Содержание

[Введение 4](#_Toc184398000)

[1 Анализ предметной области и постановка задачи 6](#_Toc184398001)

[1.1 Анализ предметной области 6](#_Toc184398002)

[1.2 Постановка задачи 6](#_Toc184398003)

[2 Разработка структуры страниц 8](#_Toc184398004)

[2.1 Выбор и обоснование метода построение структуры страницы 8](#_Toc184398005)

[2.2 Выбор и обоснование доступа к содержимому страницы 8](#_Toc184398006)

[2.3 Разработка логической структуры главной страницы 9](#_Toc184398007)

[3 Разработка страниц 10](#_Toc184398008)

[3.1 Разработка шапки страницы 10](#_Toc184398009)

[3.2 Разработка меню навигации по сайту 10](#_Toc184398010)

[3.3 Настройка компонентов взаимодействия с базой данных 11](#_Toc184398011)

[4 Разработка структуры приложения и динамического содержимого страниц 13](#_Toc184398012)

[4.1 Разработка каскадных таблиц стилей CSS 13](#_Toc184398013)

[4.2 Разработка бизнес-логики работы с базой данных 17](#_Toc184398014)

[5 Тестирование сайта 22](#_Toc184398015)

[5.1 Отчёт о тестировании 22](#_Toc184398016)

[5.2 Руководство пользователя 23](#_Toc184398017)

[Заключение 27](#_Toc184398018)

[Список литературы 28](#_Toc184398019)

# Введение

Эффективное управление личными финансами является ключом к достижению финансовой стабильности и благополучия. Это веб-приложение предназначено для помощи пользователям в учете, анализе и отслеживании их личных доходов, расходов, сбережений и инвестиций. Используя современные технологии, оно обеспечит интуитивно понятный и отзывчивый интерфейс, позволяющий принимать обоснованные финансовые решения.

ASP.NET MVC — это мощный фреймворк для разработки веб-приложений, основанный на архитектуре Model-View-Controller. Он позволяет разделить логику приложения на три основных компонента: модели, представления и контроллеры, что упрощает управление кодом и повышает его тестируемость.

На стороне бэкенда основное внимание уделяется созданию моделей, которые представляют данные и бизнес-логику, а также контроллерам, которые обрабатывают запросы пользователей и взаимодействуют с моделями. Важным аспектом является использование Entity Framework для работы с базами данных, что позволяет эффективно управлять данными и выполнять операции CRUD.

Приложение работает с базой данных SSMS. Работа с базами данных в SQL Server Management Studio (SSMS) включает в себя множество возможностей. Есть возможность подключаться к серверу, управлять базами данных, выполнять SQL-запросы и работать с таблицами. SSMS позволяет создавать и изменять базы данных, а также выполнять операции импорта и экспорта данных

Настройка маршрутизации позволяет определить, как обрабатывать различные URL-запросы, а система валидации помогает обеспечить корректность вводимых данных. Кроме того, можно реализовать аутентификацию и авторизацию пользователей, чтобы защитить приложение. Это особенно важно для финансовых приложений, где безопасность данных пользователей имеет критическое значение.

Для улучшения пользовательского опыта приложение может включать функции визуализации данных, такие как графики и диаграммы, которые помогут пользователям лучше понимать свои финансовые потоки и принимать обоснованные решения. Также полезно интегрировать функции напоминаний и уведомлений, чтобы пользователи могли следить за своими платежами и сроками сбережений.

Таким образом, ASP.NET MVC предоставляет разработчикам все необходимые инструменты для создания эффективных и масштабируемых веб-приложений. Финансовое приложение предоставит пользователям удобный и надежный инструмент для учета, анализа и управления их личными финансами, что в конечном итоге способствует более осознанному финансовому поведению и достижению долгосрочных финансовых целей.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

Интернет-приложения для учёта финансовых средств стали необходимым инструментом для многих пользователей, желающих эффективно управлять своим бюджетом. Эти приложения предлагают широкий спектр возможностей, которые помогают пользователям отслеживать доходы и расходы, планировать бюджет и анализировать финансовые данные.

При анализе предметной области интернет-приложений для учёта финансовых средств необходимо учитывать не только их преимущества, но и существующие проблемы. Одной из ключевых проблем является безопасность данных. Пользователи часто беспокоятся о том, как их финансовая информация будет защищена от несанкционированного доступа. Это требует от разработчиков внедрения современных технологий для обеспечения безопасности.

