**ПРИЛОЖЕНИЕ А**

**ЛИСТИНГ ПРОГРАММЫ**

**Код класса *CallsController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class CallsController : Controller

{

private readonly TelecomDbContext \_context;

public CallsController(TelecomDbContext context)

{

\_context = context;

}

// GET: Calls

public async Task<IActionResult> Index(string dateSearch, string durationSearch, string phoneSearch, int pageNumber = 1)

{

ViewData["DateFilter"] = dateSearch;

ViewData["DurationFilter"] = durationSearch;

ViewData["PhoneFilter"] = phoneSearch;

var calls = from c in \_context.Calls.Include(c => c.Contract) select c;

if (!string.IsNullOrEmpty(dateSearch) && DateTime.TryParse(dateSearch, out var date))

{

calls = calls.Where(c => c.CallDate.Date == date.Date);

}

if (!string.IsNullOrEmpty(durationSearch) && int.TryParse(durationSearch, out var duration))

{

calls = calls.Where(c => c.CallDuration == duration);

}

if (!string.IsNullOrEmpty(phoneSearch))

{

calls = calls.Where(c => c.Contract.PhoneNumber.Contains(phoneSearch));

}

int pageSize = 30;

var totalCalls = await calls.CountAsync();

var paginatedList = await calls

.OrderBy(c => c.CallId)

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

.ToListAsync();

var viewModel = new PaginatedList<Call>(paginatedList, totalCalls, pageNumber, pageSize);

return View(viewModel);

}

// GET: Calls/Details/5

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

{

if (id == null)

{

return NotFound();

}

var call = await \_context.Calls

.Include(c => c.Contract)

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

if (call == null)

{

return NotFound();

}

return View(call);

}

// GET: Calls/Create

public IActionResult Create()

{

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View();

}

// POST: Calls/Create

// To protect from overposting attacks, 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("CallId,ContractId,CallDate,CallDuration")] Call call)

{

if (ModelState.IsValid)

{

\_context.Add(call);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(call);

}

// GET: Calls/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var call = await \_context.Calls.FindAsync(id);

if (call == null)

{

return NotFound();

}

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(call);

}

// POST: Calls/Edit/5

// To protect from overposting attacks, 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("CallId,ContractId,CallDate,CallDuration")] Call call)

{

if (id != call.CallId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(call);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!CallExists(call.CallId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(call);

}

// GET: Calls/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var call = await \_context.Calls

.Include(c => c.Contract)

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

if (call == null)

{

return NotFound();

}

return View(call);

}

// POST: Calls/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var call = await \_context.Calls.FindAsync(id);

if (call != null)

{

\_context.Calls.Remove(call);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool CallExists(int id)

{

return \_context.Calls.Any(e => e.CallId == id);

}

}

**Код класса *ContractsController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class ContractsController : Controller

{

private readonly TelecomDbContext \_context;

public ContractsController(TelecomDbContext context)

{

\_context = context;

}

// GET: Contracts

public async Task<IActionResult> Index(

int pageNumber = 1,

string searchSubscriber = "",

string searchTariff = "",

string searchStaff = "",

string searchPhone = "",

string sortOrder = "")

{

int pageSize = 30;

ViewData["CurrentSort"] = sortOrder;

ViewData["SubscriberSortParm"] = String.IsNullOrEmpty(sortOrder) ? "subscriber\_desc" : "";

ViewData["TariffSortParm"] = sortOrder == "tariff" ? "tariff\_desc" : "tariff";

ViewData["StaffSortParm"] = sortOrder == "staff" ? "staff\_desc" : "staff";

ViewData["PhoneSortParm"] = sortOrder == "phone" ? "phone\_desc" : "phone";

var telecomDbContext = \_context.Contracts

.Include(c => c.Staff)

.Include(c => c.Subscriber)

.Include(c => c.TariffPlan)

.AsQueryable();

if (!string.IsNullOrEmpty(searchSubscriber))

{

telecomDbContext = telecomDbContext.Where(c => c.Subscriber.FullName.Contains(searchSubscriber));

}

if (!string.IsNullOrEmpty(searchTariff))

{

telecomDbContext = telecomDbContext.Where(c => c.TariffPlan.TariffName.Contains(searchTariff));

}

if (!string.IsNullOrEmpty(searchStaff))

{

telecomDbContext = telecomDbContext.Where(c => c.Staff.FullName.Contains(searchStaff));

}

if (!string.IsNullOrEmpty(searchPhone))

{

telecomDbContext = telecomDbContext.Where(c => c.PhoneNumber.Contains(searchPhone));

}

telecomDbContext = sortOrder switch

{

"subscriber\_desc" => telecomDbContext.OrderByDescending(c => c.Subscriber.FullName),

"tariff" => telecomDbContext.OrderBy(c => c.TariffPlan.TariffName),

"tariff\_desc" => telecomDbContext.OrderByDescending(c => c.TariffPlan.TariffName),

"staff" => telecomDbContext.OrderBy(c => c.Staff.FullName),

"staff\_desc" => telecomDbContext.OrderByDescending(c => c.Staff.FullName),

"phone" => telecomDbContext.OrderBy(c => c.PhoneNumber),

"phone\_desc" => telecomDbContext.OrderByDescending(c => c.PhoneNumber),

\_ => telecomDbContext.OrderBy(c => c.Subscriber.FullName),

};

var totalContracts = await telecomDbContext.CountAsync();

var contracts = await telecomDbContext

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

.ToListAsync();

var viewModel = new PaginatedList<Contract>(contracts, totalContracts, pageNumber, pageSize);

ViewData["searchSubscriber"] = searchSubscriber;

ViewData["searchTariff"] = searchTariff;

ViewData["searchStaff"] = searchStaff;

ViewData["searchPhone"] = searchPhone;

return View(viewModel);

}

// GET: Contracts/Details/5

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

{

if (id == null)

{

return NotFound();

}

var contract = await \_context.Contracts

.Include(c => c.Staff)

.Include(c => c.Subscriber)

.Include(c => c.TariffPlan)

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

if (contract == null)

{

return NotFound();

}

return View(contract);

}

// GET: Contracts/Create

public IActionResult Create()

{

ViewData["StaffId"] = new SelectList(\_context.Staff, "StaffId", "FullName");

ViewData["SubscriberId"] = new SelectList(\_context.Subscribers, "SubscriberId", "FullName");

ViewData["TariffPlanId"] = new SelectList(\_context.TariffPlans, "TariffPlanId", "TariffName");

return View();

}

// POST: Contracts/Create

// To protect from overposting attacks, 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("ContractId,SubscriberId,TariffPlanId,ContractDate,ContractEndDate,PhoneNumber,StaffId")] Contract contract)

{

if (ModelState.IsValid)

{

\_context.Add(contract);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["StaffId"] = new SelectList(\_context.Staff, "StaffId", "StaffId", contract.StaffId);

ViewData["SubscriberId"] = new SelectList(\_context.Subscribers, "SubscriberId", "SubscriberId", contract.SubscriberId);

ViewData["TariffPlanId"] = new SelectList(\_context.TariffPlans, "TariffPlanId", "TariffPlanId", contract.TariffPlanId);

return View(contract);

}

// GET: Contracts/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var contract = await \_context.Contracts.FindAsync(id);

if (contract == null)

{

return NotFound();

}

ViewData["StaffName"] = new SelectList(\_context.Staff, "StaffId", "FullName", contract.StaffId);

ViewData["SubscriberName"] = new SelectList(\_context.Subscribers, "SubscriberId", "FullName", contract.SubscriberId);

ViewData["TariffPlanName"] = new SelectList(\_context.TariffPlans, "TariffPlanId", "TariffName", contract.TariffPlanId);

return View(contract);

}

// POST: Contracts/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("ContractId,SubscriberId,TariffPlanId,ContractDate,ContractEndDate,PhoneNumber,StaffId")] Contract contract)

{

if (id != contract.ContractId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(contract);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

catch (DbUpdateConcurrencyException)

{

if (!ContractExists(contract.ContractId))

{

return NotFound();

}

else

{

throw;

}

}

}

ViewData["StaffName"] = new SelectList(\_context.Staff, "StaffId", "FullName", contract.StaffId);

ViewData["SubscriberName"] = new SelectList(\_context.Subscribers, "SubscriberId", "FullName", contract.SubscriberId);

ViewData["TariffPlanName"] = new SelectList(\_context.TariffPlans, "TariffPlanId", "TariffName", contract.TariffPlanId);

return View(contract);

}

// GET: Contracts/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var contract = await \_context.Contracts

.Include(c => c.Staff)

.Include(c => c.Subscriber)

.Include(c => c.TariffPlan)

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

if (contract == null)

{

return NotFound();

}

return View(contract);

}

// POST: Contracts/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var contract = await \_context.Contracts.FindAsync(id);

if (contract != null)

{

\_context.Contracts.Remove(contract);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool ContractExists(int id)

{

return \_context.Contracts.Any(e => e.ContractId == id);

}

}

}

**Код класса *InternetUsagesController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class InternetUsagesController : Controller

{

private readonly TelecomDbContext \_context;

public InternetUsagesController(TelecomDbContext context)

{

\_context = context;

}

// GET: InternetUsages

public async Task<IActionResult> Index(DateTime? dateSearch, decimal? dataSentSearch, decimal? dataReceivedSearch, string phoneSearch, int pageNumber = 1)

{

ViewData["DateFilter"] = dateSearch?.ToString("yyyy-MM-dd");

ViewData["DataSentFilter"] = dataSentSearch;

ViewData["DataReceivedFilter"] = dataReceivedSearch;

ViewData["PhoneFilter"] = phoneSearch;

var internetUsages = \_context.InternetUsages.Include(i => i.Contract).AsQueryable();

if (dateSearch.HasValue)

{

internetUsages = internetUsages.Where(i => i.UsageDate.Date == dateSearch.Value.Date);

