLEXPRESS";

public static string GetConnectionString() => @"Data Source=DESKTOP-V49F727\SQLEXPRESS;Initial Catalog=DBLibraryHome;Integrated Security=True";

public static FluentConfiguration GetConfiguration(Assembly assembly, bool showSQL = false)

{

var configuration = MsSqlConfiguration.MsSql2012.ConnectionString(GetConnectionString());

#if DEBAG

configuration = configuration.ShowSql().FormatSql();

#endif

if (showSQL)

{

configuration = configuration.ShowSql().FormatSql();

}

return config = Fluently.Configure()

.Database(configuration)

.Mappings(m => m.FluentMappings.AddFromAssembly(assembly)

.Conventions.Add<MyIdConvention>()

.Conventions.Add<MyForeignKeyConvention>()

.Conventions.Add<MyManyToManyTableNameConvention>());

}

public static ISessionFactory GetSessionFactory() => config?.BuildSessionFactory();

}

}

1. NHibernateTestHelper.cs

namespace Infrastructure.Helpers

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using FluentNHibernate.Cfg;

using FluentNHibernate.Cfg.Db;

using NHibernate;

using NHibernate.Cfg;

using NHibernate.Tool.hbm2ddl;

public static class NHibernateTestHelper

{

private static Configuration savedConfiguration = null;

public static ISession GetSession(bool showSql, params Type[] mappingTypes)

{

return CreateConfiguration(showSql, Register(mappingTypes)).CreateSession();

}

public static ISession GetSession(bool showSql, params Assembly[] assemblies)

{

return CreateConfiguration(showSql, Register(assemblies)).CreateSession();

}

public static ISession GetSession(bool showSql, Action<FluentMappingsContainer> registrar)

{

return CreateConfiguration(showSql, Register(registrar)).CreateSession();

}

private static Action<FluentConfiguration> Register(IEnumerable<Type> types)

{

return cfg => cfg.Mappings(x => x.FluentMappings.AddRange(types));

}

private static Action<FluentConfiguration> Register(IEnumerable<Assembly> assemblies)

{

return cfg => cfg.Mappings(x => x.FluentMappings.AddFromAssemblyRange(assemblies));

}

private static Action<FluentConfiguration> Register(Action<FluentMappingsContainer> registrar)

{

return cfg => cfg.Mappings(x => registrar?.Invoke(x.FluentMappings));

}

private static FluentConfiguration CreateConfiguration(bool showSql, Action<FluentConfiguration> registerAction)

{

var sqliteConfiguration = SQLiteConfiguration.Standard.InMemory();

if (showSql)

{

sqliteConfiguration = sqliteConfiguration.ShowSql().FormatSql();

}

var cfg = Fluently.Configure().Database(sqliteConfiguration);

registerAction?.Invoke(cfg);

return cfg.ExposeConfiguration(configuration => savedConfiguration = configuration);

}

private static ISession CreateSession(this FluentConfiguration configuration)

{

var session = configuration.BuildSessionFactory().OpenSession();

new SchemaExport(savedConfiguration ?? configuration.BuildConfiguration())

.Execute(true, true, false, session.Connection, null);

return session;

}

}

internal static class FluentMappingsContainerExtensions

{

public static void AddRange(this FluentMappingsContainer container, IEnumerable<Type> types)

{

foreach (var type in types ?? Enumerable.Empty<Type>())

{

container?.Add(type);

}

}

public static void AddFromAssemblyRange(this FluentMappingsContainer container, IEnumerable<Assembly> assemblies)

{

foreach (var assembly in assemblies ?? Enumerable.Empty<Assembly>())

{

container?.AddFromAssembly(assembly);

}

}

}

internal static class FluentConfigurationExtensions

{

public static FluentConfiguration RegisterMappingTypes(this FluentConfiguration cfg, Type[] mappingTypes)

{

return cfg.Mappings(x => x.FluentMappings.AddRange(mappingTypes));

}

}

}

1. Library.Domain:
2. Author.cs:

namespace Library.Domain

{

using System;

using System.Collections.Generic;

using System.Text.Json.Serialization;

public class Author

{

public virtual int ID { get; protected set; }

public virtual Name Name { get; protected set; }

[JsonIgnore]

public virtual ISet<Book> Books { get; protected set; } = new HashSet<Book>();

public override string ToString() => $"{this.Name}";