Перспективы развития данной области выглядят многообещающими. С увеличением финансовой грамотности населения и растущим интересом к личным финансам, спрос на такие приложения будет только расти. В дальнейшем можно ожидать появления новых функций, таких как искусственный интеллект для анализа финансовых данных и предсказания финансового поведения, а также улучшение интеграции с другими сервисами и платформами.

Таким образом, анализ предметной области интернет-приложений для учёта финансовых средств показывает, что, несмотря на существующие недостатки, рынок продолжает развиваться и адаптироваться к потребностям пользователей.

Создание интернет-приложения для учета и анализа финансовых средств значительно упростит процесс управления личными финансами. Удобный учет, возможность анализа финансовых потоков и персонализированные графики сделают работу с приложением более эффективной и приятной для пользователей. Это позволит повысить финансовую осведомленность и способствует более обоснованным финансовым решениям.

1.2 Постановка задачи

При разработке интернет-приложений для учета и анализа финансов важнейшую роль играет бэкенд, который отвечает за обработку данных, бизнес-логику и взаимодействие с базой данных. Основная цель — создание надежной и масштабируемой серверной части, способной обрабатывать запросы пользователей и обеспечивать безопасность данных.

Одной из ключевых задач является проектирование структуры базы данных. Необходимо определить основные сущности, такие как пользователи, транзакции, категории расходов и доходов, а также установить связи между ними. Для хранения данных можно использовать реляционные базы данных, для данного приложения решено использовать MS SQL.

Для реализации серверной части рекомендуется использовать ASP.NET MVC, который обеспечивает четкое разделение логики приложения на модели, представления и контроллеры. Это упрощает разработку и тестирование, а также позволяет легко поддерживать код.

Необходимо также обеспечить интеграцию с базой данных. Использование Entity Framework позволит работать с реляционными базами данных, такими как MS SQL, и упростит процесс миграции схемы данных. Это поможет поддерживать целостность и актуальность данных.

Не менее важна безопасность приложения. Реализация механизма аутентификации и авторизации пользователей поможет защитить данные от несанкционированного доступа. Необходимо внедрить механизмы аутентификации и авторизации пользователей. ASP.NET Identity предоставляет готовые решения для управления пользователями и их правами доступа, что позволяет легко интегрировать безопасные методы входа в приложение.

2 Разработка структуры страниц

2.1 Выбор и обоснование метода построение структуры страницы

При разработке интернет-приложений для учета и анализа финансов одним из наиболее эффективных методов построения структуры страниц является использование Razor Pages в ASP.NET. Этот подход позволяет разработчикам создавать динамические и интерактивные веб-страницы с использованием кода C#, что значительно упрощает процесс отображения сложных объектов и управления данными.

Для передачи данных между сервером и клиентом Razor Pages предлагает несколько подходов, среди которых наиболее распространены ViewModel, ViewBag и ViewData.

ViewModel – это наиболее предпочтительный способ передачи данных, так как он обеспечивает строгую типизацию и ясность. ViewModel представляет собой класс, который содержит все необходимые данные, которые должны быть переданы на страницу.

ViewBag – это динамическое свойство, которое позволяет передавать данные с сервера на клиент без необходимости создания отдельного класса.

ViewData – это еще один способ передачи данных, который использует словарь для хранения информации в формате ключ-значение.

Таким образом, основным способом построение структуры страниц стало использование Razor Pages, которые предоставляли возможность использования языка C#, также для построения структуры страницы использовались данные, которые передавались с помощью ViewModel, ViewBag и ViewData.

2.2 Выбор и обоснование доступа к содержимому страницы

В процессе создания веб-приложения для учета и анализа финансов необходимо учитывать важный аспект — метод доступа к содержимому страниц, основанный на ролях пользователей. Такой подход позволяет эффективно управлять правами доступа в зависимости от уровня полномочий каждого пользователя, что особенно важно для финансовых приложений, где защита данных является критически важной.

На начальном этапе следует установить различные роли пользователей, такие как администратор, обычный пользователь. Каждая из этих ролей будет иметь свои специфические права доступа к определенным частям приложения.

Внедрение механизма контроля доступа позволяет ограничивать доступ к определенному контенту в зависимости от роли пользователя. Это помогает защищать конфиденциальные данные и предотвращает несанкционированный доступ.

