Министерство Образования, Культуры и Исследований

**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

**Департамент Информатики**

Отчёт по Индивидуальной работе №2

по предмету

Фреймворки

Проверил:

S. Bodrug, lector universitar

Выполнил:

студент группы IA 2003

Анощенков Данила

Кишинев, 2022

Библиотека

# Цель работы

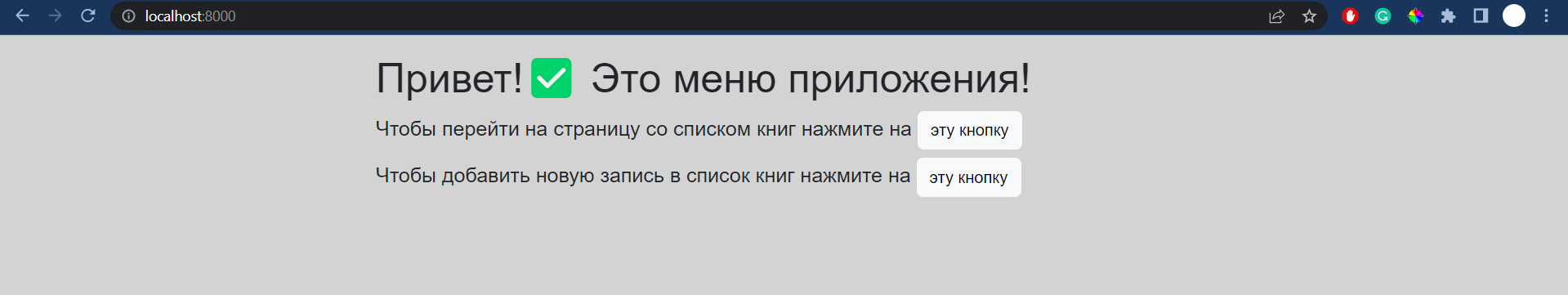
Целью данной индивидуальной работы является использование всех раннее полученных знаний в области разработки backend’а при помощи таких инструментов, как Symfony и языка программирования Php. Главная задача работы заключается в разработки сайта на тему “Библиотека” с обязательной реализацией следующих компонентов:

* Панель меню
* Добавление, манипулирование и удаление элементов списка

# Страница меню

Страница меню – это первичная страница вебсайта, на которую попадает пользователь в начале использования веб-проекта. На ней располагаются следующие элементы:

* Текст приветствия
* Кнопка с ссылкой на страницу записей из списка книг
* Кнопка с ссылкой создания записи в списке книг



**Код twig страницы:**

index.html.twig

{% extends 'base.html.twig' %}

{% block title %} Меню {% endblock %}

{% block body %}

<style>

.outter-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }

</style>

<div class="outter-wrapper">

<h1>Привет!✅ Это меню приложения! </h1>

<h5>Чтобы перейти на страницу со списком книг нажмите на <a href="/books" class="btn btn-light">эту кнопку</a> </h5>

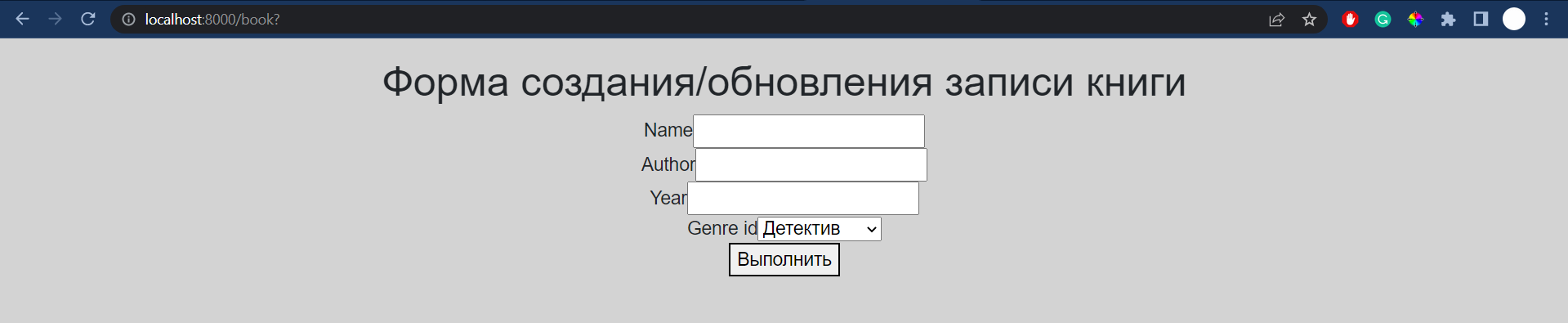
<h5>Чтобы добавить новую запись в список книг нажмите на <a href="/book" class="btn btn-light">эту кнопку</a> </h5>