[Obsolete("Конструктор только для ORM", true)]

protected Author() { }

public Author(string lastName, string firstName, string middleName)

{

this.Name = new Name(firstName, lastName, middleName);

}

public Author(Name name)

{

this.Name = name;

}

}

}

1. Book.cs:

namespace Library.Domain

{

using System.Collections.Generic;

using Infrastructure.Extensions;

public class Book

{

protected Book() { }

public Book(string title)

{

this.Title = title;

}

public virtual int Id { get; protected set; }

public virtual string Title { get; protected set; }

public virtual ISet<Author> Authors { get; protected set; } = new HashSet<Author>();

public virtual ISet<Genre> Genres { get; protected set; } = new HashSet<Genre>();

public virtual ISet<Publisher> Publishers { get; protected set; } = new HashSet<Publisher>();

public virtual Shelf Shelf { get; protected set; }

public override string ToString()

{

return $"{this.Title} -- {this.Authors.Join()} -- {this.Genres} -- {this.Publishers}";

}

}

}

1. Genre.cs

namespace Library.Domain

{

using System.Collections.Generic;

using System.Text.Json.Serialization;

public class Genre

{

public Genre() { }

public Genre(string name)

{

this.Name = name;

}

public virtual int Id { get; protected set; }

public virtual string Name { get; protected set; }

[JsonIgnore]

public virtual ISet<Book> Books { get; protected set; } = new HashSet<Book>();

public override string ToString() => this.Name;

}

}

1. Name.cs

namespace Library.Domain

{

using System;

using Infrastructure.Extensions;

public class Name

{

public string FirstName { get; }

public string LastName { get; }

public string MiddleName { get; }

protected Name() { }

public Name(string firstName, string lastName, string middleName)

{

this.FirstName = firstName.NullIfNullOrWhiteSpace() ??

throw new ArgumentOutOfRangeException(nameof(firstName));

this.LastName = lastName.NullIfNullOrWhiteSpace() ??

throw new ArgumentOutOfRangeException(nameof(lastName));

if (middleName == null && string.IsNullOrWhiteSpace(middleName))

{

throw new ArgumentOutOfRangeException(nameof(middleName));

}

this.MiddleName = middleName.NullIfNullOrWhitespaceTrim();

}

public override int GetHashCode()

{

return HashCode.Combine(this.FirstName, this.LastName, this.MiddleName ?? string.Empty);

}

public override bool Equals(object obj)

{

if (object.ReferenceEquals(obj, null))

{

return false;

}

if (object.ReferenceEquals(obj, this))

{

return true;

}

return obj is Name other

&& string.Equals(this.FirstName, other.FirstName, StringComparison.InvariantCulture)

&& string.Equals(this.LastName, other.LastName, StringComparison.InvariantCulture)

&& string.Equals(this.MiddleName, other.MiddleName, StringComparison.InvariantCulture);

}

public override string ToString() => FullName;

public virtual string FullName => $"{this.FirstName} {this.LastName} {this.MiddleName}".Trim();

}

}

1. Publisher.cs:

namespace Library.Domain

{

using System.Collections.Generic;

using System.Text.Json.Serialization;

public class Publisher

{

public Publisher() { }

public Publisher(string name)

{

this.Name = name;

}

public virtual int Id { get; protected set; }

public virtual string Name { get; protected set; }

[JsonIgnore]

public virtual ISet<Book> Books { get; protected set; } = new HashSet<Book>();

public override string ToString() => this.Name;

}

}

1. Room.cs

namespace Library.Domain

{

using System.Collections.Generic;

using System.Text.Json.Serialization;

public class Room

{

public Room() { }

public Room(string name)

{

this.Name = name;

}

public virtual int Id { get; protected set; }

public virtual string Name { get; protected set; }

[JsonIgnore]

public virtual ISet<Shelf> Shelves { get; protected set; } = new HashSet<Shelf>();

public override string ToString() => this.Name;

}

}

1. Shelf.cs

namespace Library.Domain

{

using System.Collections.Generic;

using System.Text.Json.Serialization;

public class Shelf

{

public Shelf() { }

public Shelf(string description)

{

this.Description = description;

}

public virtual int Id { get; protected set; }

public virtual string Description { get; protected set; }

public virtual Room Room { get; protected set; }

[JsonIgnore]

public virtual ISet<Book> Books { get; protected set; } = new HashSet<Book>();

public override string ToString()