}

if (dataSentSearch.HasValue)

{

internetUsages = internetUsages.Where(i => i.DataSentMb >= dataSentSearch.Value);

}

if (dataReceivedSearch.HasValue)

{

internetUsages = internetUsages.Where(i => i.DataReceivedMb >= dataReceivedSearch.Value);

}

if (!string.IsNullOrEmpty(phoneSearch))

{

internetUsages = internetUsages.Where(i => i.Contract.PhoneNumber.Contains(phoneSearch));

}

int pageSize = 30;

var totalItems = await internetUsages.CountAsync();

var paginatedList = await internetUsages

.OrderBy(i => i.UsageDate)

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

.ToListAsync();

var viewModel = new PaginatedList<InternetUsage>(paginatedList, totalItems, pageNumber, pageSize);

return View(viewModel);

}

// GET: InternetUsages/Details/5

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

{

if (id == null)

{

return NotFound();

}

var internetUsage = await \_context.InternetUsages

.Include(i => i.Contract)

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

if (internetUsage == null)

{

return NotFound();

}

return View(internetUsage);

}

// GET: InternetUsages/Create

public IActionResult Create()

{

ViewData["ContractId"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View();

}

// POST: InternetUsages/Create

// To protect from overposting attacks, 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("UsageId,ContractId,UsageDate,DataSentMb,DataReceivedMb")] InternetUsage internetUsage)

{

if (ModelState.IsValid)

{

\_context.Add(internetUsage);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["ContractId"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(internetUsage);

}

// GET: InternetUsages/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var internetUsage = await \_context.InternetUsages.FindAsync(id);

if (internetUsage == null)

{

return NotFound();

}

ViewData["ContractId"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(internetUsage);

}

// POST: InternetUsages/Edit/5

// To protect from overposting attacks, 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("UsageId,ContractId,UsageDate,DataSentMb,DataReceivedMb")] InternetUsage internetUsage)

{

if (id != internetUsage.UsageId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(internetUsage);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!InternetUsageExists(internetUsage.UsageId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["ContractId"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(internetUsage);

}

// GET: InternetUsages/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var internetUsage = await \_context.InternetUsages

.Include(i => i.Contract)

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

if (internetUsage == null)

{

return NotFound();

}

return View(internetUsage);

}

// POST: InternetUsages/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var internetUsage = await \_context.InternetUsages.FindAsync(id);

if (internetUsage != null)

{

\_context.InternetUsages.Remove(internetUsage);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool InternetUsageExists(int id)

{

return \_context.InternetUsages.Any(e => e.UsageId == id);

}

}

}

**Код класса *MessagesController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class MessagesController : Controller

{

private readonly TelecomDbContext \_context;

public MessagesController(TelecomDbContext context)

{

\_context = context;

}

// GET: Messages

public async Task<IActionResult> Index(string dateSearch, string phoneNumberSearch, string isMmsSearch, int pageNumber = 1)

{

ViewData["DateFilter"] = dateSearch;

ViewData["PhoneNumberFilter"] = phoneNumberSearch;

ViewData["IsMmsFilter"] = isMmsSearch;

var messages = from m in \_context.Messages.Include(m => m.Contract) select m;

if (!string.IsNullOrEmpty(dateSearch) && DateTime.TryParse(dateSearch, out var date))

{

messages = messages.Where(c => c.MessageDate.Date == date.Date);

}

if (!string.IsNullOrEmpty(phoneNumberSearch))

{

messages = messages.Where(m => m.Contract.PhoneNumber.Contains(phoneNumberSearch));

}

if (isMmsSearch == "true")

{

messages = messages.Where(m => m.IsMms);

}

else if (isMmsSearch == "false")

{

messages = messages.Where(m => !m.IsMms);

}

int pageSize = 30;

var totalMessages = await messages.CountAsync();

var paginatedList = await messages

.OrderBy(m => m.MessageDate)

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

.ToListAsync();

var viewModel = new PaginatedList<Message>(paginatedList, totalMessages, pageNumber, pageSize);

return View(viewModel);

}

// GET: Messages/Details/5

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

{

if (id == null)

{

return NotFound();

}

var message = await \_context.Messages

.Include(m => m.Contract)

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

if (message == null)

{

return NotFound();

}

return View(message);

}

// GET: Messages/Create

public IActionResult Create()

{

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View();

}

// POST: Messages/Create

// To protect from overposting attacks, 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("MessageId,ContractId,MessageDate,IsMms")] Message message)

{

if (ModelState.IsValid)

{

\_context.Add(message);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(message);

}

// GET: Messages/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var message = await \_context.Messages.FindAsync(id);

if (message == null)

{

return NotFound();

}

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(message);

}

// POST: Messages/Edit/5

// To protect from overposting attacks, 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("MessageId,ContractId,MessageDate,IsMms")] Message message)

{

if (id != message.MessageId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(message);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!MessageExists(message.MessageId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["ContractName"] = new SelectList(\_context.Contracts, "ContractId", "PhoneNumber");

return View(message);

}

// GET: Messages/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var message = await \_context.Messages

.Include(m => m.Contract)

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

if (message == null)

{

return NotFound();

}

return View(message);

}

// POST: Messages/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var message = await \_context.Messages.FindAsync(id);

if (message != null)

{

\_context.Messages.Remove(message);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool MessageExists(int id)

{

return \_context.Messages.Any(e => e.MessageId == id);

}

}

}

**Код класса *StaffPositionsController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class StaffPositionsController : Controller

{

private readonly TelecomDbContext \_context;

public StaffPositionsController(TelecomDbContext context)

{

\_context = context;

}

// GET: StaffPositions

public async Task<IActionResult> Index()

{

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

}

// GET: StaffPositions/Details/5

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

{

if (id == null)

{

return NotFound();

}

var staffPosition = await \_context.StaffPositions

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

if (staffPosition == null)

{

return NotFound();

}

return View(staffPosition);

}

// GET: StaffPositions/Create

public IActionResult Create()

{

return View();

}

// POST: StaffPositions/Create

// To protect from overposting attacks, 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("PositionId,PositionName")] StaffPosition staffPosition)

{

if (ModelState.IsValid)

{

\_context.Add(staffPosition);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(staffPosition);

}

// GET: StaffPositions/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var staffPosition = await \_context.StaffPositions.FindAsync(id);

if (staffPosition == null)

{

return NotFound();

}

return View(staffPosition);

}

// POST: StaffPositions/Edit/5

// To protect from overposting attacks, 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("PositionId,PositionName")] StaffPosition staffPosition)

{

if (id != staffPosition.PositionId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(staffPosition);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!StaffPositionExists(staffPosition.PositionId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(staffPosition);

}

// GET: StaffPositions/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var staffPosition = await \_context.StaffPositions

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

if (staffPosition == null)

{

return NotFound();

}

return View(staffPosition);

}

// POST: StaffPositions/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var staffPosition = await \_context.StaffPositions.FindAsync(id);

if (staffPosition != null)

{

\_context.StaffPositions.Remove(staffPosition);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool StaffPositionExists(int id)

{

return \_context.StaffPositions.Any(e => e.PositionId == id);

}

}

}

**Код класса *StaffsController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class StaffsController : Controller

{

private readonly TelecomDbContext \_context;

public StaffsController(TelecomDbContext context)

{

\_context = context;

}

// GET: Staffs

public async Task<IActionResult> Index()

{

var telecomDbContext = \_context.Staff.Include(s => s.Position);

return View(await telecomDbContext.ToListAsync());

}

// GET: Staffs/Details/5

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

{

if (id == null)

{

return NotFound();

}

var staff = await \_context.Staff

.Include(s => s.Position)

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

if (staff == null)

{

return NotFound();

}

return View(staff);

}

// GET: Staffs/Create

public IActionResult Create()

{

ViewData["PositionName"] = new SelectList(\_context.StaffPositions, "PositionId", "PositionName");

return View();

}

// POST: Staffs/Create

// To protect from overposting attacks, 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("StaffId,FullName,PositionId,Education")] Staff staff)

{

if (ModelState.IsValid)

{

\_context.Add(staff);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

ViewData["PositionName"] = new SelectList(\_context.StaffPositions, "PositionId", "PositionName", staff.PositionId);

return View(staff);

}

// GET: Staffs/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var staff = await \_context.Staff.FindAsync(id);

if (staff == null)

{

return NotFound();

}

ViewData["PositionName"] = new SelectList(\_context.StaffPositions, "PositionId", "PositionName", staff.PositionId);

return View(staff);

}

// POST: Staffs/Edit/5

// To protect from overposting attacks, 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("StaffId,FullName,PositionId,Education")] Staff staff)

{

if (id != staff.StaffId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(staff);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!StaffExists(staff.StaffId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

ViewData["PositionName"] = new SelectList(\_context.StaffPositions, "PositionId", "PositionName", staff.PositionId);

return View(staff);

}

// GET: Staffs/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var staff = await \_context.Staff

.Include(s => s.Position)

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

if (staff == null)

{

return NotFound();

}

return View(staff);

}

// POST: Staffs/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var staff = await \_context.Staff.FindAsync(id);

if (staff != null)

{

\_context.Staff.Remove(staff);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool StaffExists(int id)

{

return \_context.Staff.Any(e => e.StaffId == id);

}

}

}

**Код класса *SubscribersController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class SubscribersController : Controller

{

private readonly TelecomDbContext \_context;

public SubscribersController(TelecomDbContext context)

{

\_context = context;

}

public async Task<IActionResult> Index(

string nameSearch,

string addressSearch,

string passportSearch,

string tariffSearch,

DateTime? startDate,

DateTime? endDate,

int pageNumber = 1,

string sortOrder = "",

string currentSort = "")

{

ViewBag.Tariffs = await \_context.TariffPlans

.Select(t => t.TariffName)