</div>

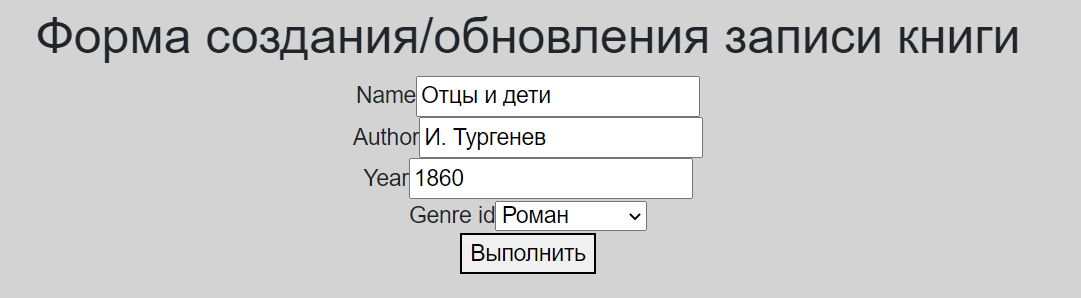
{% endblock %}

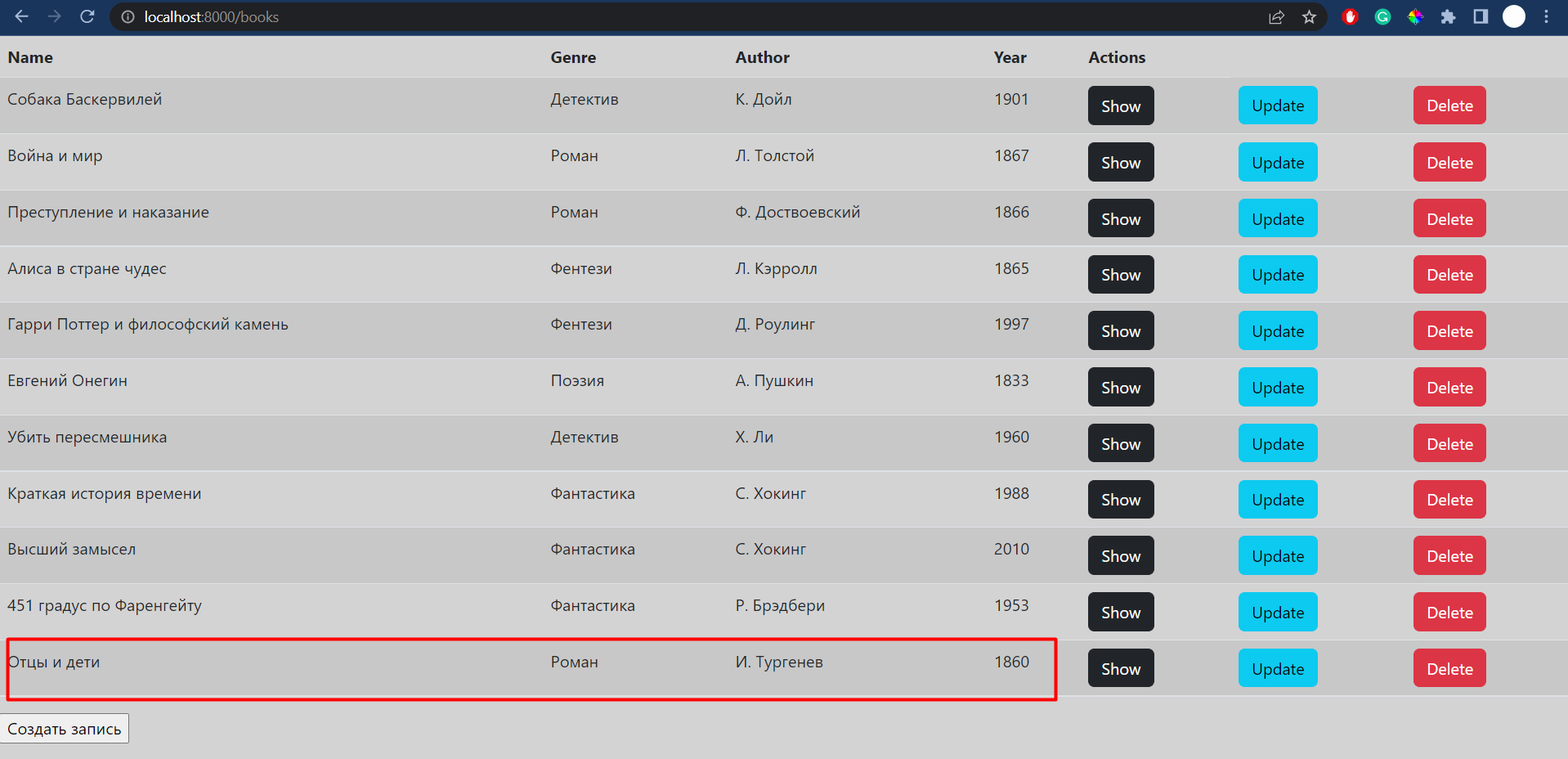
# Страница с формой

Страница с формой – это страница, на которую попадает пользователь после клика на кнопку с ссылкой на страницу добавления новой записи, которая находится в меню. На ней непосредственно форма с возможностью добавления или обновления какой-либо записи из списка всех книг.



Проверим создание новой записи в таблице на практике:





**Код twig страницы:**

form.html.twig:

{% extends('base.html.twig') %}

{% block title%} Форма для книг {% endblock %}

{% block body%}

<style>

.outter-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; text-align: center; }