{

return this.Description;

}

}

}

1. Library.NH:
2. AuthorMap:

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class AuthorMap : ClassMap<Author>

{

public AuthorMap()

{

this.Table("Authors");

this.Id(x => x.ID).GeneratedBy.Increment();

this.Component(x => x.Name);

this.HasManyToMany(x => x.Books).Inverse();

}

}

}

1. BookMap.cs:

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class BookMap : ClassMap<Book>

{

public BookMap()

{

this.Table("Books");

this.Id(x => x.Id).GeneratedBy.Increment();

this.Map(x => x.Title);

this.HasManyToMany(x => x.Authors);

this.HasManyToMany(x => x.Publishers);

this.HasManyToMany(x => x.Genres);

this.References(x => x.Shelf, "ID\_Shelf");

}

}

}

1. GenreMap.cs

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class GenreMap : ClassMap<Genre>

{

public GenreMap()

{

this.Table("Genres");

this.Id(x => x.Id);

this.Map(x => x.Name);

this.HasManyToMany(x => x.Books);

}

}

}

1. NameMap.cs

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class NameMap : ComponentMap<Name>

{

public NameMap()

{

this.Map(x => x.FirstName);

this.Map(x => x.LastName);

this.Map(x => x.MiddleName);

}

}

}

1. PublisherMap.cs

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class PublisherMap : ClassMap<Publisher>

{

public PublisherMap()

{

this.Table("Publishers");

this.Id(x => x.Id);

this.Map(x => x.Name);

this.HasManyToMany(x => x.Books);

}

}

}

1. RoomMap.cs

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class RoomMap : ClassMap<Room>

{

public RoomMap()

{

this.Table("Rooms");

this.Id(x => x.Id);

this.Map(x => x.Name);

this.HasMany(x => x.Shelves);

}

}

}

1. ShelfMap.cs

namespace Library.NH.Maps

{

using FluentNHibernate.Mapping;

using Library.Domain;

public class ShelfMap : ClassMap<Shelf>

{

public ShelfMap()

{

this.Table("Shelves");

this.Id(x => x.Id);

this.Map(x => x.Description);

this.HasMany(x => x.Books);

this.References(x => x.Room, "ID\_Room");

}

}

}

1. AuthorRepository.cs

namespace Library.NH.Repositories

{

using System;

using System.Linq;

using System.Linq.Expressions;

using Library.Domain;

using NHibernate;

public class AuthorRepository : IAuthorRepository

{

private readonly ISessionFactory sessionFactory;

private readonly ISession session;

public AuthorRepository(ISessionFactory sessionFactory)

{

var factory = sessionFactory ?? throw new ArgumentNullException(nameof(sessionFactory));

this.session = factory.OpenSession();

}

public IQueryable<Author> GetAll()

{

return this.session.Query<Author>();

}

public IQueryable<Author> Filter(Expression<Func<Author, bool>> filter)

{

if (filter == null)

{

throw new ArgumentNullException(nameof(filter));

}

//return this.GetAll().Where(x => filter(x));

return this.GetAll().Where(filter);

}

public Author Get(int id)

{

return this.GetAll().SingleOrDefault(x => x.ID == id);

}

public bool TryGet(int id, out Author author)

{

author = this.GetAll().SingleOrDefault(t => t.ID == id);

return author != null;

//return (author = this.Get(id)) != null;

}

public Author Create(Author author)

{

var id = (int)this.session.Save(author);

this.session.Flush();

return author;

}

public void Delete(int id)

{

if (!this.TryGet(id, out var author)) return;

this.session.Delete(author);

this.session.Flush();

}

}

}

1. BookRepository.cs

namespace Library.NH.Repositories

{

using System;

using System.Linq;

using Library.Domain;

using NHibernate;

public class BookRepository : IBookRepository

{

private readonly ISession session;

public BookRepository(ISession session)

{

this.session = session ?? throw new ArgumentNullException(nameof(session));

}

public IQueryable<Book> GetAll()

{

return this.session.Query<Book>();

}

public Book GetBooksForAuthorID(int id)

{

return this.GetAll().SingleOrDefault(x => x.Id == id);

}

public Book GetId(int id)

{

return this.GetAll().SingleOrDefault(x => x.Id == id);