.Distinct()

.ToListAsync();

ViewData["NameFilter"] = nameSearch;

ViewData["AddressFilter"] = addressSearch;

ViewData["PassportFilter"] = passportSearch;

ViewData["TariffFilter"] = tariffSearch;

ViewData["StartDate"] = startDate?.ToString("yyyy-MM-dd");

ViewData["EndDate"] = endDate?.ToString("yyyy-MM-dd");

ViewData["CurrentSort"] = sortOrder;

ViewData["NameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewData["AddressSortParm"] = sortOrder == "address" ? "address\_desc" : "address";

ViewData["PassportSortParm"] = sortOrder == "passport" ? "passport\_desc" : "passport";

ViewData["TariffSortParm"] = sortOrder == "tariff" ? "tariff\_desc" : "tariff";

var subscribers = from s in \_context.Subscribers

.Include(s => s.Contracts)

.ThenInclude(c => c.TariffPlan)

select s;

if (!string.IsNullOrEmpty(nameSearch))

{

subscribers = subscribers.Where(s => s.FullName.Contains(nameSearch));

}

if (!string.IsNullOrEmpty(addressSearch))

{

subscribers = subscribers.Where(s => s.HomeAddress.Contains(addressSearch));

}

if (!string.IsNullOrEmpty(passportSearch))

{

subscribers = subscribers.Where(s => s.PassportData.Contains(passportSearch));

}

if (!string.IsNullOrEmpty(tariffSearch))

{

subscribers = subscribers.Where(s => s.Contracts.Any(c => c.TariffPlan.TariffName.Contains(tariffSearch)));

}

DateOnly? startDateOnly = startDate.HasValue ? DateOnly.FromDateTime(startDate.Value) : null;

DateOnly? endDateOnly = endDate.HasValue ? DateOnly.FromDateTime(endDate.Value) : null;

if (startDateOnly.HasValue && endDateOnly.HasValue)

{

subscribers = subscribers.Where(s =>

s.Contracts.Any() &&

s.Contracts.All(c => c.ContractDate >= startDateOnly && c.ContractDate <= endDateOnly));

}

subscribers = sortOrder switch

{

"name\_desc" => subscribers.OrderByDescending(s => s.FullName),

"address" => subscribers.OrderBy(s => s.HomeAddress),

"address\_desc" => subscribers.OrderByDescending(s => s.HomeAddress),

"passport" => subscribers.OrderBy(s => s.PassportData),

"passport\_desc" => subscribers.OrderByDescending(s => s.PassportData),

\_ => subscribers.OrderBy(s => s.FullName),

};

int pageSize = 30;

var totalSubscribers = await subscribers.CountAsync();

var paginatedList = await subscribers

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

.ToListAsync();

var viewModel = new PaginatedList<Subscriber>(paginatedList, totalSubscribers, pageNumber, pageSize);

return View(viewModel);

}

// GET: Subscribers/Details/5

public async Task<IActionResult> Details(int? id, DateTime? startDate, DateTime? endDate)

{

if (id == null)

{

return NotFound();

}

var subscriber = await \_context.Subscribers

.Include(s => s.Contracts)

.ThenInclude(c => c.Calls)

.Include(s => s.Contracts)

.ThenInclude(c => c.Messages)

.Include(s => s.Contracts)

.ThenInclude(c => c.InternetUsages)

.Include(s => s.Contracts)

.ThenInclude(c => c.TariffPlan)

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

if (subscriber == null)

{

return NotFound();

}

decimal grandTotalCost = 0;

if (startDate.HasValue && endDate.HasValue)

{

foreach (var contract in subscriber.Contracts)

{

var tariff = contract.TariffPlan;

if (tariff != null)

{

decimal totalCallCost = contract.Calls

.Where(call => call.CallDate >= startDate && call.CallDate <= endDate)

.Sum(call => call.CallDuration \* (tariff.IsPerSecond ? tariff.LocalCallRate / 60 : tariff.LocalCallRate));

decimal totalSmsCost = contract.Messages

.Where(m => m.MessageDate >= startDate && m.MessageDate <= endDate)

.Count(m => !m.IsMms) \* tariff.SmsRate;

decimal totalMmsCost = contract.Messages

.Where(m => m.MessageDate >= startDate && m.MessageDate <= endDate)

.Count(m => m.IsMms) \* tariff.MmsRate;

decimal totalInternetCost = contract.InternetUsages

.Where(u => u.UsageDate >= startDate && u.UsageDate <= endDate)

.Sum(u => (u.DataSentMb + u.DataReceivedMb) \* tariff.DataRatePerMb);

grandTotalCost += totalCallCost + totalSmsCost + totalMmsCost + totalInternetCost;

}

}

}

ViewBag.GrandTotalCost = grandTotalCost;

ViewBag.StartDate = startDate?.ToString("yyyy-MM-dd");

ViewBag.EndDate = endDate?.ToString("yyyy-MM-dd");

return View(subscriber);

}

// GET: Subscribers/Create

public IActionResult Create()

{

return View();

}

// POST: Subscribers/Create

// To protect from overposting attacks, 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("SubscriberId,FullName,HomeAddress,PassportData")] Subscriber subscriber)

{

if (ModelState.IsValid)

{

\_context.Add(subscriber);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(subscriber);

}

// GET: Subscribers/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var subscriber = await \_context.Subscribers.FindAsync(id);

if (subscriber == null)

{

return NotFound();

}

return View(subscriber);

}

// POST: Subscribers/Edit/5

// To protect from overposting attacks, 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("SubscriberId,FullName,HomeAddress,PassportData")] Subscriber subscriber)

{

if (id != subscriber.SubscriberId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(subscriber);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!SubscriberExists(subscriber.SubscriberId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(subscriber);

}

// GET: Subscribers/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var subscriber = await \_context.Subscribers

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

if (subscriber == null)

{

return NotFound();

}

return View(subscriber);

}

// POST: Subscribers/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var subscriber = await \_context.Subscribers.FindAsync(id);

if (subscriber != null)

{

\_context.Subscribers.Remove(subscriber);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool SubscriberExists(int id)

{

return \_context.Subscribers.Any(e => e.SubscriberId == id);

}

}

}

**Код класса *TariffPlansController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers.tables

{

[Authorize]

public class TariffPlansController : Controller

{

private readonly TelecomDbContext \_context;

public TariffPlansController(TelecomDbContext context)

{

\_context = context;

}

// GET: TariffPlans

public async Task<IActionResult> Index(

int pageNumber = 1,

string searchTariffName = "",

string searchSubscriptionFee = "",

string searchLocalCallRate = "",

string searchLongDistanceCallRate = "",

string searchInternationalCallRate = "",

string sortOrder = "",

string currentSort = "")

{

int pageSize = 30;

ViewData["CurrentSort"] = sortOrder;

ViewData["TariffNameSortParm"] = String.IsNullOrEmpty(sortOrder) ? "tariffName\_desc" : "";

ViewData["SubscriptionFeeSortParm"] = sortOrder == "subscriptionFee" ? "subscriptionFee\_desc" : "subscriptionFee";

ViewData["LocalCallRateSortParm"] = sortOrder == "localCallRate" ? "localCallRate\_desc" : "localCallRate";

var telecomDbContext = \_context.TariffPlans.AsQueryable();

if (!string.IsNullOrEmpty(searchTariffName))

{

telecomDbContext = telecomDbContext.Where(t => t.TariffName.Contains(searchTariffName));

}

if (decimal.TryParse(searchSubscriptionFee, out decimal subscriptionFee))

{

telecomDbContext = telecomDbContext.Where(t => t.SubscriptionFee == subscriptionFee);

}

if (decimal.TryParse(searchLocalCallRate, out decimal localCallRate))

{

telecomDbContext = telecomDbContext.Where(t => t.LocalCallRate == localCallRate);

}

if (decimal.TryParse(searchLongDistanceCallRate, out decimal longDistanceCallRate))

{

telecomDbContext = telecomDbContext.Where(t => t.LongDistanceCallRate == longDistanceCallRate);

}

if (decimal.TryParse(searchInternationalCallRate, out decimal internationalCallRate))

{

telecomDbContext = telecomDbContext.Where(t => t.InternationalCallRate == internationalCallRate);

}

telecomDbContext = sortOrder switch

{

"tariffName\_desc" => telecomDbContext.OrderByDescending(t => t.TariffName),

"subscriptionFee" => telecomDbContext.OrderBy(t => t.SubscriptionFee),

"subscriptionFee\_desc" => telecomDbContext.OrderByDescending(t => t.SubscriptionFee),

"localCallRate" => telecomDbContext.OrderBy(t => t.LocalCallRate),

"localCallRate\_desc" => telecomDbContext.OrderByDescending(t => t.LocalCallRate),

\_ => telecomDbContext.OrderBy(t => t.TariffName)

};

var totalTariffPlans = await telecomDbContext.CountAsync();

var tariffPlans = await telecomDbContext

.Skip((pageNumber - 1) \* pageSize)

.Take(pageSize)

.ToListAsync();

var viewModel = new PaginatedList<TariffPlan>(tariffPlans, totalTariffPlans, pageNumber, pageSize);

ViewData["searchTariffName"] = searchTariffName;

ViewData["searchSubscriptionFee"] = searchSubscriptionFee;

ViewData["searchLocalCallRate"] = searchLocalCallRate;

ViewData["searchLongDistanceCallRate"] = searchLongDistanceCallRate;

ViewData["searchInternationalCallRate"] = searchInternationalCallRate;

return View(viewModel);

}

// GET: TariffPlans/Details/5

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

{

if (id == null)

{

return NotFound();

}

var tariffPlan = await \_context.TariffPlans

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

if (tariffPlan == null)

{

return NotFound();

}

return View(tariffPlan);

}

// GET: TariffPlans/Create

public IActionResult Create()