Применение ролей также упрощает управление правами доступа. При необходимости добавления новых функций или ролей достаточно внести изменения в настройки, не затрагивая основной код приложения. Это обеспечивает высокую гибкость и масштабируемость, позволяя легко адаптировать приложение к изменяющимся требованиям бизнеса.

Для администратора будет предоставлен доступ к категориям. Управление этими данными будет осуществляться с помощью соответствующих кнопок на странице. Администратор сможет просматривать все категории, а также добавлять новые и редактировать уже имеющиеся. При добавлении будет открываться новая страница для заполнения соответствующих полей. Администратору будет предоставлена возможность получения отчёта по всем пользователям.

Таким образом, определён доступ к содержимому страницы для администратора.

2.3 Разработка логической структуры главной страницы

Для главной страницы администратора необходимо тщательно продумать ее логическую структуру. Это обеспечит администратору интуитивно понятный интерфейс и легкий доступ ко всем необходимым функциям. Основные компоненты структуры включают следующие элементы:

1. Заголовок. Заголовок страницы должен содержать название страницы, которое отображает, что будет на ней происходить.

2. Подзаголовки. Они будут отображать элементы для анализа такие, как категории доходов и расходов.

3. Карточки элементов. Будут отображать всю информацию о категории включая изображение, id, название категории.

4. Кнопки управления. Кнопки для управления содержимым, то есть кнопки для создания новой категории дохода или расхода или их редактирования. Также на страницы будет находится кнопка для создания отчёта по пользователям.

Таким образом, определена логическая структура страницы администратора.

3 Разработка страниц

## 3.1 Разработка шапки страницы

Шапка сайта – это титульная верхняя часть страницы, в которой располагаются элементы навигации (меню) и элементы, содержащие информацию об интернет-приложении.

Рекомендации по разработке шапки:

1. Структура и элементы: шапка обычно включает в себя название приложения, логотип и навигационное меню. Эти элементы помогают пользователю быстро ориентироваться в приложении и переходить к основным разделам.

2. Доступность: важно, чтобы элементы шапки были доступны пользователям на всех страницах приложения. Это может быть достигнуто с помощью макетов или частичных представлений в ASP.NET, которые обеспечивают единый стиль и функциональность.

При разработке приложения было решено добавить шапку на страницы для работы администратора и пользователя. В шапку было решено добавить навигационное меню и логотип приложения. В результате разработки была создана шапка для страниц администратора (рисунок 1).

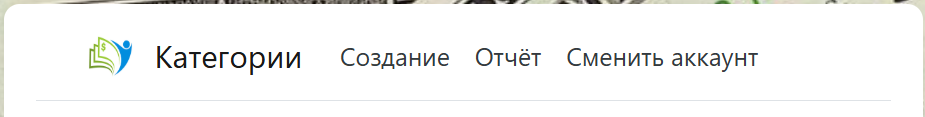


Рисунок 1 – Шапка страниц администратора

Таким образом, была создана шапка для страниц администратора, которая включает в себя логотип приложения и навигационные элементы.

## 3.2 Разработка меню навигации по сайту

Меню навигации является одним из ключевых элементов пользовательского интерфейса интернет-приложения, так как оно обеспечивает пользователям легкий доступ ко всем основным разделам и функциям. Меню навигации должно быть логично структурировано и интуитивно понятным. Оно должно быть доступным на всех страницах сайта. Меню навигации должно быть визуально привлекательным и соответствовать общему стилю приложения. Использование четких шрифтов, контрастных цветов и интуитивных иконок поможет сделать меню более понятным и удобным для пользователей.

В рамках интернет-приложения для анализа и учёта финансовых средств было создано меню навигации, которое находится в шапке страниц. Для администратора были созданы кнопки навигации:

1. Категории – переход на главную страницу администратора;

2. Создание – переход на страницу для создания новой категории дохода или расхода;

3. Отчёт – При нажатии на элемент происходит загрузка отчёта по всем пользователям.

4. Сменить аккаунт – осуществляет переход на страницу входа.

Таким образом, администраторы приложения будут иметь возможность передвигаться по сайту с помощью навигационного меню, что упрощает взаимодействие с приложением и делает переход между страницами более удобным.