}

public IQueryable<Book> GetBooksByTitle()

{

return this.GetAll();

}

}

}

1. IAuthorRepository.cs

namespace Library.NH.Repositories

{

using System;

using System.Linq;

using System.Linq.Expressions;

using Library.Domain;

public interface IAuthorRepository

{

IQueryable<Author> GetAll();

IQueryable<Author> Filter(Expression<Func<Author, bool>> filter);

Author Get(int id);

bool TryGet(int id, out Author author);

Author Create(Author author);

void Delete(int id);

}

}

1. IBookRepository.cs

namespace Library.NH.Repositories

{

using System.Linq;

using Library.Domain;

public interface IBookRepository

{

IQueryable<Book> GetAll();

Book GetBooksForAuthorID(int id);

Book GetId(int id);

IQueryable<Book> GetBooksByTitle();

}

}

1. LibraryNHibernateConfigurator.cs

namespace Library.NH

{

using System;

using System.Reflection;

public class LibraryNHibernateConfigurator

{

/// <summary>

/// Метод получения сборки.

/// </summary>

/// <returns>Исполняемая сборка.</returns>

[Obsolete("Стоит переписать на расширение с учётом регистрации правил отображения и конвенций.")]

public static Assembly GetAssembly() => Assembly.GetExecutingAssembly();

}

}

1. Library.Services:
2. AuthorService.cs:

namespace Library.Services

{

using Library.Domain;

using Library.NH.Repositories;

using System;

using System.Linq;

public class AuthorService : IAuthorService

{

private readonly IAuthorRepository authorRepository;

private readonly IBookService bookService;

public AuthorService(IAuthorRepository authorRepository, IBookService bookService)

{

this.authorRepository = authorRepository ?? throw new ArgumentNullException(nameof(authorRepository));

this.bookService = bookService ?? throw new ArgumentNullException(nameof(bookService));

}

public IQueryable<Author> GetAll()

{

return this.authorRepository.GetAll();

}

public Author GetId(int id)

{

return this.authorRepository.Get(id);

}

public IQueryable<Author> GetAuthorByBookId(int bookId)

{

if (!this.bookService.TryGet(bookId, out var targetBook))

{

//return this.publisherRepository.Filter(t => t.Books.Contains(author.)).ToList();

return Enumerable.Empty<Author>().AsQueryable();

}

return authorRepository.Filter(t => t.Books.Contains(targetBook));

}

public bool TryGet(int id, out Author author)

{

throw new **NotImplementedException**();

}

public Author Create(string lastName, string firstName, string middleName)

{

Author author = null;

try

{

author = new Author(lastName, firstName, middleName);

return this.authorRepository.Create(author);

}

catch (Exception e)

{

return author;

}

}

public void Delete(int id)

{

this.authorRepository.Delete(id);

}

}

}

1. BookService.cs:

namespace Library.Services

{

using System;

using System.Collections.Generic;

using System.Linq;

using Library.Domain;

using Library.NH.Repositories;

public class BookService : IBookService

{

private readonly IBookRepository bookRepository;

public BookService(IBookRepository bookRepository)

{

this.bookRepository = bookRepository ?? throw new ArgumentNullException(nameof(bookRepository));

}

public IList<Book> GetAll()

{

return this.bookRepository.GetAll().ToList();

}

public bool TryGet(int id, out Book book)

{

book = this.GetAll().SingleOrDefault(g => g.Id == id);

return book != null;

}

public Book GetBooksForAuthorID(int id)

{

return this.bookRepository.GetBooksForAuthorID(id);

}

public Book GetBookId(int id)

{

if (this.TryGet(id, out Book book)) return this.bookRepository.GetId(id);

throw new ArgumentNullException(nameof(id));

}

public IQueryable<Book> GetBooksByTitle(string str)

{

return this.bookRepository.GetBooksByTitle().Where(x => x.Title.Contains(str));

}

}

}

1. IAuthorService.cs:

namespace Library.Services

{

using System.Linq;

using Library.Domain;

public interface IAuthorService

{

bool TryGet(int id, out Author author);

IQueryable<Author> GetAll();

Author GetId(int id);

IQueryable<Author> GetAuthorByBookId(int bookId);

Author Create(string lastName, string firstName, string middleName);

void Delete(int id);

