**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине: **«Разработка приложений баз данных для информационных систем»**

на тему: **«Разработка моделей и контроллеров ASP.NET MVC** **приложения баз данных»**

Выполнил: студент гр. ИТП-31

Сетко А.И.

Принял: доцент

Асенчик О.Д.

Гомель 2019

**Цель работы:** ознакомиться с возможностями ASP.NET Core MVC и Entity Framework Core для разработки слоя доступа к данным, хранящимся в базе данных, и обработки запросов пользователя посредством контроллеров.

**Задание:**

Создать с использованием ASP.NET Core MVC Web-приложение, содержащее набор классов, моделирующих предметную область, и осуществляющих генерацию и заполнение тестовыми наборами записей базу данных. Разработать один компонент middleware, контроллеры и представления для выборки и отображения информации из не менее чем 3- таблиц базы данных с использованием механизма внедрение зависимостей.

Для этого необходимо создать:

* Классы, моделирующие не менее чем три таблицы базы данных согласно вашему варианту. Перечень таблиц предварительно согласовывается с преподавателем. Одна из таблиц обязательно должна находиться на стороне отношения «многие» связи с другой таблицей в схеме базы данных.
* Класс контекста данных.
* Другие классы, например, классы View Model и т.п. (при необходимости).
* Компонент middleware, вызываемый в классе Startup, для инициализации базы данных путем заполнения ее таблиц тестовым набором записей.
* Классы контроллеров (по одному на каждую таблицу базы данных) для обработки обращений пользователя, выборки данных из таблиц и вызова соответствующих представлений для отображения выбранных данных.
* Разработать представления для отображения данных из таблиц, выбранных контроллерами. Представления, работающими с таблицами, стоящими на стороне отношения «многие» в схеме базы данных, должны выводить вместо кодов внешних ключей смысловые значения из связанных таблиц, стоящих на стороне отношения «один».
* Используя предварительно созданный и сконфигурированный в классе Startup профиль кэширования, подключить кэширование вывода для страниц с использованием атрибута ResponseCache для соответствующих методов контроллера. Данные в кэше хранить неизменными в течение 2\*N+240 секунд, где N- номер вашего варианта.
* С использованием средств разработчика браузера (Chrome, Firefox) продемонстрировать ускорение обработки запроса при наличии кэширования с использованием атрибута ResponseCache.

**Ход работы**

**Вариант 14**

**Верификация программы:**

При запуске приложения, будет открыта стартовая страница, которая содержит в себе навигационное меню, для удобного перемещения по страницам. Начальная страница относится к стандартному контроллеру, который создаётся приложением с самого начала. Начальная страница приведена на рисунке 1.1.



Рисунок 1.1 – Главная страница, за которую отвечает контроллер *Home*

Для каждого пункта навигационного меню, разработаны собственные контроллеры, которые отвечают за вывод содержимого на экран. Результат работы контроллеров приведён на рисунках 1.2 – 1.7.