</style>

<div class="outter-wrapper">

<h1>Форма создания/обновления записи книги</h1>

{{ form\_start(form) }}

{{ form\_widget(form) }}

{{ form\_end (form) }}

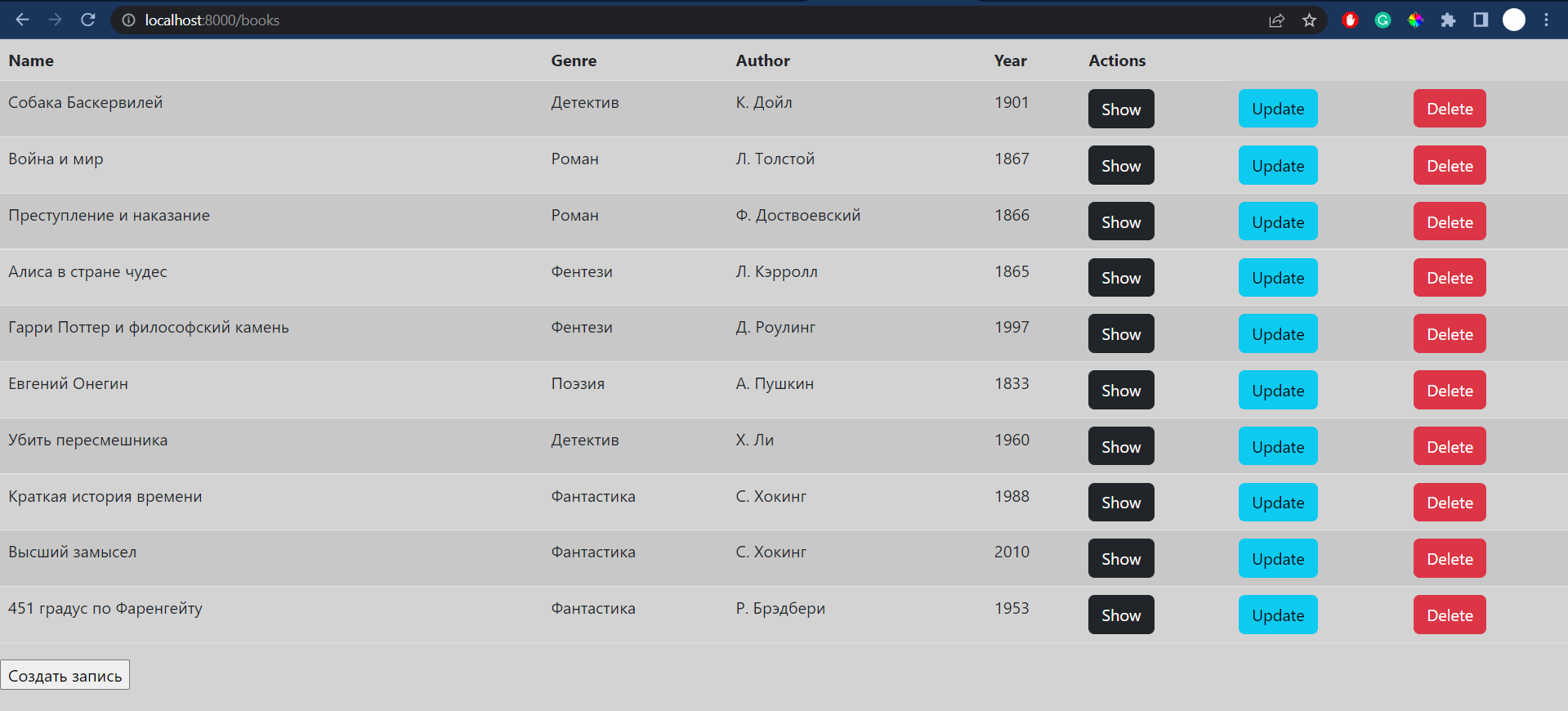
</div>

{% endblock %}

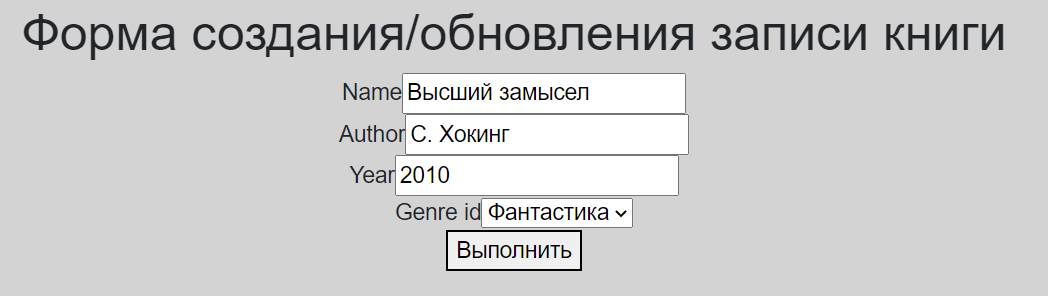
# Страница списка книг

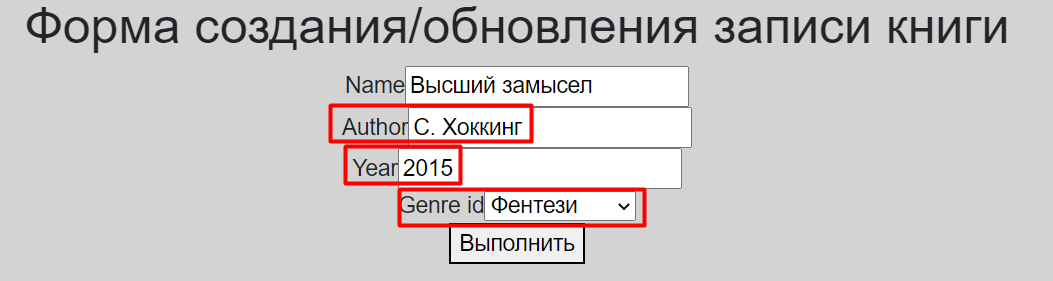
Страница списка книг – это страница, на которую пользователь попадает после клика на кнопку с ссылкой на страницу записей из списка книг, которая находится в меню. На ней располагаются следующие элементы:

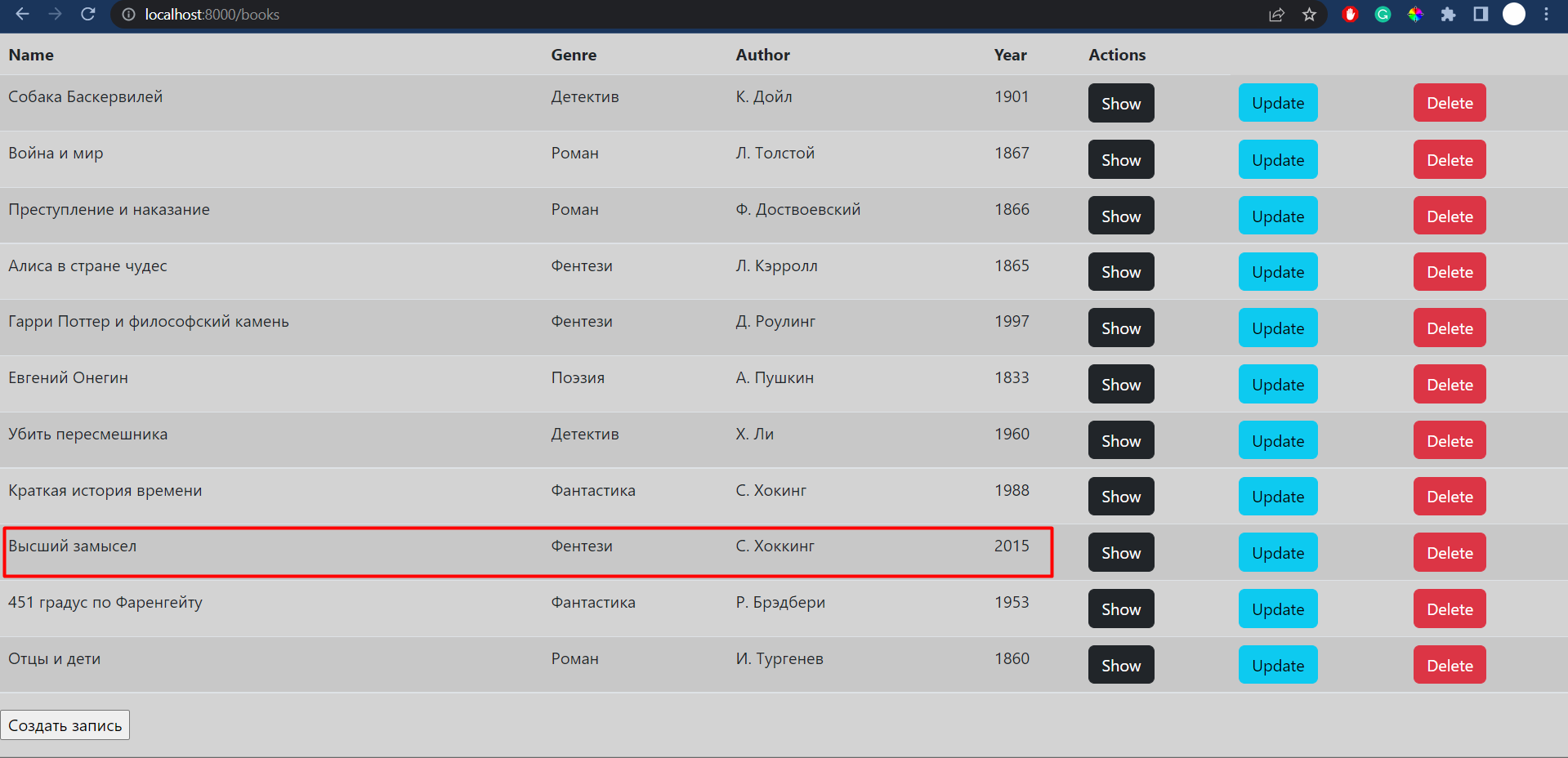
* Таблица для отображения всех записей из списка книг
* Кнопка добавления новой записи, которая переносит пользователя на отдельную страницу с формой создания/обновления
* Кнопка обновления существующей записи, которая переносит пользователя на отдельную страницу с формой создания/обновления
* Кнопка удаления записи
* Кнопка подробного просмотра записи, которая переносит пользователя на отдельную страницу