}

}

1. IBookService.cs:

using System.Linq;

namespace Library.Services

{

using System.Collections.Generic;

using Library.Domain;

public interface IBookService

{

IList<Book> GetAll();

bool TryGet(int id, out Book book);

Book GetBooksForAuthorID(int id);

Book GetBookId(int id);

IQueryable<Book> GetBooksByTitle(string str);

}

}

1. Library.WebServices:
2. AuthorController:

namespace Library.WebServices.Controllers

{

using System;

using Library.Domain;

using Microsoft.AspNetCore.Mvc;

using Library.Services;

using System.Linq;

using System.Collections.Generic;

using AutoMapper;

using Library.WebServices.ViewModels;

[Route("api/[controller]")]

[ApiController]

public class AuthorController : Controller

{

private readonly IAuthorService authorService;

private readonly IMapper mapper;

public AuthorController(IAuthorService authorService, IMapper mapper)

{

this.authorService = authorService ?? throw new ArgumentNullException(nameof(authorService));

this.mapper = mapper ?? throw new ArgumentNullException(nameof(mapper));

}

[HttpGet("for-book/{bookId}")]

public IEnumerable<AuthorViewModel> Filter([FromRoute] int bookId)

{

return this.authorService.GetAuthorByBookId(bookId).Select(t => new AuthorViewModel(t.Name));

}

[HttpGet]

public IActionResult GetAll()

{

var result = this.authorService.GetAll().AsEnumerable().Select(x => this.mapper.Map<Author, AuthorDTO>(x));

//this.mapper.Map<IEnumerable<Author>, IEnumerable<AuthorDTO>>(this.authorService.GetAll());

return this.Ok(result);

}

[HttpGet("{authorId}")]

public Author GetId([FromRoute] int authorId)

{

return this.authorService.GetId(authorId);

}

[HttpPost]

public IActionResult Create([FromBody] AuthorDTO dto)

{

if (dto is null)

{

return this.Ok(new {Error = $"{nameof(dto)} is null"});

}

var author = this.authorService.Create(dto.LastName, dto.FirstName, dto.MiddleName);

if (author is null)

{

return this.Ok(new {Error = $"{nameof(author)} is null"});

}

return this.Ok(new {Id = author.ID});

}

[HttpDelete("{id}")]

public IActionResult Delete([FromRoute] int id)

{

this.authorService.Delete(id);

return this.Ok(new {Id = id});

}

}

}

1. BookController.cs:

using System.Collections.Generic;

namespace Library.WebServices.Controllers

{

using Microsoft.AspNetCore.Mvc;

using System;

using System.Linq;

using AutoMapper;

using Library.Domain;

using Library.Services;

using Library.WebServices.ViewModels;

[Route("api/[controller]")]

[ApiController]

public class BookController : ControllerBase

{

private readonly IBookService bookService;

private readonly IMapper mapper;

public BookController(IBookService bookService, IMapper mapper)

{

this.bookService = bookService ?? throw new ArgumentNullException(nameof(bookService));

this.mapper = mapper ?? throw new ArgumentNullException(nameof(mapper));

}

[HttpGet]

public IActionResult GetAll()

{

var result = this.bookService.GetAll()

.AsEnumerable()

.Select(x => this.mapper.Map<Book, BookDTO>(x));

return this.Ok(result);

}

[HttpGet("{Id}")]

public Book GetBookId(int id)

{

return this.bookService.GetBookId(id);

}

[HttpGet("title/{str}")]

public IActionResult GetBookByTitleName(string str)

{

var books = this.bookService.GetBooksByTitle(str);

if (books == null) return this.NotFound();

return this.Ok(books);

}

}

}

1. ServiceCollectionExtension.cs:

namespace Library.WebServices.Controllers

{

using AutoMapper;

using Microsoft.Extensions.DependencyInjection;

public static class ServiceCollectionExtension

{

public static IServiceCollection AddMapper(this IServiceCollection services)

{

services.AddScoped<MapConfigurator>();

services.AddScoped<IMapper>(sp => sp.GetService<MapConfigurator>().CreateMapper());

return services;

}

}

}

1. AuthorDTO.cs:

namespace Library.WebServices.ViewModels

{

public class AuthorDTO

{

public string FirstName { get; set; }

public string MiddleName { get; set; }

public string LastName { get; set; }

}

}

1. AuthorViewModel.cs:

namespace Library.WebServices.ViewModels

{

using System;

using Library.Domain;

public class AuthorViewModel

{

public string FullName { get; }

public AuthorViewModel(Name name)