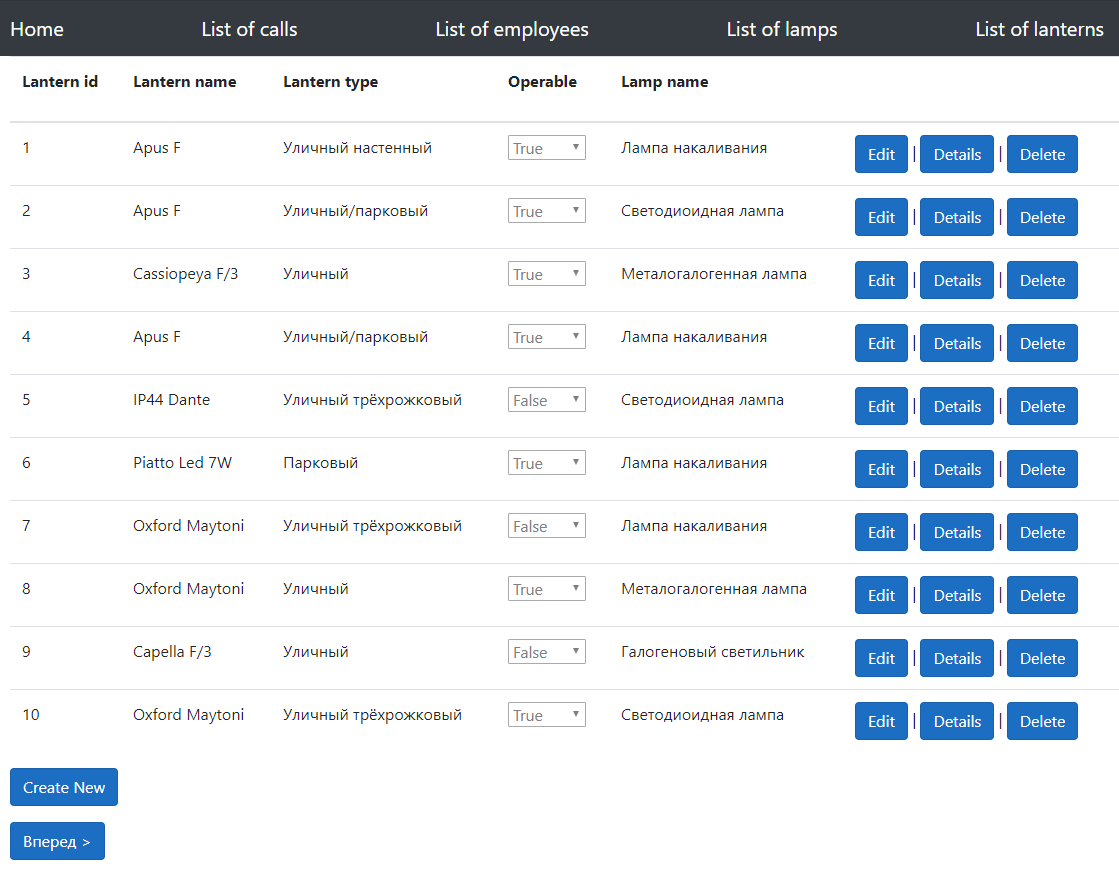


Рисунок 1.2 – Переход на страницу, за которую отвечает контроллер *Lanterns*

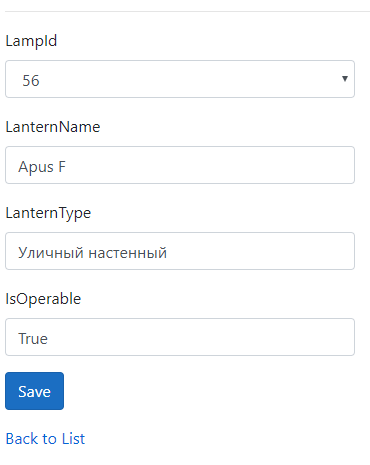


Рисунок 1.3 – Страница изменения данных в таблице

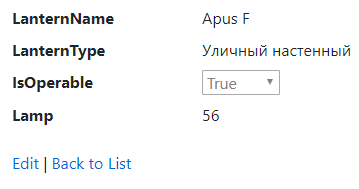


Рисунок 1.4 – Страница просмотра конкретных данных

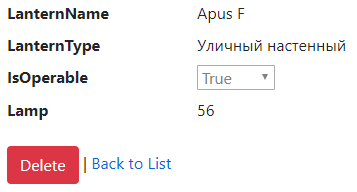


Рисунок 1.5 – Страница удаления данных

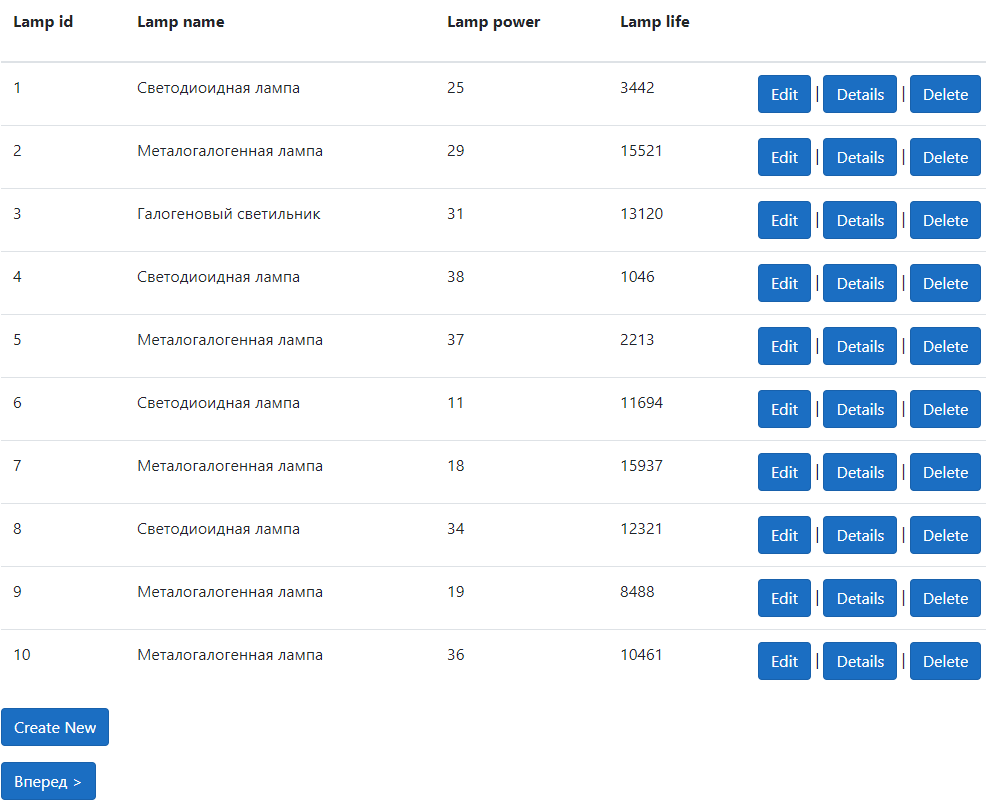


Рисунок 1.6 – Переход на страницу, за которую отвечает контроллер *Lamps*

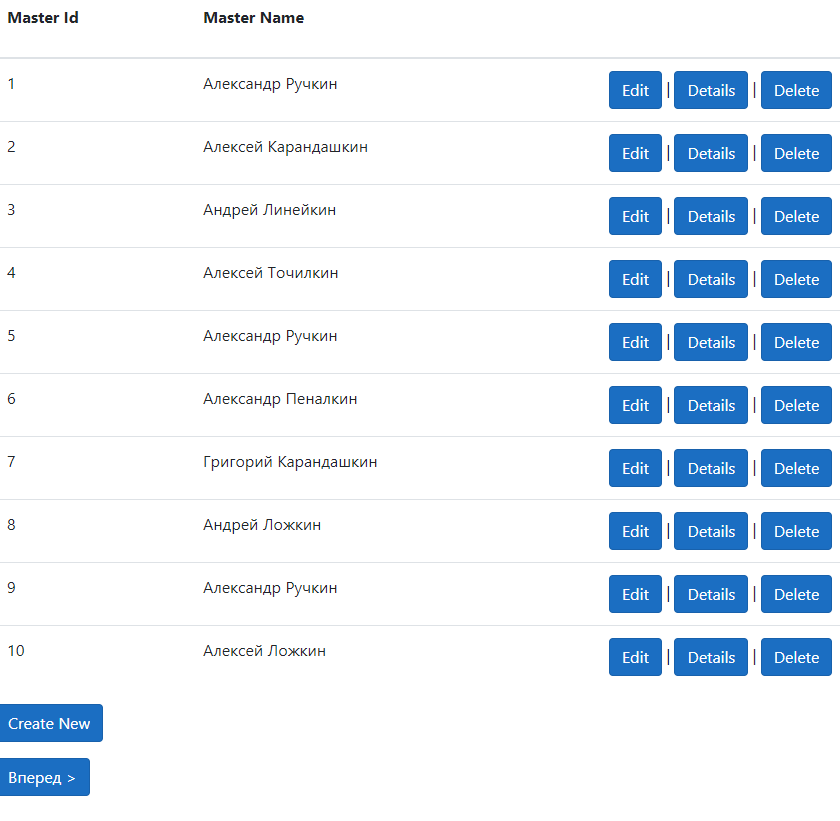


Рисунок 1.7 – Переход на страницу, за которую отвечает контроллер *Masters*

Результат работы кеширования при использовании атрибута *ResponseCashe* приведён на рисунках 1.8-1.9



Рисунок 1.8 – Происходит кеширование страницы



Рисунок 1.9 – Происходит загрузка из кэша о чём свидетельствует надпись *disk cache* в меню *Size*

**Вывод:** в ходе выполнения лабораторной работы была изучена произошло ознакомление с разработкой приложений на *Asp.Net Core*. Особенностью работы на данной платформе является наличие контроллеров, которые дают возможность обрабатывать запросы. Для лучшей переносимости программы, был добавлен компонент *middleware*, который