{

return View();

}

// POST: TariffPlans/Create

// To protect from overposting attacks, 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("TariffPlanId,TariffName,SubscriptionFee,LocalCallRate,LongDistanceCallRate,InternationalCallRate,IsPerSecond,SmsRate,MmsRate,DataRatePerMb")] TariffPlan tariffPlan)

{

if (ModelState.IsValid)

{

\_context.Add(tariffPlan);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(tariffPlan);

}

// GET: TariffPlans/Edit/5

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

{

if (id == null)

{

return NotFound();

}

var tariffPlan = await \_context.TariffPlans.FindAsync(id);

if (tariffPlan == null)

{

return NotFound();

}

return View(tariffPlan);

}

// POST: TariffPlans/Edit/5

// To protect from overposting attacks, 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("TariffPlanId,TariffName,SubscriptionFee,LocalCallRate,LongDistanceCallRate,InternationalCallRate,IsPerSecond,SmsRate,MmsRate,DataRatePerMb")] TariffPlan tariffPlan)

{

if (id != tariffPlan.TariffPlanId)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(tariffPlan);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!TariffPlanExists(tariffPlan.TariffPlanId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(tariffPlan);

}

// GET: TariffPlans/Delete/5

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

{

if (id == null)

{

return NotFound();

}

var tariffPlan = await \_context.TariffPlans

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

if (tariffPlan == null)

{

return NotFound();

}

return View(tariffPlan);

}

// POST: TariffPlans/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var tariffPlan = await \_context.TariffPlans.FindAsync(id);

if (tariffPlan != null)

{

\_context.TariffPlans.Remove(tariffPlan);

}

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

private bool TariffPlanExists(int id)

{

return \_context.TariffPlans.Any(e => e.TariffPlanId == id);

}

}

}

**Код класса *UsersController.cs***

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Data;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers

{

[Authorize(Roles = "admin")]

public class UsersController : Controller

{

private readonly UserManager<IdentityUser> \_userManager;

public UsersController(UserManager<IdentityUser> userManager)

{

\_userManager = userManager;

}

// GET: Users

public async Task<IActionResult> Index()

{

var users = await \_userManager.Users.ToListAsync();

return View(users);

}

// GET: Users/Details/5

public async Task<IActionResult> Details(string id)

{

if (id == null)

{

return NotFound();

}

var user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

return NotFound();

}

return View(user);

}

// GET: Users/Create

public IActionResult Create()

{

return View();

}

// POST: Users/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("UserName,Email,Password")] CreateUserViewModel model)

{

if (ModelState.IsValid)

{

var user = new IdentityUser { UserName = model.UserName, Email = model.Email };

var result = await \_userManager.CreateAsync(user, model.Password);

\_userManager.AddToRoleAsync(user, Role.manager.ToString()).Wait();

if (result.Succeeded)

{

return RedirectToAction(nameof(Index));

}

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

return View(model);

}

// GET: Users/Edit/5

public async Task<IActionResult> Edit(string id)

{

if (id == null)

{

return NotFound();

}

var user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

return NotFound();

}

var model = new EditUserViewModel { Id = user.Id, UserName = user.UserName, Email = user.Email };

return View(model);

}

// POST: Users/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(string id, [Bind("Id,UserName,Email")] EditUserViewModel model)

{

if (id != model.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

var user = await \_userManager.FindByIdAsync(id);

if (user != null)

{

user.UserName = model.UserName;

user.Email = model.Email;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return RedirectToAction(nameof(Index));

}

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

// GET: Users/Delete/5

public async Task<IActionResult> Delete(string id)

{

if (id == null)

{

return NotFound();

}

var user = await \_userManager.FindByIdAsync(id);

if (user == null)

{

return NotFound();

}

return View(user);

}

// POST: Users/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(string id)

{

var user = await \_userManager.FindByIdAsync(id);

if (user != null)

{

await \_userManager.DeleteAsync(user);

}

return RedirectToAction(nameof(Index));

}

}

}

**Код класса *HomeController.cs***

using Microsoft.AspNetCore.Mvc;

using System.Diagnostics;

using TelecomWeb.Models;

namespace TelecomWeb.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger)

{

\_logger = logger;

}

public IActionResult Index()

{

return View();

}

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 });

}

}

}

**Код класса *DbInitializer.cs***

using Microsoft.EntityFrameworkCore;

using System.Text;

using TelecomWeb.Models;

namespace TelecomWeb.Data

{

public static class DbInitializer

{

public static void Initialize(TelecomDbContext db)

{

db.Database.EnsureCreated();

string sqlFilePath = "sql/insert.sql";

if (File.Exists(sqlFilePath))

{

string sql = File.ReadAllText(sqlFilePath, Encoding.UTF8);

db.Database.ExecuteSqlRaw(sql);

}

else

{

Console.WriteLine($"Файл не найден: {sqlFilePath}");

}

}

}

}

**Код перечисления *Role.cs***

namespace TelecomWeb.Data;

public enum Role

{

admin,

manager

}

**Код класса *DbInitializerMiddleware.cs***

using TelecomWeb.Data;

using TelecomWeb.Models;

namespace TelecomWeb.Middleware

{

public class DbInitializerMiddleware

{

private readonly RequestDelegate \_next;

public DbInitializerMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task Invoke(HttpContext context, TelecomDbContext dbContext)

{

var hasSubscribers = dbContext.Subscribers.Any();

if (!hasSubscribers)

{

DbInitializer.Initialize(dbContext);

}

await \_next(context);

}

}

public static class DbInitializerExtensions

{

public static IApplicationBuilder UseDbInitializer(this IApplicationBuilder builder)

{

return builder.UseMiddleware<DbInitializerMiddleware>();

}

}

}

**Код класса *Call.cs***

namespace TelecomWeb.Models;

public partial class Call

{

public int CallId { get; set; }

public int ContractId { get; set; }

public DateTime CallDate { get; set; }

public int CallDuration { get; set; }

public virtual Contract? Contract { get; set; } = null!;

}

**Код класса *Contract.cs***

namespace TelecomWeb.Models;

public partial class Contract

{

public int ContractId { get; set; }

public int SubscriberId { get; set; }

public int TariffPlanId { get; set; }

public DateOnly ContractDate { get; set; }

public DateOnly? ContractEndDate { get; set; }

public string PhoneNumber { get; set; } = null!;

public int StaffId { get; set; }

public virtual ICollection<Call> Calls { get; set; } = new List<Call>();

public virtual ICollection<InternetUsage> InternetUsages { get; set; } = new List<InternetUsage>();

public virtual ICollection<Message> Messages { get; set; } = new List<Message>();

public virtual Staff? Staff { get; set; } = null!;

public virtual Subscriber? Subscriber { get; set; } = null!;

public virtual TariffPlan? TariffPlan { get; set; } = null!;

}

**Код класса *InternetUsage.cs***

using System.ComponentModel.DataAnnotations;

namespace TelecomWeb.Models;

public partial class InternetUsage

{

public int UsageId { get; set; }

public int ContractId { get; set; }

public DateTime UsageDate { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid data sent value.")]

public decimal DataSentMb { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid data receive value.")]

public decimal DataReceivedMb { get; set; }

public virtual Contract? Contract { get; set; } = null!;

}

**Код класса *Message.cs***

namespace TelecomWeb.Models;

public partial class Message

{

public int MessageId { get; set; }

public int ContractId { get; set; }

public DateTime MessageDate { get; set; }

public bool IsMms { get; set; }

public virtual Contract? Contract { get; set; } = null!;

}

**Код класса *PaginatedList.cs***

namespace TelecomWeb.Models;

public class PaginatedList<T> : List<T>

{

public int PageIndex { get; private set; }

public int TotalPages { get; private set; }

public PaginatedList(List<T> items, int count, int pageIndex, int pageSize)

{

PageIndex = pageIndex;

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

this.AddRange(items);

}

public bool HasPreviousPage => PageIndex > 1;

public bool HasNextPage => PageIndex < TotalPages;

public static PaginatedList<T> Create(List<T> source, int pageIndex, int pageSize)

{

var count = source.Count;

var items = source.Skip((pageIndex - 1) \* pageSize).Take(pageSize).ToList();

return new PaginatedList<T>(items, count, pageIndex, pageSize);

}

}

**Код класса *Staff.cs***

namespace TelecomWeb.Models;

public partial class Staff

{

public int StaffId { get; set; }

public string FullName { get; set; } = null!;

public int PositionId { get; set; }

public string? Education { get; set; }

public virtual ICollection<Contract> Contracts { get; set; } = new List<Contract>();

public virtual StaffPosition? Position { get; set; } = null!;

}

**Код класса *StaffPosition.cs***

namespace TelecomWeb.Models;

public partial class StaffPosition

{

public int PositionId { get; set; }

public string PositionName { get; set; } = null!;

public virtual ICollection<Staff> Staff { get; set; } = new List<Staff>();

}

**Код класса *Subscriber.cs***

namespace TelecomWeb.Models;

public partial class Subscriber

{

public int SubscriberId { get; set; }

public string FullName { get; set; } = null!;

public string? HomeAddress { get; set; }

public string? PassportData { get; set; }

public virtual ICollection<Contract> Contracts { get; set; } = new List<Contract>();

}

**Код класса *SubscriberInfo.cs***

namespace TelecomWeb.Models;

public partial class SubscriberInfo

{

public int SubscriberId { get; set; }

public string SubscriberFullName { get; set; } = null!;

public string? HomeAddress { get; set; }

public string? PassportData { get; set; }

public int ContractId { get; set; }

public DateOnly ContractDate { get; set; }

public DateOnly? ContractEndDate { get; set; }

public string PhoneNumber { get; set; } = null!;

public string TariffName { get; set; } = null!;