## 3.3 Настройка компонентов взаимодействия с базой данных

При разработке интернет-приложения для учета и анализа финансов важным аспектом является настройка компонентов, которые обеспечивают взаимодействие с базой данных. Это включает реализацию кнопок для выполнения операций и отображение данных на страницах.

Кнопки в пользовательском интерфейсе играют ключевую роль в обеспечении интерактивности приложения. Основные действия, которые пользователи могут выполнять с данными, включают создание, редактирование и удаление записей.

В рамках части администратора интернет-приложения для учёта и анализа финансовых средств будут добавлены такие элементы управления, как кнопка для добавления категории дохода или расхода на выбор и кнопка для редактирования определённой категории.

Также для удобства некоторые страницы работы с БД вынесены в навигационное меню.

Кроме того, будет добавлена кнопка для получения отчёта по всем пользователям. Тем самым будет реализована работа с базой данных клиентов.

Для отображения данных из БД на странице используется Razor Pages в ASP.NET. Данные могут быть получены из базы данных и переданы в представление, где они будут динамически отображаться.

Для просмотра данных, которые имеются в БД будут отображены карточки с категориями. На карточке будет вся информация о категории: её id, название, изображение.

Таким образом, продумана реализация компонентов для работы с базой данных.

4 Разработка структуры приложения и динамического содержимого страниц

4.1 Разработка каскадных таблиц стилей CSS

Каскадные таблицы стилей (CSS) являются важным инструментом для оформления и стилизации веб-приложений. Они позволяют разработчикам эффективно управлять внешним видом элементов на страницах, обеспечивая при этом гибкость и согласованность дизайна. В этом разделе рассмотрим основные аспекты разработки каскадных таблиц стилей, которые помогут создать визуально привлекательное и удобное приложение для учета и анализа финансов.

Для оформления страниц приложения были созданы таблицы стилей. Основной файл содержал оформление body, my-container, h1, а также оформление кнопок по умолчанию:

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: #f8f9fa;

color: #343a40;

margin: 0;

padding: 20px;

background-image: url('/images/siteBackgroungImg.jpg');

background-size: cover;

}

.my-container {

max-width: 50%;

margin: auto;

background-color: #ffffff;

border-radius: 10px;

box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);

padding-bottom: 20px;

padding-right: 20px;

padding-left: 20px;

padding-top: 5px;

}

h1 {

text-align: center;

color: green;

position: relative;

}

button {

background-color: green;

color: white;

}

button:hover {

background-color: #0056b3;

}

При разработке главной страницы администратора были добавлены стили для новых элементов, а также изменена стилизация старых элементов: button, button:hover.

Добавлены новые стили:

card – для отображения карточки категории;

card-img-top – для отображения изображения категории;

card-text – для отображения id

card-title – для отображения названия категории;

my-add-button – для отображения кнопки для добавления;

my-add-button img – для отображения изображения кнопки для добавления;

edit-button, edit-button img – для отображения кнопки для редактирования определённой категории;

report-button – для отображения кнопки создания отчёта;

card-deck – для отображения карточек определённой категории;

button {

background-color: white;

color: white;

}

button:hover {

background-color: white;

}

h2 {

display: flex;

justify-content: space-between;

align-items: center;

}

.card {

border: 1px solid #e0e0e0;

border-radius: 10px;

margin-bottom: 10px;

text-align: center;

}

.card-img-top {

width: 70px;

height: 70px;

margin: 5px auto;

border-radius: 50%;

}

.card-text {

margin-top: 0;

font-weight: bold;

font-size: 1em;

margin-bottom: 0px;

margin-top: 0px;

flex: auto;

}

.card-title {

font-weight: bold;

font-size: 1.2em;

margin-bottom: 0px;

margin-top: 0px;

}

.my-add-button {

background: none;

border: none;

cursor: pointer;

margin-left: 5px;

}

.my-add-button img {

width: 30px;

height: 30px;

}

.edit-button {

width: 25px;

height: 25px;

background: none;

border: none;

cursor: pointer;

padding: 0;

margin: 0;

}

.edit-button img {

width: 25px;

height: 25px;

display: block;

}

.report-button {

width: 150px;

height: 40px;

background-color: green;

color: white;

border: none;

border-radius: 5px;

padding: 10px 15px;

font-weight: bold;

font-size: 15px;

cursor: pointer;

transition: background-color 0.3s;

}

.report-button:hover {

background-color: #0056b3;

}

.card-deck {

display: flex;

flex-wrap: wrap;

justify-content: space-between;