заполняет несколько основных таблиц и в случае надобности, создаёт базу данных. Также для удобства в использовании, была добавлена простейшая пагинация, которая представляет собой две кнопки: «Вперёд» и «Назад».

**Приложение А**

**Листинг программного комплекса**

**appsettings.json**

"ConnectionStrings": {

"SqlServerConnection": "Server=.\\sqlexpress;Database=dbStreetLight;Trusted\_Connection=True;MultipleActiveResultSets=True"

},

**Startup.cs**

namespace MvcApplication

{

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)

{

string connection = Configuration.GetConnectionString("SqlServerConnection");

services.AddDbContext<LightingContext>(options => options.UseSqlServer(connection));

services.AddSession();

services.AddControllersWithViews();

services.AddMvc();

}

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

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseSession();

app.UseDbInitializer();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

}

}

}

**Master.cs**

namespace MvcApplication.Models

{

public partial class Master

{

public Master()

{

Calls = new HashSet<Call>();

}

public int MasterId { get; set; }

[DisplayName("Sensei name")]

public string MasterName { get; set; }

public virtual ICollection<Call> Calls { get; set; }

}

}

**Lamp.cs**

namespace MvcApplication.Models

{

public partial class Lamp

{

public Lamp()

{

Lanterns = new HashSet<Lantern>();

}

public int LampId { get; set; }

[DisplayName("Lamp name")]

public string LampName { get; set; }

public int? LampPower { get; set; }

public int? LampLife { get; set; }

public virtual ICollection<Lantern> Lanterns { get; set; }

}

}

**Lantern.cs**

namespace MvcApplication.Models

{

public partial class Lantern

{

public Lantern()

{

Calls = new HashSet<Call>();

SectionsLightings = new HashSet<SectionsLighting>();

}

public int LanternId { get; set; }

public int? LampId { get; set; }

public string LanternName { get; set; }

public string LanternType { get; set; }

public bool? IsOperable { get; set; }

public virtual Lamp Lamp { get; set; }

public virtual ICollection<Call> Calls { get; set; }

public virtual ICollection<SectionsLighting> SectionsLightings { get; set; }

}

}

**LightingContext.cs**

namespace MvcApplication.Data

{

public partial class LightingContext : DbContext

{

public LightingContext(DbContextOptions<LightingContext> options)

: base(options)