}

**Код класса *TariffPlan.cs***

using System.ComponentModel.DataAnnotations;

namespace TelecomWeb.Models;

public partial class TariffPlan

{

public int TariffPlanId { get; set; }

public string TariffName { get; set; } = null!;

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid subscription fee.")]

public decimal SubscriptionFee { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid local call rate.")]

public decimal LocalCallRate { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid long distance call rate.")]

public decimal LongDistanceCallRate { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid international call rate.")]

public decimal InternationalCallRate { get; set; }

public bool IsPerSecond { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid SMS rate.")]

public decimal SmsRate { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid MMS rate.")]

public decimal MmsRate { get; set; }

[Range(0.0, double.MaxValue, ErrorMessage = "Please enter a valid data rate per MB.")]

public decimal DataRatePerMb { get; set; }

public virtual ICollection<Contract> Contracts { get; set; } = new List<Contract>();

}

**Код класса *TelecomDbContext.cs***

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

namespace TelecomWeb.Models;

public partial class TelecomDbContext : IdentityDbContext<IdentityUser>

//public partial class TelecomDbContext : DbContext

{

public TelecomDbContext()

{

}

public TelecomDbContext(DbContextOptions<TelecomDbContext> options)

: base(options)

{

}

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

public virtual DbSet<Contract> Contracts { get; set; }

public virtual DbSet<ContractCall> ContractCalls { get; set; }

public virtual DbSet<ContractInternetUsage> ContractInternetUsages { get; set; }

public virtual DbSet<ContractMessage> ContractMessages { get; set; }

public virtual DbSet<EmployeeInfo> EmployeeInfos { get; set; }

public virtual DbSet<InternetUsage> InternetUsages { get; set; }

public virtual DbSet<Message> Messages { get; set; }

public virtual DbSet<Staff> Staff { get; set; }

public virtual DbSet<StaffPosition> StaffPositions { get; set; }

public virtual DbSet<Subscriber> Subscribers { get; set; }

public virtual DbSet<SubscriberInfo> SubscriberInfos { get; set; }

public virtual DbSet<TariffPlan> TariffPlans { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.UseCollation("Cyrillic\_General\_CI\_AS");

modelBuilder.Entity<Call>(entity =>

{

entity.HasKey(e => e.CallId).HasName("PK\_\_Calls\_\_5180CF8A3D2D2ED2");

entity.Property(e => e.CallId).HasColumnName("CallID");

entity.Property(e => e.CallDate).HasColumnType("datetime");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.HasOne(d => d.Contract).WithMany(p => p.Calls)

.HasForeignKey(d => d.ContractId)

.HasConstraintName("FK\_\_Calls\_\_ContractI\_\_14B10FFA");

});

modelBuilder.Entity<Contract>(entity =>

{

entity.HasKey(e => e.ContractId).HasName("PK\_\_Contract\_\_C90D34095C346A02");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.PhoneNumber).HasMaxLength(20);

entity.Property(e => e.StaffId).HasColumnName("StaffID");

entity.Property(e => e.SubscriberId).HasColumnName("SubscriberID");

entity.Property(e => e.TariffPlanId).HasColumnName("TariffPlanID");

entity.HasOne(d => d.Staff).WithMany(p => p.Contracts)

.HasForeignKey(d => d.StaffId)

.HasConstraintName("FK\_\_Contracts\_\_Staff\_\_11D4A34F");

entity.HasOne(d => d.Subscriber).WithMany(p => p.Contracts)

.HasForeignKey(d => d.SubscriberId)

.HasConstraintName("FK\_\_Contracts\_\_Subsc\_\_0FEC5ADD");

entity.HasOne(d => d.TariffPlan).WithMany(p => p.Contracts)

.HasForeignKey(d => d.TariffPlanId)

.HasConstraintName("FK\_\_Contracts\_\_Tarif\_\_10E07F16");

});

modelBuilder.Entity<ContractCall>(entity =>

{

entity

.HasNoKey()

.ToView("ContractCalls");

entity.Property(e => e.CallDate).HasColumnType("datetime");

entity.Property(e => e.CallId).HasColumnName("CallID");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.PhoneNumber).HasMaxLength(20);

});

modelBuilder.Entity<ContractInternetUsage>(entity =>

{

entity

.HasNoKey()

.ToView("ContractInternetUsage");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.DataReceivedMb)

.HasColumnType("decimal(10, 2)")

.HasColumnName("DataReceivedMB");

entity.Property(e => e.DataSentMb)

.HasColumnType("decimal(10, 2)")

.HasColumnName("DataSentMB");

entity.Property(e => e.PhoneNumber).HasMaxLength(20);

entity.Property(e => e.UsageDate).HasColumnType("datetime");

entity.Property(e => e.UsageId).HasColumnName("UsageID");

});

modelBuilder.Entity<ContractMessage>(entity =>

{

entity

.HasNoKey()

.ToView("ContractMessages");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.IsMms).HasColumnName("IsMMS");

entity.Property(e => e.MessageDate).HasColumnType("datetime");

entity.Property(e => e.MessageId).HasColumnName("MessageID");

entity.Property(e => e.PhoneNumber).HasMaxLength(20);

});

modelBuilder.Entity<EmployeeInfo>(entity =>

{

entity

.HasNoKey()

.ToView("EmployeeInfo");

entity.Property(e => e.Education).HasMaxLength(100);

entity.Property(e => e.FullName).HasMaxLength(150);

entity.Property(e => e.PositionName).HasMaxLength(100);

entity.Property(e => e.StaffId).HasColumnName("StaffID");

});

modelBuilder.Entity<InternetUsage>(entity =>

{

entity.HasKey(e => e.UsageId).HasName("PK\_\_Internet\_\_29B197C0F4400AED");

entity.ToTable("InternetUsage");

entity.Property(e => e.UsageId).HasColumnName("UsageID");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.DataReceivedMb)

.HasColumnType("decimal(10, 2)")

.HasColumnName("DataReceivedMB");

entity.Property(e => e.DataSentMb)

.HasColumnType("decimal(10, 2)")

.HasColumnName("DataSentMB");

entity.Property(e => e.UsageDate).HasColumnType("datetime");

entity.HasOne(d => d.Contract).WithMany(p => p.InternetUsages)

.HasForeignKey(d => d.ContractId)

.HasConstraintName("FK\_\_InternetU\_\_Contr\_\_1A69E950");

});

modelBuilder.Entity<Message>(entity =>

{

entity.HasKey(e => e.MessageId).HasName("PK\_\_Messages\_\_C87C037C0167DB96");

entity.Property(e => e.MessageId).HasColumnName("MessageID");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.IsMms).HasColumnName("IsMMS");

entity.Property(e => e.MessageDate).HasColumnType("datetime");

entity.HasOne(d => d.Contract).WithMany(p => p.Messages)

.HasForeignKey(d => d.ContractId)

.HasConstraintName("FK\_\_Messages\_\_Contra\_\_178D7CA5");

});

modelBuilder.Entity<Staff>(entity =>

{

entity.HasKey(e => e.StaffId).HasName("PK\_\_Staff\_\_96D4AAF7835BD125");

entity.Property(e => e.StaffId).HasColumnName("StaffID");

entity.Property(e => e.Education).HasMaxLength(100);

entity.Property(e => e.FullName).HasMaxLength(150);

entity.Property(e => e.PositionId).HasColumnName("PositionID");

entity.HasOne(d => d.Position).WithMany(p => p.Staff)

.HasForeignKey(d => d.PositionId)

.HasConstraintName("FK\_\_Staff\_\_PositionI\_\_093F5D4E");

});

modelBuilder.Entity<StaffPosition>(entity =>

{

entity.HasKey(e => e.PositionId).HasName("PK\_\_StaffPos\_\_60BB9A59D4D3AD5C");

entity.ToTable("StaffPosition");

entity.Property(e => e.PositionId).HasColumnName("PositionID");

entity.Property(e => e.PositionName).HasMaxLength(100);

});

modelBuilder.Entity<Subscriber>(entity =>

{

entity.HasKey(e => e.SubscriberId).HasName("PK\_\_Subscrib\_\_7DFEB634B3799F70");

entity.Property(e => e.SubscriberId).HasColumnName("SubscriberID");

entity.Property(e => e.FullName).HasMaxLength(150);

entity.Property(e => e.HomeAddress).HasMaxLength(255);

entity.Property(e => e.PassportData).HasMaxLength(100);

});

modelBuilder.Entity<SubscriberInfo>(entity =>

{

entity

.HasNoKey()

.ToView("SubscriberInfo");

entity.Property(e => e.ContractId).HasColumnName("ContractID");

entity.Property(e => e.HomeAddress).HasMaxLength(255);

entity.Property(e => e.PassportData).HasMaxLength(100);

entity.Property(e => e.PhoneNumber).HasMaxLength(20);

entity.Property(e => e.SubscriberFullName).HasMaxLength(150);

entity.Property(e => e.SubscriberId).HasColumnName("SubscriberID");

entity.Property(e => e.TariffName).HasMaxLength(100);

});

modelBuilder.Entity<TariffPlan>(entity =>

{

entity.HasKey(e => e.TariffPlanId).HasName("PK\_\_TariffPl\_\_29A9282A1C614268");

entity.Property(e => e.TariffPlanId).HasColumnName("TariffPlanID");

entity.Property(e => e.DataRatePerMb)

.HasColumnType("decimal(10, 2)")

.HasColumnName("DataRatePerMB");

entity.Property(e => e.InternationalCallRate).HasColumnType("decimal(10, 2)");

entity.Property(e => e.LocalCallRate).HasColumnType("decimal(10, 2)");

entity.Property(e => e.LongDistanceCallRate).HasColumnType("decimal(10, 2)");

entity.Property(e => e.MmsRate).HasColumnType("decimal(10, 2)");

entity.Property(e => e.SmsRate).HasColumnType("decimal(10, 2)");

entity.Property(e => e.SubscriptionFee).HasColumnType("decimal(10, 2)");

entity.Property(e => e.TariffName).HasMaxLength(100);