{

if (name == null)

throw new ArgumentNullException(nameof(name));

this.FullName = name.FullName;

}

}

}

1. BookDTO.cs:

namespace Library.WebServices.ViewModels

{

using System.Collections.Generic;

using Library.Domain;

public class BookDTO

{

public int Id { get; set; }

public string Title { get; set; }

public IEnumerable<Author> Authors { get; set; }

public IEnumerable<Genre> Genres { get; set; }

public IEnumerable<Publisher> Publishers { get; set; }

public Shelf Shelf { get; set; }

}

}

1. BookViewModel.cs

namespace Library.WebServices.ViewModels

{

using System;

using Library.Domain;

public class BookViewModel

{

public string Title { get; }

public Author Author { get; }

public Genre Genre { get; }

public Publisher Publisher { get; }

public BookViewModel(string title, Author author, Genre genre, Publisher publisher)

{

if (title == null || author == null || genre == null || publisher == null || genre == null) throw new ArgumentNullException($"Имя объекта {nameof(title)} или сущность автора равна null");

this.Title = title;

this.Author = author;

this.Genre = genre;

this.Publisher = publisher;

}

}

}

1. Appsettings.json:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"embedded": "Data Source=DESKTOP-V49F727\\SQLEXPRESS;Initial Catalog=DBLibraryHome;Integrated Security=True",

"relative": "Data Source=DESKTOP-V49F727\\SQLEXPRESS;Initial Catalog=DBLibraryHome;Integrated Security=True"

}

}

1. MapConfigurator.cs:

namespace Library.WebServices

{

using System;

using AutoMapper;

using AutoMapper.Configuration;

using Library.Domain;

using Library.WebServices.ViewModels;

internal class MapConfigurator

{

public IMapper CreateMapper()

{

return this.GetConfiguration().CreateMapper();

}

internal MapperConfiguration GetConfiguration()

{

var configuration = new MapperConfigurationExpression();

this.CreateMaps(configuration);

return new MapperConfiguration(configuration);

}

internal void CreateMaps(MapperConfigurationExpression expression)

{

expression.CreateMapFromAuthorToAuthorDto();

expression.CreateMapFromAuthorToAuthorViewModel();

expression.CreateMapFromBookToBookDto();

expression.CreateMapFromBookToBookViewModel();

}

}

internal static class MappingConfigurationExpressionExtension

{

public static void CreateMapFromAuthorToAuthorDto(this IProfileExpression expression)

{

expression.CreateMap<Author, AuthorDTO>()

.ForMember(

d => d.FirstName,

opt => opt.MapFrom(s => GetValueSafety(s.Name, n => n.FirstName)))

.ForMember(

d => d.MiddleName,

opt => opt.MapFrom(s => GetValueSafety(s.Name, n => n.MiddleName)))

.ForMember(

d => d.LastName,

opt => opt.MapFrom(s => GetValueSafety(s.Name, n => n.LastName)));

}

public static void CreateMapFromBookToBookDto(this IProfileExpression expression)

{

expression.CreateMap<Book, BookDTO>()

.ForMember(d => d.Title, opt => opt.MapFrom(s => s.Title))

.ForMember(d => d.Id, opt => opt.MapFrom(src => src.Id))

.ForMember(d => d.Shelf, opt => opt.MapFrom(src => src.Shelf));

}

private static TResult GetValueSafety<TInput, TResult>(TInput target, Func<TInput, TResult> getter)

{

return target is null || getter is null ? default : getter.Invoke(target);

}

public static void CreateMapFromAuthorToAuthorViewModel(this IProfileExpression expression)

{

expression.CreateMap<Author, AuthorViewModel>()

.ForMember(

d => d.FullName,

opt => opt.MapFrom(s => s.Name.FullName));

}

public static void CreateMapFromBookToBookViewModel(this IProfileExpression expression)

{

expression.CreateMap<Book, BookViewModel>()

.ForMember(

d => d.Title,

opt => opt.MapFrom(s => s.Title))

.ForMember(

d => d.Author,

opt => opt.MapFrom(s => s.Authors));

}

}

}

1. Program:

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Hosting;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Library.WebServices

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>();

});