{

}

public virtual DbSet<Call> Calls { get; set; }

public virtual DbSet<Lamp> Lamps { get; set; }

public virtual DbSet<Lantern> Lanterns { get; set; }

public virtual DbSet<Master> Masters { get; set; }

public virtual DbSet<Section> Sections { get; set; }

public virtual DbSet<SectionsLighting> SectionsLightings { get; set; }

public virtual DbSet<Street> Streets { get; set; }

}

}

**DbInitializerMiddleware.cs**

namespace MvcApplication.Middleware

{

public class DbInitializerMiddleware

{

private readonly RequestDelegate \_next;

public DbInitializerMiddleware(RequestDelegate next)

{

// инициализация базы данных по университетам

\_next = next;

}

public Task Invoke(HttpContext context, IServiceProvider serviceProvider, LightingContext dbContext)

{

if (!(context.Session.Keys.Contains("starting")))

{

DbInitializer.Initialize(dbContext);

context.Session.SetString("starting", "Yes");

}

// Call the next delegate/middleware in the pipeline

return \_next.Invoke(context);

}

}

public static class DbInitializerExtensions

{

public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

**DbInitializer.cs**

namespace MvcApplication.Data

{

public static class DbInitializer

{

public static void Initialize(LightingContext db)

{

Random random = new Random();

db.Database.EnsureCreated();

if (db.Masters.Any() || db.Calls.Any() || db.Lamps.Any())

{

return;

}

/\*================================================= Блок инициализации данных =================================================\*/

int mastersNumber = 100;

int lampsNumber = 100;

int lanternNumber = 100;

int callsNumber = 100;

int streetsNumber = 100;

int sectionsNumber = 100;

int sectionsLightingsNumber = 100;

string[] masterNames = { "Григорий", "Фёдор", "Пётр", "Андрей", "Николай", "Александр", "Алексей", "Вячеслав" };

string[] masterSurnames = { "Трампов", "Линейкин", "Ручкин", "Карандашкин", "Стёркин", "Точилкин", "Пеналкин", "Ложкин" };

string[] lampNames = { "Лампа накаливания", "Светодиоидная лампа", "Металогалогенная лампа", "Галогеновый светильник" };

string[] lanternNames = { "IP44 Dante", "Oxford Maytoni", "Capella F/3", "Apus F", "Cassiopeya F/3", "Piatto Led 7W" };

string[] lanternTypes = { "Уличный", "Парковый", "Уличный/парковый", "Уличный трёхрожковый", "Уличный настенный" };

string[] streetNames = { "Садовая", "Ленина", "Фрунзе", "Гагарина", "Советская", "Богдана Хмельницкого" };

int lampLife, lampPower, workTime, countOfHouses, sectionNumber;

DateTime dateCall;

/\*================================================= Блок заполнения таблиц =================================================\*/

for (int i = 0; i < mastersNumber; i++)

{

StringBuilder name = new StringBuilder();

name.Append(getRandomString(masterNames, random)).Append(" ").Append(getRandomString(masterSurnames, random));

db.Masters.Add(new Master { MasterName = name.ToString() });

}

db.SaveChanges();

for (int i = 0; i < lampsNumber; i++)

{

StringBuilder name = new StringBuilder();

name.Append(getRandomString(lampNames, random));

db.Lamps.Add(new Lamp { LampName = name.ToString(), LampPower = random.Next(5, 40), LampLife = random.Next(1000, 20000) });

}

db.SaveChanges();

for (int i = 0; i < lanternNumber; i++)

{

StringBuilder name = new StringBuilder();

StringBuilder type = new StringBuilder();

name.Append(getRandomString(lanternNames, random));

type.Append(getRandomString(lanternTypes, random));

bool isOperable = random.Next(0, 2) == 0 ? false : true;

db.Lanterns.Add(new Lantern { LanternName = name.ToString(), LanternType = type.ToString(), LampId = random.Next(1, lanternNumber), IsOperable = isOperable });

}

db.SaveChanges();

}

private static string getRandomString(string[] array, Random random)

{

return array[random.Next(0, array.Length)];