}

.card-deck .card {

flex: 1 1 calc(48% - 10px);

margin: 5px;

min-width: 150px;

}

При разработке страницы для добавления новых категорий добавлены стили для полей ввода:

.form-group {

margin-bottom: 15px;

}

label {

font-weight: bold;

}

input[type="text"], select {

width: 100%;

padding: 10px;

border: 1px solid #ced4da;

border-radius: 5px;

}

input[type="file"] {

padding: 10px;

}

.img-thumbnail {

margin-top: 10px;

border: 1px solid #ced4da;

border-radius: 5px;

}

4.2 Разработка бизнес-логики работы с базой данных

При создании интернет-приложения для учета и анализа финансов важным этапом является разработка бизнес-логики, которая управляет взаимодействием с базой данных. Бизнес-логика определяет, как данные будут обрабатываться, хранятся и извлекаться, а также какие правила и условия будут применяться к этому процессу.

Для администратора реализовано чтение записей о категориях:

[HttpGet]

public async Task<IActionResult> Categories()

{

var categories = await \_context.Categories

.Include(c => c.CategoryType)

.ToListAsync();

return View(categories);

}

Администратор может добавлять новую информацию о категориях:

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(CreateCategoryViewModel model)

{

if (ModelState.IsValid)

{

string imageUrl = "/images/noImg.png";

if (model.ImageFile != null && model.ImageFile.Length > 0)

{

var fileName = Path.GetFileName(model.ImageFile.FileName);

var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images", fileName);

using (var stream = new FileStream(filePath, FileMode.Create))

{

await model.ImageFile.CopyToAsync(stream);

}

imageUrl = $"/images/{fileName}";

}

var category = new Category

{

Name = model.CategoryName,

ImageUrl = imageUrl,

CategoryTypeId = model.CategoryTypeId

};

\_context.Categories.Add(category);

try

{

await \_context.SaveChangesAsync();

return RedirectToAction("Categories");

}

catch (Exception ex)

{

ModelState.AddModelError("", "Ошибка при сохранении категории. Подробности: " + ex.Message);

}

}

var categories = await \_context.CategoryTypes.ToListAsync();

ViewBag.CategoryTypes = categories.Select(ct => new SelectListItem

{

Value = ct.CategoryTypeId.ToString(),

Text = ct.TypeName

}).ToList();

return View(model);

}

Администратор может редактировать информацию о категориях:

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(EditCategoryViewModel model)

{

if (ModelState.IsValid)

{

var category = await \_context.Categories.FindAsync(model.CategoryId);

if (category == null)

{

return NotFound();

}

category.Name = model.CategoryName;

category.CategoryTypeId = model.CategoryTypeId;

if (model.ImageFile != null && model.ImageFile.Length > 0)

{

var fileName = Path.GetFileName(model.ImageFile.FileName);

var filePath = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot/images", fileName);

using (var stream = new FileStream(filePath, FileMode.Create))

{

await model.ImageFile.CopyToAsync(stream);

}

category.ImageUrl = $"/images/{fileName}";

}

else

{

category.ImageUrl = model.CurrentImageUrl;

}

try

{

await \_context.SaveChangesAsync();

return RedirectToAction("Categories");

}

catch (Exception ex)

{

ModelState.AddModelError("", "Ошибка при обновлении категории. Подробности: " + ex.Message);

}

}

var categoriesList = await \_context.CategoryTypes.ToListAsync();

ViewBag.CategoryTypes = categoriesList.Select(ct => new SelectListItem

{

Value = ct.CategoryTypeId.ToString(),

Text = ct.TypeName

}).ToList();

return View(model);

}

При создании отчёта данные фильтруются:

[HttpGet]

public async Task<IActionResult> CreateReport()

{

var users = \_userManager.Users.ToList();

var reportData = new List<UserReport>();

foreach (var user in users)

{

var incomes = await \_context.Incomes

.Where(i => i.UserId == user.Id)

.Include(i => i.Category)

.ToListAsync();

var expenses = await \_context.Expenses

.Where(e => e.UserId == user.Id)

.Include(e => e.Category)

.ToListAsync();

var totalIncome = incomes.Sum(i => i.Amount);

var totalExpense = expenses.Sum(e => e.Amount);

var balance = totalIncome - totalExpense;

var highestIncomeCategory = incomes

.GroupBy(i => i.Category != null ? i.Category.Name : "Нет категории")