}

}

1. Startup.cs:

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using System;

using System.IO;

using Infrastructure.Helpers;

using Library.NH;

using Library.NH.Repositories;

using Library.Services;

using Library.WebServices.Controllers;

using NHibernate;

namespace Library.WebServices

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

AddConfigurationFromFile(services, "appsettings.json");

AddNHibernateConfiguration(services, "relative");

services.AddTransient(sp => sp.GetService<ISessionFactory>().OpenSession());

services.AddSingleton<IAuthorRepository, AuthorRepository>();

services.AddScoped<IAuthorService, AuthorService>();

services.AddSingleton<IBookRepository, BookRepository>();

services.AddScoped<IBookService, BookService>();

services.AddMapper();

}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

internal static IServiceCollection AddConfigurationFromFile(IServiceCollection serviceCollection, string filename)

{

var basePath = Directory.GetParent(AppContext.BaseDirectory).FullName;

var configuration = new ConfigurationBuilder()

.SetBasePath(basePath)

.AddJsonFile(filename, optional: false, reloadOnChange: true)

.Build();

return serviceCollection.AddSingleton(configuration);

}

internal static IServiceCollection AddNHibernateConfiguration(IServiceCollection serviceCollection, string connectionStringKey)

{

return serviceCollection.AddSingleton(serviceProvider => GetSessionFactory(serviceProvider, connectionStringKey));

}

private static ISessionFactory GetSessionFactory(IServiceProvider serviceProvider, string connectionStringKey)

{

var basePath = Directory.GetParent(AppContext.BaseDirectory).FullName;

var configuration = serviceProvider.GetService<IConfigurationRoot>();

var dataSourceLocation = Path.GetFullPath(configuration.GetConnectionString(connectionStringKey), basePath);

NHibernateConfigurator.DataSourceLocation = dataSourceLocation;

NHibernateConfigurator.GetConfiguration(LibraryNHibernateConfigurator.GetAssembly());

return NHibernateConfigurator.GetSessionFactory();

}

}

}

1. Library.NH.Tests:
2. AuthorMappingTests.cs

using Library.NH.Maps;

namespace Library.NH.Tests

{

using NUnit.Framework;

using NHibernate;

using Infrastructure.Helpers;

using System.Collections.Generic;

using FluentNHibernate.Testing;

using Library.Domain;

/// <summary>

/// Модульные тесты для класса <see cref="AuthorMap"/>.

/// </summary>

public class AuthorMappingTests

{

private ISession session;

[SetUp]

public void Setup()

{

this.session = NHibernateTestHelper.GetSession(true, typeof(AuthorMap).Assembly);

}

[Test]

public void Test1()

{

// arrange

var books = new List<Book>

{

new Book("Капитанская дочка"),

new Book("Евгений Онегин"),

new Book("Руслан и Людмила")

};

var testName = new Name("Александр", "Пушкин", "Сергеевич");

//act & assert

new PersistenceSpecification<Author>(this.session)

.CheckProperty(x => x.Name, testName)

.CheckList(x => x.Books, books, book=>book.Id)

.VerifyTheMappings();

}

}

}

1. BookMappingTests.cs:

using Library.NH.Maps;

namespace Library.NH.Tests

{

using NUnit.Framework;

using NHibernate;

using Infrastructure.Helpers;

using System.Collections.Generic;

using FluentNHibernate.Testing;

using Library.Domain;

/// <summary>

/// Модульные тесты для класса <see cref="BookMap"/>.

/// </summary>

public class BookMappingTests

{

private ISession session;

[SetUp]

public void Setup()

{

this.session = NHibernateTestHelper.GetSession(true, typeof(BookMap).Assembly);

}

[Test]

public void Test1()

{

// arrange

var genres = new List<Genre>

{

new Genre("Роман"),

new Genre("Роман\_2")

};

var testName = new Name("Федор", "Достоевский", "Иванович");

var authors = new List<Author>

{

new Author((testName))

};

var testShelve = new Shelf("Вторая сверху полка");

var publishers = new List<Publisher>

{

new Publisher("Эксмо"),

new Publisher("Росмэн")

};

//act & assert

new PersistenceSpecification<Book>(this.session)

.CheckProperty(x => x.Title, "Идиот")

.CheckList(x => x.Genres, genres, genre => genre.Id)

.CheckList(x => x.Authors, authors, author => author.ID)

.CheckReference(x => x.Shelf, testShelve, shelf => shelf.Id)

.CheckList(x => x.Publishers, publishers, publisher => publisher.Id)

.VerifyTheMappings();

}

}

}

1. GenreMappingTests.cs:

using Library.NH.Maps;

namespace Library.NH.Tests

{

using NUnit.Framework;

using NHibernate;

using Infrastructure.Helpers;

using System.Collections.Generic;

using FluentNHibernate.Testing;

using Library.Domain;

/// <summary>

/// Модульные тесты для класса <see cref="GenreMap"/>.