}

}

}

**HomeController.cs**

namespace MvcApplication.Controllers

{

public class HomeController : Controller

{

private LightingContext \_db;

public HomeController(LightingContext db)

{

\_db = db;

}

public ActionResult Index()

{

var masters = \_db.Masters.ToList();

return View(masters);

}

public IActionResult Privacy()

{

return View();

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

public IActionResult Masters()

{

return View();

}

}

}

namespace MvcApplication.Controllers

{

public class LampsController : Controller

{

private readonly LightingContext \_context;

public LampsController(LightingContext context)

{

\_context = context;

}

// GET: Lamps

[ResponseCache(Duration = 250)]

public async Task<IActionResult> Index(int page = 1)

{

int pageSize = 10;

var lamps = \_context.Lamps.ToList();

var count = lamps.Count();

var items = lamps.Skip((page - 1) \* pageSize).Take(pageSize).ToList();

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

IndexViewModel viewModel = new IndexViewModel

{

PageViewModel = pageViewModel,

Lamps = items

};

return View(viewModel);

}

// GET: Lamps/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var lamp = await \_context.Lamps

.FirstOrDefaultAsync(m => m.LampId == id);

if (lamp == null)

{

return NotFound();

}

return View(lamp);

}

// GET: Lamps/Create

public IActionResult Create()

{

return View();

}

// POST: Lamps/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("LampId,LampName,LampPower,LampLife")] Lamp lamp)

{

if (ModelState.IsValid)

{

\_context.Add(lamp);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(lamp);

}

// GET: Lamps/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var lamp = await \_context.Lamps.FindAsync(id);

if (lamp == null)

{

return NotFound();

}

return View(lamp);

}

// POST: Lamps/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("LampId,LampName,LampPower,LampLife")] Lamp lamp)

{

if (id != lamp.LampId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(lamp);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!LampExists(lamp.LampId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(lamp);

}

// GET: Lamps/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var lamp = await \_context.Lamps

.FirstOrDefaultAsync(m => m.LampId == id);

if (lamp == null)

{

return NotFound();

}

return View(lamp);

}

// POST: Lamps/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var lamp = await \_context.Lamps.FindAsync(id);

\_context.Lamps.Remove(lamp);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool LampExists(int id)

{

return \_context.Lamps.Any(e => e.LampId == id);

}

}

}

**LanternsController.cs**

namespace MvcApplication.Controllers

{

public class LanternsController : Controller

{

private readonly LightingContext \_context;

public LanternsController(LightingContext context)

{

\_context = context;

}

// GET: Lanterns

[ResponseCache(Duration = 250)]

public async Task<IActionResult> Index(int page = 1)

{

int pageSize = 10;

var lightingContext = \_context.Lanterns.Include(l => l.Lamp);

var count = lightingContext.Count();

var items = lightingContext.Skip((page - 1) \* pageSize).Take(pageSize).ToList();

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

IndexViewModel viewModel = new IndexViewModel

{

PageViewModel = pageViewModel,

Lanterns = items

};

return View(viewModel);

}

// GET: Lanterns/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var lantern = await \_context.Lanterns

.Include(l => l.Lamp)

.FirstOrDefaultAsync(m => m.LanternId == id);

if (lantern == null)

{

return NotFound();

}

return View(lantern);

}

// GET: Lanterns/Create

public IActionResult Create()

{

ViewData["LampId"] = new SelectList(\_context.Lamps, "LampId", "LampId");

return View();

}

// POST: Lanterns/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("LanternId,LampId,LanternName,LanternType,IsOperable")] Lantern lantern)

{

if (ModelState.IsValid)

{

\_context.Add(lantern);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["LampId"] = new SelectList(\_context.Lamps, "LampId", "LampId", lantern.LampId);

return View(lantern);

}

// GET: Lanterns/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var lantern = await \_context.Lanterns.FindAsync(id);

if (lantern == null)

{

return NotFound();

}

ViewData["LampId"] = new SelectList(\_context.Lamps, "LampId", "LampId", lantern.LampId);

return View(lantern);