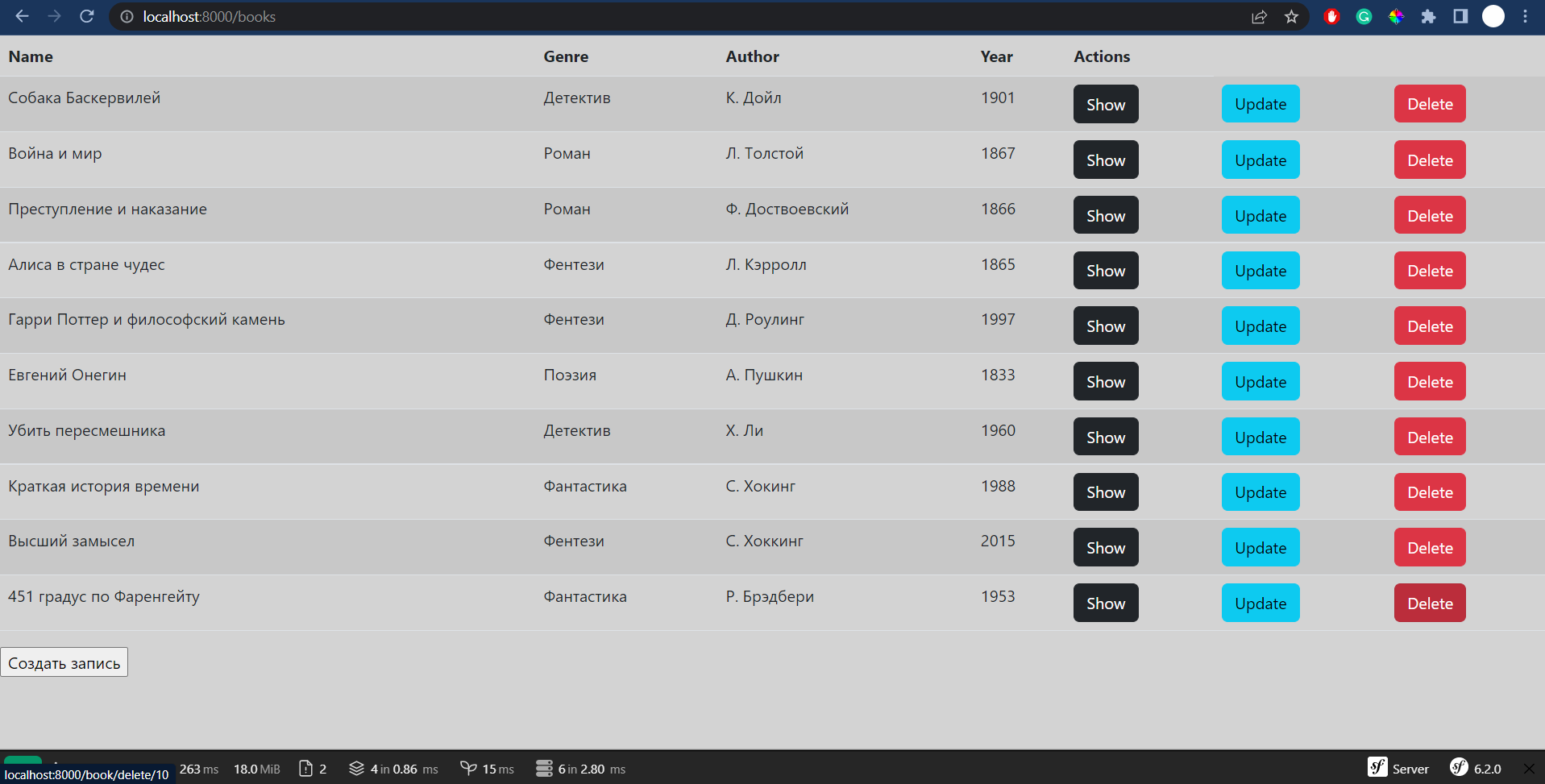
Проверим обновление уже существующей записи в таблице на практике, нажав на кнопку ‘Update’:







Проверим удаление существующей записи в таблице на практике, нажав на кнопку ‘Delete’ и удалив последнюю запись:



**Код twig страницы:**

books.html.twig:

{% extends 'base.html.twig' %}

{% block title%} Список книг {% endblock %}

{% block body %}

{% if books %}

<table class="table table-striped">

<thead>

<tr>

<th>Name</th>

<th>Genre</th>

<th>Author</th>

<th>Year</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

{% for book in books %}

<tr>

<td>{{ book.name }}</td>

<td>{{ book.genreId.name }}</td>

<td>{{ book.author }}</td>

<td>{{ book.year }}</td>

<td>

<a href="/book/{{ book.id }}" class="btn btn-dark">Show</a>

</td>

<td>

<a href="/book/update/{{ book.id }}" class="btn btn-info">Update</a>

</td>

<td>

<a href="/book/delete/{{ book.id }}" class="btn btn-danger">Delete</a>

</td>

</tr>

{% endfor %}

</tbody>

</table>

<form action="/book">

<input type="submit" value="Создать запись" />

</form>

{% else %}

{% endif %}

{% endblock %}

**Код контроллера:**

BookController.php:

class BookController extends AbstractController

{

    #[Route('/')]

    public function index(LoggerInterface $logger): Response

    {

        $logger->info("Index page: user IP " . $\_SERVER['REMOTE\_ADDR']);

        return $this->render('book/index.html.twig', []);

    }

    /\*\*

     \* Matches /blog/\*

     \*

     \* @Route("/books", name="display\_books")

     \*/

    public function displayCars(ManagerRegistry $doctrine, LoggerInterface $logger): Response

    {

        $bookService = new BookService($doctrine->getManager(), Book::class);

        $books = $bookService->getAllBooks();

        $logger->info("List of cars page: user IP " . $\_SERVER['REMOTE\_ADDR']);

        return $this->render('default/books.html.twig', array('books' => $books));