});

OnModelCreatingPartial(modelBuilder);

}

partial void OnModelCreatingPartial(ModelBuilder modelBuilder);

}

**Код класса *IdentitySeedData.cs***

using Microsoft.AspNetCore.Identity;

using TelecomWeb.Data;

namespace TelecomWeb.Services

{

public class IdentitySeedData

{

private readonly UserManager<IdentityUser> \_userManager;

private readonly RoleManager<IdentityRole> \_roleManager;

private readonly IConfiguration \_configuration;

public IdentitySeedData(UserManager<IdentityUser> userManager, RoleManager<IdentityRole> roleManager, IConfiguration configuration)

{

\_userManager = userManager;

\_roleManager = roleManager;

\_configuration = configuration;

}

public void Initialize()

{

foreach (var roleName in Enum.GetNames(typeof(Role)))

if (!\_roleManager.RoleExistsAsync(roleName).Result)

\_roleManager.CreateAsync(new IdentityRole(roleName)).Wait();

var adminEmail = \_configuration["IdentitySeed:AdminEmail"];

var adminPassword = \_configuration["IdentitySeed:AdminPassword"];

if (string.IsNullOrEmpty(adminEmail) || string.IsNullOrEmpty(adminPassword))

{

throw new InvalidOperationException("Admin email or password is not configured in appsettings.json.");

}

var user = \_userManager.FindByNameAsync(adminEmail).Result;

if (user == null)

{

user = new IdentityUser

{

UserName = adminEmail,

Email = adminEmail

};

var createResult = \_userManager.CreateAsync(user, adminPassword).Result;

if (createResult.Succeeded)

{

\_userManager.AddToRoleAsync(user, Role.admin.ToString()).Wait();

}

else

{

throw new Exception($"Failed to create admin user: {string.Join(", ", createResult.Errors)}");

}

}

}

}

}

**Код класса *SessionExtensions.cs***

using System.Text.Json;

namespace TelecomWeb.Services

{

public static class SessionExtensions

{

public static void Set<T>(this ISession session, string key, T value)

{

session.SetString(key, JsonSerializer.Serialize(value));

}

public static T Get<T>(this ISession session, string key)

{

var value = session.GetString(key);

return value == null ? default : JsonSerializer.Deserialize<T>(value);

}

}

}

**Код класса *Program.cs***

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Middleware;

using TelecomWeb.Models;

using TelecomWeb.Services;

namespace TelecomWeb

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<TelecomDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("TelecomDatabase")));

builder.Services.AddMemoryCache();

builder.Services.AddDistributedMemoryCache();

builder.Services.AddSession();

builder.Services.AddIdentity<IdentityUser, IdentityRole>(options =>

{

options.Password.RequireDigit = false;

options.Password.RequiredLength = 4;

options.Password.RequireNonAlphanumeric = false;

options.Password.RequireUppercase = false;

options.Password.RequireLowercase = false;

})

.AddEntityFrameworkStores<TelecomDbContext>()

.AddDefaultUI()

.AddDefaultTokenProviders();

builder.Services.AddScoped<IdentitySeedData>();

builder.Services.AddControllersWithViews();

var app = builder.Build();

app.UseSession();

app.UseDbInitializer();

app.UseResponseCaching();

app.UseAuthentication();

app.UseAuthorization();

using (var scope = app.Services.CreateScope())

{

var initializer = scope.ServiceProvider.GetRequiredService<IdentitySeedData>();

initializer.Initialize();

}

if (!app.Environment.IsDevelopment())

{

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

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints => { endpoints.MapRazorPages(); });

app.MapControllerRoute(

name: "default",

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

app.Run();

}

}

}

**Код шаблона *Create.cshtml***

@model TelecomWeb.Models.Call

@{

ViewData["Title"] = "Создание";

}

<h1>Создание</h1>

<h4>Звонок</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<div class="form-group">

<label asp-for="ContractId" class="control-label">Дата</label>

<select asp-for="ContractId" class="form-control" asp-items="ViewBag.ContractName"></select>

</div>

<div class="form-group">

<label asp-for="CallDate" class="control-label">Продолжительность</label>

<input asp-for="CallDate" class="form-control" />

<span asp-validation-for="CallDate" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CallDuration" class="control-label">Номер телефона</label>

<input asp-for="CallDuration" class="form-control" />

<span asp-validation-for="CallDuration" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Создать" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Назад к списку</a>

</div>

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

**Код шаблона *Delete.cshtml***

@model TelecomWeb.Models.Call

@{

ViewData["Title"] = "Удаление";

}

<h1>Удаление</h1>

<h3>Вы уверены, что хотите удалить эту запись?</h3>

<div>

<h4>Вызов</h4>

<hr />

<dl class="row">

<dt class="col-sm-2">

Дата

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.CallDate)

</dd>

<dt class="col-sm-2">

Продолжительность

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.CallDuration)

</dd>

<dt class="col-sm-2">

Номер телефона

</dt>

<dd class="col-sm-10">

@Html.DisplayFor(model => model.Contract.PhoneNumber)

</dd>

</dl>

<form asp-action="Delete">

<input type="hidden" asp-for="CallId" />

<input type="submit" value="Удалить" class="btn btn-danger" /> |

<a asp-action="Index">Назад к списку</a>

</form>

</div>

**Код шаблона *Edit.cshtml***

@model TelecomWeb.Models.Call

@{

ViewData["Title"] = "Подробности";

}

<h1>Подробности</h1>

<div>

<h4>Звонок</h4>

<hr />

<dl class="row">

<dt class = "col-sm-2">

Дата

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.CallDate)

</dd>

<dt class = "col-sm-2">

Продолжительность

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.CallDuration)

</dd>

<dt class = "col-sm-2">

Номер телефона

</dt>

<dd class = "col-sm-10">

@Html.DisplayFor(model => model.Contract.PhoneNumber)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model?.CallId">Редактировать</a> |

<a asp-action="Index">Вернуться к списку</a>

</div>

**Код шаблона *Edit.cshtml***

@model TelecomWeb.Models.Call

@{

ViewData["Title"] = "Редактирование";

}

<h1>Редактирование</h1>

<h4>Звонок</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Edit">

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="CallId" />

<div class="form-group">

<label asp-for="ContractId" class="control-label">Дата</label>

<select asp-for="ContractId" class="form-control" asp-items="ViewBag.ContractName"></select>

<span asp-validation-for="ContractId" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CallDate" class="control-label">Продолжительность</label>

<input asp-for="CallDate" class="form-control" />

<span asp-validation-for="CallDate" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CallDuration" class="control-label">Номер телефона</label>

<input asp-for="CallDuration" class="form-control" />

<span asp-validation-for="CallDuration" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Сохранить" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a asp-action="Index">Назад к списку</a>

</div>

@section Scripts {

@{

await Html.RenderPartialAsync("\_ValidationScriptsPartial");

}

}

**Код шаблона *Index.cshtml***

@{

ViewData["Title"] = "Home Page";

}

<div class="text-center">

<h1 class="display-4">Welcome</h1>

<p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>

</div>

**Код шаблона *Privacy.cshtml***

@{

ViewData["Title"] = "Privacy Policy";

}

<h1>@ViewData["Title"]</h1>

<p>Use this page to detail your site's privacy policy.</p>

**Код шаблона *\_ViewImports.cshtml***

@using TelecomWeb

@using TelecomWeb.Models

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

**Код шаблона *\_ViewStart.cshtml***

@{

Layout = "\_Layout";

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**ЛИСТИНГ МОДУЛЬНЫХ ТЕСТОВ**

**Код класса *CallsControllerTests.cs***

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Xunit;

using TelecomWeb.Controllers.tables;

using TelecomWeb.Models;

namespace TelecomWeb.Tests.Controllers

{

public class CallsControllerTests : IDisposable

{

private readonly TelecomDbContext \_context;

private readonly CallsController \_controller;

public CallsControllerTests()

{

var options = new DbContextOptionsBuilder<TelecomDbContext>()

.UseInMemoryDatabase(databaseName: "TestDatabase")

.Options;

\_context = new TelecomDbContext(options);

\_controller = new CallsController(\_context);

InitializeData();

}

private void InitializeData()

{

var subscriber = new Subscriber

{

FullName = "Иванов Иван Иванович",

HomeAddress = "Улица Пушкина, дом 1",

PassportData = "1234 567890"

};

var tariffPlan = new TariffPlan

{

TariffName = "Тариф 1",

SubscriptionFee = 100,

LocalCallRate = 1,

LongDistanceCallRate = 2,

InternationalCallRate = 3,

IsPerSecond = true,

SmsRate = 0.5m,

MmsRate = 1,

DataRatePerMb = 0.01m

};

\_context.Subscribers.Add(subscriber);

\_context.TariffPlans.Add(tariffPlan);

\_context.SaveChanges();

var contract = new Contract

{

SubscriberId = subscriber.SubscriberId,

TariffPlanId = tariffPlan.TariffPlanId,

ContractDate = DateOnly.FromDateTime(DateTime.Now),

PhoneNumber = "9000000000"

};

\_context.Contracts.Add(contract);

\_context.SaveChanges();

}

[Fact]

public async Task Index\_ReturnsViewResult\_WithCalls()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

\_context.Calls.Add(call);

await \_context.SaveChangesAsync();

var result = await \_controller.Index(null, null, null);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<PaginatedList<Call>>(viewResult.Model);

Assert.Single(model);

}

[Fact]

public async Task Details\_ReturnsViewResult\_WithCall()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

\_context.Calls.Add(call);

await \_context.SaveChangesAsync();

var result = await \_controller.Details(call.CallId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<Call>(viewResult.Model);

Assert.Equal(call.CallId, model.CallId);