}

// POST: Lanterns/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("LanternId,LampId,LanternName,LanternType,IsOperable")] Lantern lantern)

{

if (id != lantern.LanternId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(lantern);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!LanternExists(lantern.LanternId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["LampId"] = new SelectList(\_context.Lamps, "LampId", "LampId", lantern.LampId);

return View(lantern);

}

// GET: Lanterns/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var lantern = await \_context.Lanterns

.Include(l => l.Lamp)

.FirstOrDefaultAsync(m => m.LanternId == id);

if (lantern == null)

{

return NotFound();

}

return View(lantern);

}

// POST: Lanterns/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var lantern = await \_context.Lanterns.FindAsync(id);

\_context.Lanterns.Remove(lantern);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool LanternExists(int id)

{

return \_context.Lanterns.Any(e => e.LanternId == id);

}

}

}

**MastersController.cs**

namespace MvcApplication.Controllers

{

public class MastersController : Controller

{

private readonly LightingContext \_context;

public MastersController(LightingContext context)

{

\_context = context;

}

// GET: Masters

[ResponseCache(Duration = 250)]

public async Task<IActionResult> Index(int page = 1)

{

int pageSize = 10;

var masters = \_context.Masters.ToList();

var count = masters.Count();

var items = masters.Skip((page - 1) \* pageSize).Take(pageSize).ToList();

PageViewModel pageViewModel = new PageViewModel(count, page, pageSize);

IndexViewModel viewModel = new IndexViewModel

{

PageViewModel = pageViewModel,

Masters = items

};

return View(viewModel);

}

public async Task<IActionResult> Lamps()

{

return View(await \_context.Lamps.ToListAsync());

}

// GET: Masters/Details/5

public async Task<IActionResult> Details(int? id)

{

if (id == null)

{

return NotFound();

}

var master = await \_context.Masters

.FirstOrDefaultAsync(m => m.MasterId == id);

if (master == null)

{

return NotFound();

}

return View(master);

}

// GET: Masters/Create

public IActionResult Create()

{

return View();

}

// POST: Masters/Create

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("MasterId,MasterName")] Master master)

{

if (ModelState.IsValid)

{

\_context.Add(master);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(master);

}

// GET: Masters/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var master = await \_context.Masters.FindAsync(id);

if (master == null)

{

return NotFound();

}

return View(master);

}

// POST: Masters/Edit/5

// To protect from overposting attacks, please enable the specific properties you want to bind to, for

// more details see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("MasterId,MasterName")] Master master)

{

if (id != master.MasterId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(master);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!MasterExists(master.MasterId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(master);

}

// GET: Masters/Delete/5

public async Task<IActionResult> Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var master = await \_context.Masters

.FirstOrDefaultAsync(m => m.MasterId == id);

if (master == null)

{

return NotFound();

}

return View(master);

}

// POST: Masters/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var master = await \_context.Masters.FindAsync(id);

\_context.Masters.Remove(master);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool MasterExists(int id)

{

return \_context.Masters.Any(e => e.MasterId == id);

}

}

}

**IndexViewModel.cs**

namespace MvcApplication.Models

{

public class IndexViewModel

{

public List<Master> Masters { get; set; }

public List<Lantern> Lanterns { get; set; }

public List<Lamp> Lamps { get; set; }

public PageViewModel PageViewModel { get; set; }

}

}

**PageViewModel.cs**

namespace MvcApplication.Models

{

public class PageViewModel

{

public int PageNumber { get; private set; }

public int TotalPages { get; private set; }

public PageViewModel(int count, int pageNumber, int pageSize)

{

PageNumber = pageNumber;

TotalPages = (int)Math.Ceiling(count / (double)pageSize);

}

public bool HasPreviousPage

{

get

{

return (PageNumber > 1);

}

}

public bool HasNextPage

{

get

{

return (PageNumber < TotalPages);

}

}

}

}