    }

    #[Route('/book', name: 'create\_book')]

    public function createCar(Request $request, ManagerRegistry $doctrine, LoggerInterface $logger): Response

    {

        $book = new Book();

        $form = $this->createForm(BookFormType::class, $book);

        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {

            $book = $form->getData();

            $entityManager = $doctrine->getManager();

            $entityManager->persist($book);

            $entityManager->flush();

            $logger->info("Create book action: user IP " . $\_SERVER['REMOTE\_ADDR']);

            return $this->redirectToRoute('display\_books');

        }

        $logger->info("Create book action: user IP " . $\_SERVER['REMOTE\_ADDR']);

        return $this->render('default/form.html.twig', array(

            'form' => $form->createView()

        ));

    }

    #[Route('/book/{id}', name: 'display\_book')]

    public function displayCar(int $id, ManagerRegistry $doctrine, LoggerInterface $logger): Response

    {

        $bookService  = new BookService($doctrine->getManager(), Book::class);

        $book = $bookService->getBook($id);

        $logger->info("Book page: user IP " . $\_SERVER['REMOTE\_ADDR']);

        return $this->render('book/display.html.twig', array('book' => $book));

    }

    #[Route('/book/update/{id}', name: 'update\_book')]

    public function updateCar(Request $request, $id, ManagerRegistry $doctrine, LoggerInterface $logger): Response

    {

        $book = $doctrine->getRepository(Book::class)->find($id);

        $form = $this->createForm(BookFormType::class, $book);

        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {

            $bookService = new BookService($doctrine->getManager(), Book::class);

            $bookService->addBook($book);

            $logger->info("Update book action: user IP " . $\_SERVER['REMOTE\_ADDR']);

            return $this->redirectToRoute('display\_books');

        }

        $logger->info("Update car action: user IP " . $\_SERVER['REMOTE\_ADDR']);

        return $this->render('default/form.html.twig', array(

            'form' => $form->createView()

        ));

    }

    #[Route('/book/delete/{id}', name: 'delete\_book')]

    public function deleteCar(Request $request, $id, ManagerRegistry $doctrine, LoggerInterface $logger): Response

    {

        $bookService = new BookService($doctrine->getManager(), Book::class);

        $bookService->deleteBook($id);

        $logger->info("Delete book action: user IP " . $\_SERVER['REMOTE\_ADDR']);

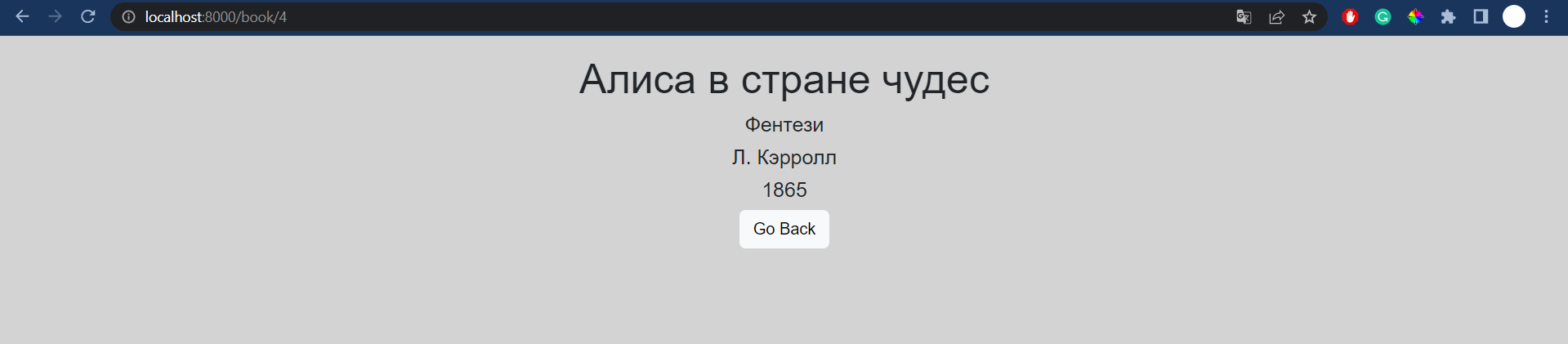
        return $this->redirectToRoute('display\_books');

    }

}

# Страница просмотра книги

Страница просмотра книги – это страница, на которую пользователь попадает после клика на кнопку 'Show', которая находится на странице списка всех записей книг. На ней располагаются подробное описание записи о книге.