}

[Fact]

public async Task Details\_ReturnsNotFound\_WhenCallDoesNotExist()

{

var result = await \_controller.Details(999);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Create\_ReturnsRedirectToActionResult\_WhenModelIsValid()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

var result = await \_controller.Create(call);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

}

[Fact]

public async Task Create\_ReturnsViewResult\_WhenModelIsInvalid()

{

\_controller.ModelState.AddModelError("CallDuration", "Required");

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now

};

var result = await \_controller.Create(call);

var viewResult = Assert.IsType<ViewResult>(result);

Assert.Equal(call, viewResult.Model);

}

[Fact]

public async Task Edit\_ReturnsNotFound\_WhenCallDoesNotExist()

{

var result = await \_controller.Edit(999);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Edit\_ReturnsViewResult\_WithCall()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

\_context.Calls.Add(call);

await \_context.SaveChangesAsync();

var result = await \_controller.Edit(call.CallId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<Call>(viewResult.Model);

Assert.Equal(call.CallId, model.CallId);

}

[Fact]

public async Task Edit\_ReturnsRedirectToActionResult\_WhenModelIsValid()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

\_context.Calls.Add(call);

await \_context.SaveChangesAsync();

call.CallDuration = 130;

var result = await \_controller.Edit(call.CallId, call);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

}

[Fact]

public async Task Delete\_ReturnsViewResult\_WithCall()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

\_context.Calls.Add(call);

await \_context.SaveChangesAsync();

var result = await \_controller.Delete(call.CallId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<Call>(viewResult.Model);

Assert.Equal(call.CallId, model.CallId);

}

[Fact]

public async Task DeleteConfirmed\_ReturnsRedirectToActionResult\_WhenCallExists()

{

var call = new Call

{

ContractId = \_context.Contracts.First().ContractId,

CallDate = DateTime.Now,

CallDuration = 120

};

\_context.Calls.Add(call);

await \_context.SaveChangesAsync();

var result = await \_controller.DeleteConfirmed(call.CallId);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

Assert.Null(await \_context.Calls.FindAsync(call.CallId));

}

public void Dispose()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

**Код класса *InternetUsagesControllerTests.cs***

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Xunit;

using TelecomWeb.Controllers.tables;

using TelecomWeb.Models;

namespace TelecomWeb.Tests.Controllers

{

public class InternetUsagesControllerTests : IDisposable

{

private readonly TelecomDbContext \_context;

private readonly InternetUsagesController \_controller;

public InternetUsagesControllerTests()

{

var options = new DbContextOptionsBuilder<TelecomDbContext>()

.UseInMemoryDatabase(databaseName: "TestInternetUsageDatabase")

.Options;

\_context = new TelecomDbContext(options);

\_controller = new InternetUsagesController(\_context);

InitializeData();

}

private void InitializeData()

{

var subscriber = new Subscriber

{

FullName = "Иванов Иван Иванович",

HomeAddress = "Улица Пушкина, дом 1",

PassportData = "1234 567890"

};

var tariffPlan = new TariffPlan

{

TariffName = "Тариф 1",

SubscriptionFee = 100,

LocalCallRate = 1,

LongDistanceCallRate = 2,

InternationalCallRate = 3,

IsPerSecond = true,

SmsRate = 0.5m,

MmsRate = 1,

DataRatePerMb = 0.01m

};

\_context.Subscribers.Add(subscriber);

\_context.TariffPlans.Add(tariffPlan);

\_context.SaveChanges();

var contract = new Contract

{

SubscriberId = subscriber.SubscriberId,

TariffPlanId = tariffPlan.TariffPlanId,

ContractDate = DateOnly.FromDateTime(DateTime.Now),

PhoneNumber = "9000000000"

};

\_context.Contracts.Add(contract);

\_context.SaveChanges();

}

[Fact]

public async Task Index\_ReturnsViewResult\_WithInternetUsages()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

\_context.InternetUsages.Add(internetUsage);

await \_context.SaveChangesAsync();

var result = await \_controller.Index(null, null, null, null);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<PaginatedList<InternetUsage>>(viewResult.Model);

Assert.Single(model);

}

[Fact]

public async Task Details\_ReturnsViewResult\_WithInternetUsage()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

\_context.InternetUsages.Add(internetUsage);

await \_context.SaveChangesAsync();

var result = await \_controller.Details(internetUsage.UsageId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<InternetUsage>(viewResult.Model);

Assert.Equal(internetUsage.UsageId, model.UsageId);

}

[Fact]

public async Task Details\_ReturnsNotFound\_WhenInternetUsageDoesNotExist()

{

var result = await \_controller.Details(999);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Create\_ReturnsRedirectToActionResult\_WhenModelIsValid()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

var result = await \_controller.Create(internetUsage);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

}

[Fact]

public async Task Create\_ReturnsViewResult\_WhenModelIsInvalid()

{

\_controller.ModelState.AddModelError("DataSentMb", "Required");

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now

};

var result = await \_controller.Create(internetUsage);

var viewResult = Assert.IsType<ViewResult>(result);

Assert.Equal(internetUsage, viewResult.Model);

}

[Fact]

public async Task Edit\_ReturnsNotFound\_WhenInternetUsageDoesNotExist()

{

var result = await \_controller.Edit(999);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Edit\_ReturnsViewResult\_WithInternetUsage()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

\_context.InternetUsages.Add(internetUsage);

await \_context.SaveChangesAsync();

var result = await \_controller.Edit(internetUsage.UsageId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<InternetUsage>(viewResult.Model);

Assert.Equal(internetUsage.UsageId, model.UsageId);

}

[Fact]

public async Task Edit\_ReturnsRedirectToActionResult\_WhenModelIsValid()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

\_context.InternetUsages.Add(internetUsage);

await \_context.SaveChangesAsync();

internetUsage.DataSentMb = 600;

var result = await \_controller.Edit(internetUsage.UsageId, internetUsage);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

}

[Fact]

public async Task Delete\_ReturnsViewResult\_WithInternetUsage()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

\_context.InternetUsages.Add(internetUsage);

await \_context.SaveChangesAsync();

var result = await \_controller.Delete(internetUsage.UsageId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<InternetUsage>(viewResult.Model);

Assert.Equal(internetUsage.UsageId, model.UsageId);

}

[Fact]

public async Task DeleteConfirmed\_ReturnsRedirectToActionResult\_WhenInternetUsageExists()

{

var internetUsage = new InternetUsage

{

ContractId = \_context.Contracts.First().ContractId,

UsageDate = DateTime.Now,

DataSentMb = 500,

DataReceivedMb = 300

};

\_context.InternetUsages.Add(internetUsage);

await \_context.SaveChangesAsync();

var result = await \_controller.DeleteConfirmed(internetUsage.UsageId);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

Assert.Null(await \_context.InternetUsages.FindAsync(internetUsage.UsageId));

}

public void Dispose()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

**Код класса *MessagesControllerTests.cs***

using System;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Controllers.tables;

using TelecomWeb.Models;

using Xunit;

namespace TelecomWeb.Tests.Controllers

{

public class MessagesControllerTests : IDisposable

{

private readonly TelecomDbContext \_context;

private readonly MessagesController \_controller;

public MessagesControllerTests()

{

var options = new DbContextOptionsBuilder<TelecomDbContext>()

.UseInMemoryDatabase(databaseName: "TestMessagesDatabase")

.Options;

\_context = new TelecomDbContext(options);

\_controller = new MessagesController(\_context);

InitializeData();

}

private void InitializeData()

{

var subscriber = new Subscriber

{

FullName = "Иванов Иван Иванович",

HomeAddress = "Улица Пушкина, дом 1",

PassportData = "1234 567890"

};

var tariffPlan = new TariffPlan

{

TariffName = "Тариф 1",

SubscriptionFee = 100,

LocalCallRate = 1,

LongDistanceCallRate = 2,

InternationalCallRate = 3,

IsPerSecond = true,

SmsRate = 0.5m,

MmsRate = 1,

DataRatePerMb = 0.01m

};

\_context.Subscribers.Add(subscriber);

\_context.TariffPlans.Add(tariffPlan);

\_context.SaveChanges();

var contract = new Contract

{

SubscriberId = subscriber.SubscriberId,

TariffPlanId = tariffPlan.TariffPlanId,

ContractDate = DateOnly.FromDateTime(DateTime.Now),

PhoneNumber = "9000000000"

};

\_context.Contracts.Add(contract);

\_context.SaveChanges();

}

[Fact]

public async Task Index\_ReturnsViewResult\_WithMessages()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

var result = await \_controller.Index(null, null, null);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<PaginatedList<Message>>(viewResult.Model);

Assert.Single(model);

}

[Fact]

public async Task Details\_ReturnsViewResult\_WithMessage()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

var result = await \_controller.Details(message.MessageId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<Message>(viewResult.Model);

Assert.Equal(message.MessageId, model.MessageId);

}

[Fact]

public async Task Details\_ReturnsNotFound\_WhenMessageDoesNotExist()

{

var result = await \_controller.Details(999);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Create\_ReturnsRedirectToActionResult\_WhenModelIsValid()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

var result = await \_controller.Create(message);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

}

[Fact]

public async Task Create\_ReturnsViewResult\_WhenModelIsInvalid()

{

\_controller.ModelState.AddModelError("MessageDate", "Required");

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId

};

var result = await \_controller.Create(message);

var viewResult = Assert.IsType<ViewResult>(result);

Assert.Equal(message, viewResult.Model);

}

[Fact]

public async Task Edit\_ReturnsNotFound\_WhenMessageDoesNotExist()

{

var result = await \_controller.Edit(999);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Edit\_ReturnsViewResult\_WithMessage()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

var result = await \_controller.Edit(message.MessageId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<Message>(viewResult.Model);

Assert.Equal(message.MessageId, model.MessageId);