/// </summary>

public class GenreMappingTests

{

private ISession session;

[SetUp]

public void Setup()

{

this.session = NHibernateTestHelper.GetSession(true, typeof(GenreMap).Assembly);

}

[Test]

public void Test1()

{

// arrange

var books = new List<Book>

{

new Book("Горе от ума"),

new Book("Ревизор"),

new Book("Двенадцать стульев")

};

//act & assert

new PersistenceSpecification<Genre>(this.session)

.CheckProperty(x => x.Name, "Комедия")

.CheckList(x => x.Books, books, book => book.Id)

.VerifyTheMappings();

}

}

}

1. PublisherMappingTests.cs:

namespace Library.NH.Tests

{

using NUnit.Framework;

using NHibernate;

using Infrastructure.Helpers;

using System.Collections.Generic;

using FluentNHibernate.Testing;

using Library.Domain;

using Library.NH.Maps;

/// <summary>

/// Модульные тесты для класса <see cref="PublisherMap"/>.

/// </summary>

public class PublisherMappingTests

{

private ISession session;

[SetUp]

public void Setup()

{

this.session = NHibernateTestHelper.GetSession(true, typeof(PublisherMap).Assembly);

}

[Test]

public void Test1()

{

// arrange

var books = new List<Book>

{

new Book("Капитанская дочка"),

new Book("Евгений Онегин"),

new Book("Руслан и Людмила")

};

//act & assert

new PersistenceSpecification<Publisher>(this.session)

.CheckProperty(x => x.Name, "Росмэн")

.CheckList(x => x.Books, books, book=>book.Id)

.VerifyTheMappings();

}

}

}

1. RoomMappingTests.cs:

namespace Library.NH.Tests

{

using NUnit.Framework;

using NHibernate;

using Infrastructure.Helpers;

using System.Collections.Generic;

using FluentNHibernate.Testing;

using Library.Domain;

using Library.NH.Maps;

/// <summary>

/// Модульные тесты для класса <see cref="RoomMap"/>.

/// </summary>

public class RoomMappingTests

{

private ISession session;

[SetUp]

public void Setup()

{

this.session = NHibernateTestHelper.GetSession(true, typeof(BookMap).Assembly);

}

[Test]

public void Test1()

{

// arrange

var shelves = new List<Shelf>

{

new Shelf("Верхняя полка"),

new Shelf("Средняя полка"),

new Shelf("Нижняя полка"),

};

//act & assert

new PersistenceSpecification<Room>(this.session)

.CheckProperty(x => x.Name, "Спальня")

.CheckList(x => x.Shelves, shelves, shelf => shelf.Id)

.VerifyTheMappings();

}

}

}

1. ShelfMappingTests.cs:

namespace Library.NH.Tests

{

using NUnit.Framework;

using NHibernate;

using Infrastructure.Helpers;

using System.Collections.Generic;

using FluentNHibernate.Testing;

using Library.Domain;

using Library.NH.Maps;

/// <summary>

/// Модульные тесты для класса <see cref="ShelfMap"/>.

/// </summary>

public class ShelfMappingTests

{

private ISession session;

[SetUp]

public void Setup()

{

this.session = NHibernateTestHelper.GetSession(true, typeof(ShelfMap).Assembly);

}

[Test]

public void Test1()

{

// arrange

var books = new List<Book>

{

new Book("Война и мир"),

new Book("Метро 2033"),

new Book("Вино из одуванчиков")

};

var testRoom = new Room("Гостиная");

//act & assert

new PersistenceSpecification<Shelf>(this.session)

.CheckProperty(x => x.Description, "Самая нижняя полка")

.CheckList(x => x.Books, books, book => book.Id)

.CheckReference(x=>x.Room, testRoom, room=>room.Id)

.VerifyTheMappings();

}

}

}