.Select(g => new

{

Category = g.Key,

TotalAmount = g.Sum(i => i.Amount)

})

.OrderByDescending(g => g.TotalAmount)

.FirstOrDefault();

var highestExpenseCategory = expenses

.GroupBy(e => e.Category != null ? e.Category.Name : "Нет категории")

.Select(g => new

{

Category = g.Key,

TotalAmount = g.Sum(e => e.Amount)

})

.OrderByDescending(g => g.TotalAmount)

.FirstOrDefault();

reportData.Add(new UserReport

{

UserName = user.UserName,

TotalIncome = totalIncome,

IncomeCategory = highestIncomeCategory?.Category ?? "Нет данных",

TotalExpense = totalExpense,

ExpenseCategory = highestExpenseCategory?.Category ?? "Нет данных",

Balance = balance

});

}

using (var workbook = new XLWorkbook())

{

var worksheet = workbook.Worksheets.Add("Отчет");

worksheet.Cell(1, 1).Value = "Пользователь";

worksheet.Cell(1, 2).Value = "Доход за всё время";

worksheet.Cell(1, 3).Value = "Сфера наибольшего дохода";

worksheet.Cell(1, 4).Value = "Расход за всё время";

worksheet.Cell(1, 5).Value = "Сфера наибольшего расхода";

worksheet.Cell(1, 6).Value = "Остаток денег";

int row = 2;

foreach (var data in reportData)

{

worksheet.Cell(row, 1).Value = data.UserName;

worksheet.Cell(row, 2).Value = data.TotalIncome;

worksheet.Cell(row, 3).Value = data.IncomeCategory;

worksheet.Cell(row, 4).Value = data.TotalExpense;

worksheet.Cell(row, 5).Value = data.ExpenseCategory;

worksheet.Cell(row, 6).Value = data.Balance;

row++;

}

var filePath = Path.Combine(Directory.GetCurrentDirectory(), "Reports", "AllUsers\_Report.xlsx");

workbook.SaveAs(filePath);

var fileBytes = System.IO.File.ReadAllBytes(filePath);

return File(fileBytes, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "AllUsers\_Report.xlsx");

}

}

# 5 Тестирование сайта

5.1 Отчёт о тестировании

После разработки программного модуля было проведено его тестирование. Результаты тестирования представлены в таблице 1.

Таблица 1 – Тестирование разработанной программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п/п | | Описание теста | Ожидаемый результат | Результат тестирования |
| 1 | | 2 | 3 | 4 |
| 1. Регистрация | | | |  |
| 1.1 | | Нажатие на ссылку «Нет аккаунта? Загеристрироваться» на странице входа в систему | Загрузится страница регистрации пользователя | + |
| 1.2 | | В форме для регистрации поле не заполнено и нажата кнопка зарегистрироваться | Появилась подсказка под незаполненным полем    Загрузка главной страницы не произошла | + |
| 1.3 | В поле для ввода email введён некорректный адрес электронной почты и нажата кнопка зарегистрироваться | | Появилась подсказка    Загрузка главной страницы не произошла | + |
| 1.4 | Введён пароль неверного формата | | Появилось предупреждение | + |
| 1.5 | Введены корректные данные и нажата кнопка зарегистрироваться | | Загрузится Главная страница приложения | + |
| 2. Добавление новой категории | | | |  |
| 2.1 | Нажатие на кнопку добавления новой категории на странице | | Открывается страница для добавления новой категории | + |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 2.2 | Не заполнено название категории | Отображается предупреждение | + |
| 2.3 | Не выбран файл картинки | Отображается предупреждение | + |
| 2.4 | Все поля заполнены и нажата кнопка «Создать категорию» | Добавлена новая категория и открывается главная страничка администратора | + |
| 3. Редактирование категории | | | |
| 3.1 | Нажатие кнопки редактирование на элементе | Открывается страница для редактирования | + |
| 3.2 | Не введено название | Отображается сообщение:    Категория не отредактирована | + |
| 3.3 | Данные введены и нажата кнопка «Сохранить» | Данные отредактированы. Осуществляется переход на главную страницу. | + |
| 4. Создание отчёта | | | |
| 4.1 | Администратор нажимает кнопку «Создать отчёт» на главной странице или «Отчёт» в меню | Формируется отчёт и скачивается: | + |

Таким образом, были протестированы основные функции приложения.