**Код twig страницы:**

display.html.twig:

{% extends 'base.html.twig' %}

{% block title%} <p>Описание книги</p> {% endblock %}

{% block body %}

<style>

.outter-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; text-align: center; }

</style>

<div class="outter-wrapper">

<h1>{{ book.name }}</h1>

<h5>{{ book.genreId.name }}</h5>

<h5>{{ book.author }}</h5>

<h5>{{ book.year }}</h5>

<a href="/books" class="btn btn-light">Go Back</a>

</div>

{% endblock %}

# Entity объекты

Book.php

<?php

namespace App\Entity;

use App\Repository\BookRepository;

use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: BookRepository::class)]

class Book

{

    #[ORM\Id]

    #[ORM\GeneratedValue]

    #[ORM\Column]

    private ?int $id = null;

    #[ORM\Column(length: 50)]

    private ?string $name = null;

    #[ORM\Column(length: 50)]

    private ?string $author = null;

    #[ORM\Column(length: 4)]

    private ?string $year = null;

    #[ORM\ManyToOne(inversedBy: 'books')]

    #[ORM\JoinColumn(nullable: false)]

    private ?Genre $genre\_id = null;

    public function getId(): ?int

    {

        return $this->id;

    }

    public function getName(): ?string

    {

        return $this->name;

    }

    public function setName(string $name): self

    {

        $this->name = $name;

        return $this;

    }

    public function getAuthor(): ?string

    {

        return $this->author;

    }

    public function setAuthor(string $author): self

    {

        $this->author = $author;

        return $this;

    }

    public function getYear(): ?string

    {

        return $this->year;

    }

    public function setYear(string $year): self

    {

        $this->year = $year;

        return $this;

    }

    public function getGenreId(): ?Genre

    {

        return $this->genre\_id;

    }

    public function setGenreId(?Genre $genre\_id): self

    {

        $this->genre\_id = $genre\_id;

        return $this;

    }

}

Genre.php

<?php

namespace App\Entity;

use App\Repository\GenreRepository;

use Doctrine\Common\Collections\ArrayCollection;

use Doctrine\Common\Collections\Collection;

use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: GenreRepository::class)]

class Genre

{

    #[ORM\Id]

    #[ORM\GeneratedValue]

    #[ORM\Column]

    private ?int $id = null;

    #[ORM\Column(length: 50)]

    private ?string $name = null;

    #[ORM\OneToMany(mappedBy: 'genre\_id', targetEntity: Book::class)]

    private Collection $books;

    public function \_\_construct()

    {

        $this->books = new ArrayCollection();

    }

    public function getId(): ?int

    {

        return $this->id;

    }

    public function getName(): ?string

    {

        return $this->name;

    }

    public function setName(string $name): self

    {

        $this->name = $name;

        return $this;

    }

    /\*\*

     \* @return Collection<int, Book>

     \*/

    public function getBooks(): Collection

    {

        return $this->books;

    }

    public function addBook(Book $book): self

    {

        if (!$this->books->contains($book)) {

            $this->books->add($book);

            $book->setGenreId($this);

        }

        return $this;

    }

    public function removeBook(Book $book): self

    {

        if ($this->books->removeElement($book)) {

            // set the owning side to null (unless already changed)

            if ($book->getGenreId() === $this) {

                $book->setGenreId(null);

            }

        }

        return $this;

    }

    public function \_\_toString()

    {

        return $this->name;

    }

}

Form.php

<?php

namespace App\Entity;

class Form

{

    private $name;

    private $genre;

    private $author;

    private $year;

    public function getName()

    {

        return $this->name;

    }

    public function setName($name)

    {

        $this->name = $name;

    }

    public function getGenre()

    {

        return $this->genre;

    }

    public function setGenre($genre)

    {

        $this->genre = $genre;

    }

    public function getAuthor()

    {

        return $this->author;

    }

    public function setAuthor($author)

    {

        $this->author = $author;

    }

    public function getYear()

    {

        return $this->year;

    }

    public function setYear($year)

    {

        $this->year = $year;

    }

}

# Использованные команды

* composer create-project symfony/skeleton Symfony\_individual\_project
* composer require twig
* composer require annotations
* composer require profiler
* composer require logger
* composer require monolog/monolog
* composer require symfony/form
* composer require symfony/orm-pack
* composer require --dev symfony/maker-bundle
* php bin/console make:entity
* php bin/console make:migration
* php bin/console doctrine:migrations:migrate
* php bin/console make:controller BookController