}

[Fact]

public async Task Edit\_ReturnsRedirectToActionResult\_WhenModelIsValid()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

message.IsMms = true;

var result = await \_controller.Edit(message.MessageId, message);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

}

[Fact]

public async Task Delete\_ReturnsViewResult\_WithMessage()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

var result = await \_controller.Delete(message.MessageId);

var viewResult = Assert.IsType<ViewResult>(result);

var model = Assert.IsAssignableFrom<Message>(viewResult.Model);

Assert.Equal(message.MessageId, model.MessageId);

}

[Fact]

public async Task DeleteConfirmed\_ReturnsRedirectToActionResult\_WhenMessageExists()

{

var message = new Message

{

ContractId = \_context.Contracts.First().ContractId,

MessageDate = DateTime.Now,

IsMms = false

};

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

var result = await \_controller.DeleteConfirmed(message.MessageId);

var redirectResult = Assert.IsType<RedirectToActionResult>(result);

Assert.Equal("Index", redirectResult.ActionName);

Assert.Null(await \_context.Messages.FindAsync(message.MessageId));

}

public void Dispose()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

**Код класса *SubscribersControllerTests.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Controllers.tables;

using TelecomWeb.Models;

using Xunit;

namespace TelecomWeb.Tests.Controllers

{

public class SubscribersControllerTests : IDisposable

{

private readonly TelecomDbContext \_context;

private readonly SubscribersController \_controller;

public SubscribersControllerTests()

{

var options = new DbContextOptionsBuilder<TelecomDbContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new TelecomDbContext(options);

\_controller = new SubscribersController(\_context);

}

[Fact]

public async Task Index\_ReturnsViewResult\_WithPaginatedList()

{

var subscribers = new List<Subscriber>

{

new Subscriber { SubscriberId = 1, FullName = "John Doe", HomeAddress = "123 Street" },

new Subscriber { SubscriberId = 2, FullName = "Jane Smith", HomeAddress = "456 Avenue" }

};

await \_context.Subscribers.AddRangeAsync(subscribers);

await \_context.SaveChangesAsync();

var result = await \_controller.Index(null, null, null, null, 1) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.IsType<PaginatedList<Subscriber>>(result.Model);

Assert.Equal(2, ((PaginatedList<Subscriber>)result.Model).Count);

}

[Fact]

public async Task Details\_ReturnsNotFound\_WhenIdIsNull()

{

var result = await \_controller.Details(null, null, null);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Details\_ReturnsViewResult\_WithSubscriber()

{

var subscriber = new Subscriber { SubscriberId = 1, FullName = "John Doe", HomeAddress = "123 Street" };

await \_context.Subscribers.AddAsync(subscriber);

await \_context.SaveChangesAsync();

var result = await \_controller.Details(1, null, null) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.Equal(subscriber, result.Model);

}

[Fact]

public async Task Create\_RedirectsToIndex\_OnValidModel()

{

var subscriber = new Subscriber { SubscriberId = 1, FullName = "John Doe" };

var result = await \_controller.Create(subscriber) as RedirectToActionResult;

Assert.Equal("Index", result.ActionName);

Assert.Contains(subscriber, \_context.Subscribers);

}

[Fact]

public async Task Create\_ReturnsView\_OnInvalidModel()

{

\_controller.ModelState.AddModelError("Name", "Required");

var subscriber = new Subscriber();

var result = await \_controller.Create(subscriber) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.Equal(subscriber, result.Model);

}

[Fact]

public async Task Edit\_ReturnsNotFound\_WhenIdsDoNotMatch()

{

var subscriber = new Subscriber { SubscriberId = 1, FullName = "John Doe" };

var result = await \_controller.Edit(2, subscriber);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Edit\_RedirectsToIndex\_OnValidModel()

{

var subscriber = new Subscriber { SubscriberId = 1, FullName = "John Doe" };

await \_context.Subscribers.AddAsync(subscriber);

await \_context.SaveChangesAsync();

subscriber.FullName = "Updated Name";

var result = await \_controller.Edit(1, subscriber) as RedirectToActionResult;

Assert.Equal("Index", result.ActionName);

Assert.Equal("Updated Name", \_context.Subscribers.First().FullName);

}

[Fact]

public async Task DeleteConfirmed\_RedirectsToIndex\_OnSuccess()

{

var subscriber = new Subscriber { SubscriberId = 1, FullName = "John Doe" };

await \_context.Subscribers.AddAsync(subscriber);

await \_context.SaveChangesAsync();

var result = await \_controller.DeleteConfirmed(1) as RedirectToActionResult;

Assert.Equal("Index", result.ActionName);

Assert.DoesNotContain(subscriber, \_context.Subscribers);

}

[Fact]

public async Task Delete\_ReturnsNotFound\_WhenIdIsNull()

{

var result = await \_controller.Delete(null);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Delete\_ReturnsViewResult\_WithSubscriber()

{

var subscriber = new Subscriber { SubscriberId = 1, FullName = "John Doe" };

await \_context.Subscribers.AddAsync(subscriber);

await \_context.SaveChangesAsync();

var result = await \_controller.Delete(1) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.Equal(subscriber, result.Model);

}

public void Dispose()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

**Код класса *TariffPlansControllerTests.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using TelecomWeb.Controllers.tables;

using TelecomWeb.Models;

using Xunit;

namespace TelecomWeb.Tests.Controllers

{

public class TariffPlansControllerTests : IDisposable

{

private readonly TelecomDbContext \_context;

private readonly TariffPlansController \_controller;

public TariffPlansControllerTests()

{

var options = new DbContextOptionsBuilder<TelecomDbContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

\_context = new TelecomDbContext(options);

\_controller = new TariffPlansController(\_context);

}

[Fact]

public async Task Index\_ReturnsViewResult\_WithPaginatedList()

{

var tariffPlans = new List<TariffPlan>

{

new TariffPlan { TariffPlanId = 1, TariffName = "Basic", SubscriptionFee = 10.0m },

new TariffPlan { TariffPlanId = 2, TariffName = "Premium", SubscriptionFee = 20.0m }

};

await \_context.TariffPlans.AddRangeAsync(tariffPlans);

await \_context.SaveChangesAsync();

var result = await \_controller.Index(1, null, null, null, null, null) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.IsType<PaginatedList<TariffPlan>>(result.Model);

Assert.Equal(2, ((PaginatedList<TariffPlan>)result.Model).Count);

}

[Fact]

public async Task Details\_ReturnsNotFound\_WhenIdIsNull()

{

var result = await \_controller.Details(null);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Details\_ReturnsViewResult\_WithTariffPlan()

{

var tariffPlan = new TariffPlan { TariffPlanId = 1, TariffName = "Basic", SubscriptionFee = 10.0m };

await \_context.TariffPlans.AddAsync(tariffPlan);

await \_context.SaveChangesAsync();

var result = await \_controller.Details(1) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.Equal(tariffPlan, result.Model);

}

[Fact]

public async Task Create\_RedirectsToIndex\_OnValidModel()

{

var tariffPlan = new TariffPlan { TariffPlanId = 1, TariffName = "Basic", SubscriptionFee = 10.0m };

var result = await \_controller.Create(tariffPlan) as RedirectToActionResult;

Assert.Equal("Index", result.ActionName);

Assert.Contains(tariffPlan, \_context.TariffPlans);

}

[Fact]

public async Task Create\_ReturnsView\_OnInvalidModel()

{

\_controller.ModelState.AddModelError("TariffName", "Required");

var tariffPlan = new TariffPlan();

var result = await \_controller.Create(tariffPlan) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.Equal(tariffPlan, result.Model);

}

[Fact]

public async Task Edit\_ReturnsNotFound\_WhenIdsDoNotMatch()

{

var tariffPlan = new TariffPlan { TariffPlanId = 1, TariffName = "Basic" };

var result = await \_controller.Edit(2, tariffPlan);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Edit\_RedirectsToIndex\_OnValidModel()

{

var tariffPlan = new TariffPlan { TariffPlanId = 1, TariffName = "Basic", SubscriptionFee = 10.0m };

await \_context.TariffPlans.AddAsync(tariffPlan);

await \_context.SaveChangesAsync();

tariffPlan.TariffName = "Updated Tariff";

var result = await \_controller.Edit(1, tariffPlan) as RedirectToActionResult;

Assert.Equal("Index", result.ActionName);

Assert.Equal("Updated Tariff", \_context.TariffPlans.First().TariffName);

}

[Fact]

public async Task DeleteConfirmed\_RedirectsToIndex\_OnSuccess()

{

var tariffPlan = new TariffPlan { TariffPlanId = 1, TariffName = "Basic", SubscriptionFee = 10.0m };

await \_context.TariffPlans.AddAsync(tariffPlan);

await \_context.SaveChangesAsync();

var result = await \_controller.DeleteConfirmed(1) as RedirectToActionResult;

Assert.Equal("Index", result.ActionName);

Assert.DoesNotContain(tariffPlan, \_context.TariffPlans);

}

[Fact]

public async Task Delete\_ReturnsNotFound\_WhenIdIsNull()

{

var result = await \_controller.Delete(null);

Assert.IsType<NotFoundResult>(result);

}

[Fact]

public async Task Delete\_ReturnsViewResult\_WithTariffPlan()

{

var tariffPlan = new TariffPlan { TariffPlanId = 1, TariffName = "Basic" };

await \_context.TariffPlans.AddAsync(tariffPlan);

await \_context.SaveChangesAsync();

var result = await \_controller.Delete(1) as ViewResult;

Assert.IsType<ViewResult>(result);

Assert.Equal(tariffPlan, result.Model);

}

public void Dispose()

{

\_context.Database.EnsureDeleted();

\_context.Dispose();

}

}

}

**ПРИЛОЖЕНИЕ В**

(обязательное)

**ФУНКЦИОНАЛЬНАЯ СХЕМА ПРИЛОЖЕНИЯ**