5.2 Руководство пользователя

Если пользователя нет в системе, ему необходимо зарегистрироваться. Для этого пользователь нажимает на ссылку «Нет аккаунта? Регистрация». После этого открывается страница для регистрации (рисунок 2), где пользователь заполняет поля.

После успешной регистрации или успешном входе в систему пользователь попадает на главную страницу приложения. Главная страница Администратора представлена на рисунке 3.

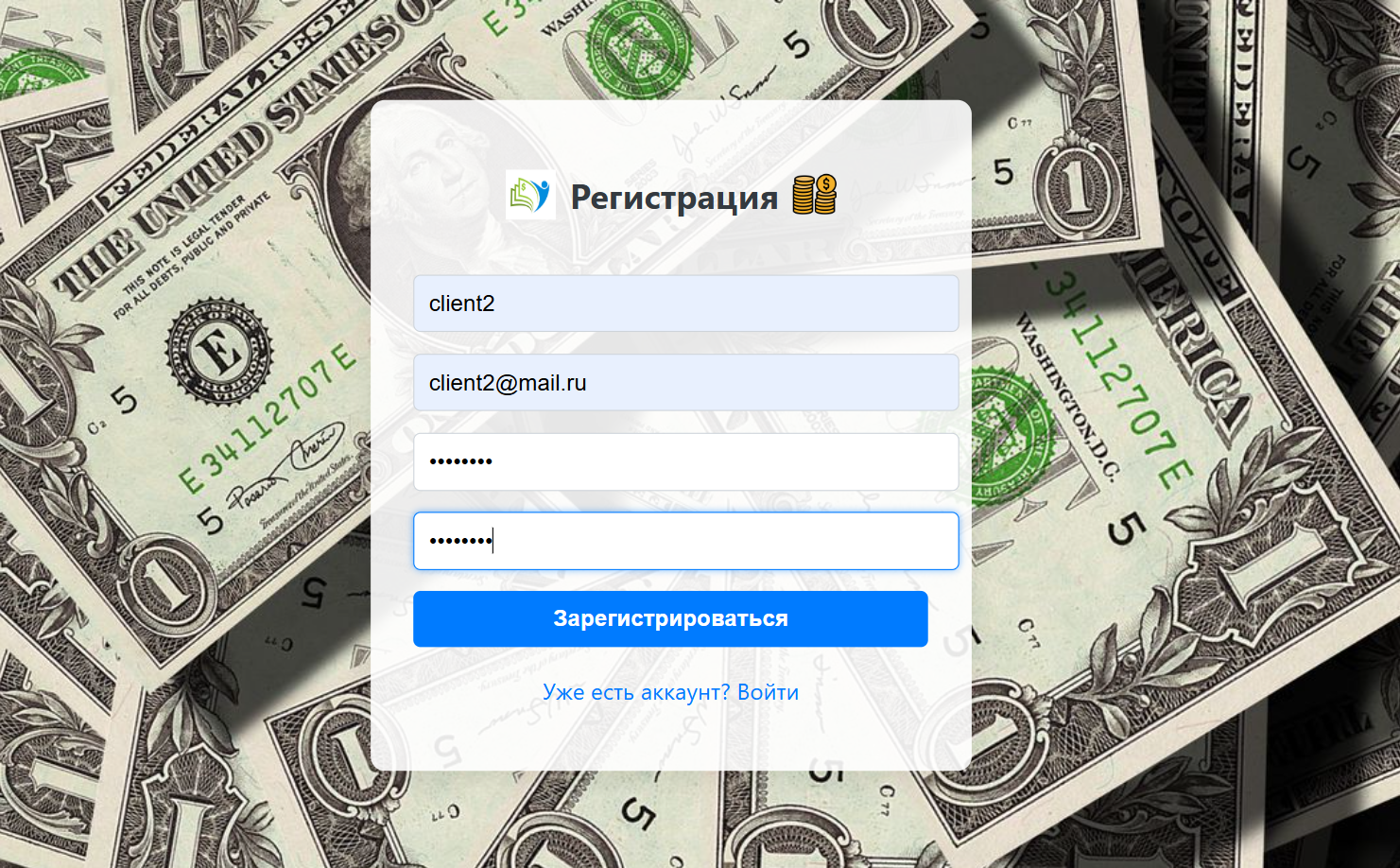


Рисунок 2 – Регистрация пользователя

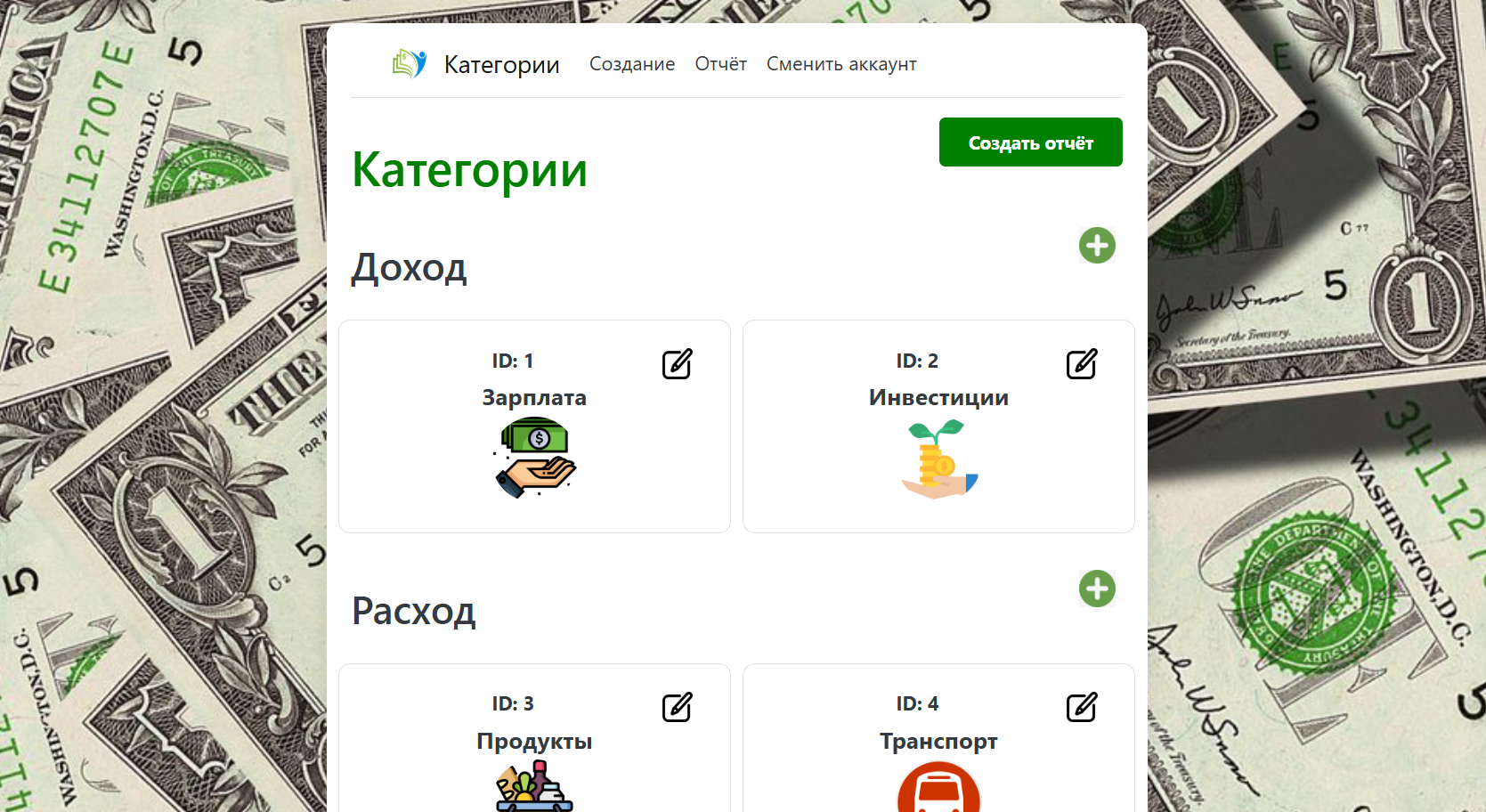


Рисунок 3 – Главная страница администратора

Для добавления новой категории Администратору необходимо щёлкнуть по кнопке (рисунок 4) или по элементу меню «Создание» откроется страница (рисунок 5), на которой необходимо заполнить поля и нажать на кнопку «Создать категорию».



Рисунок 4 – Кнопка добавления новой категории

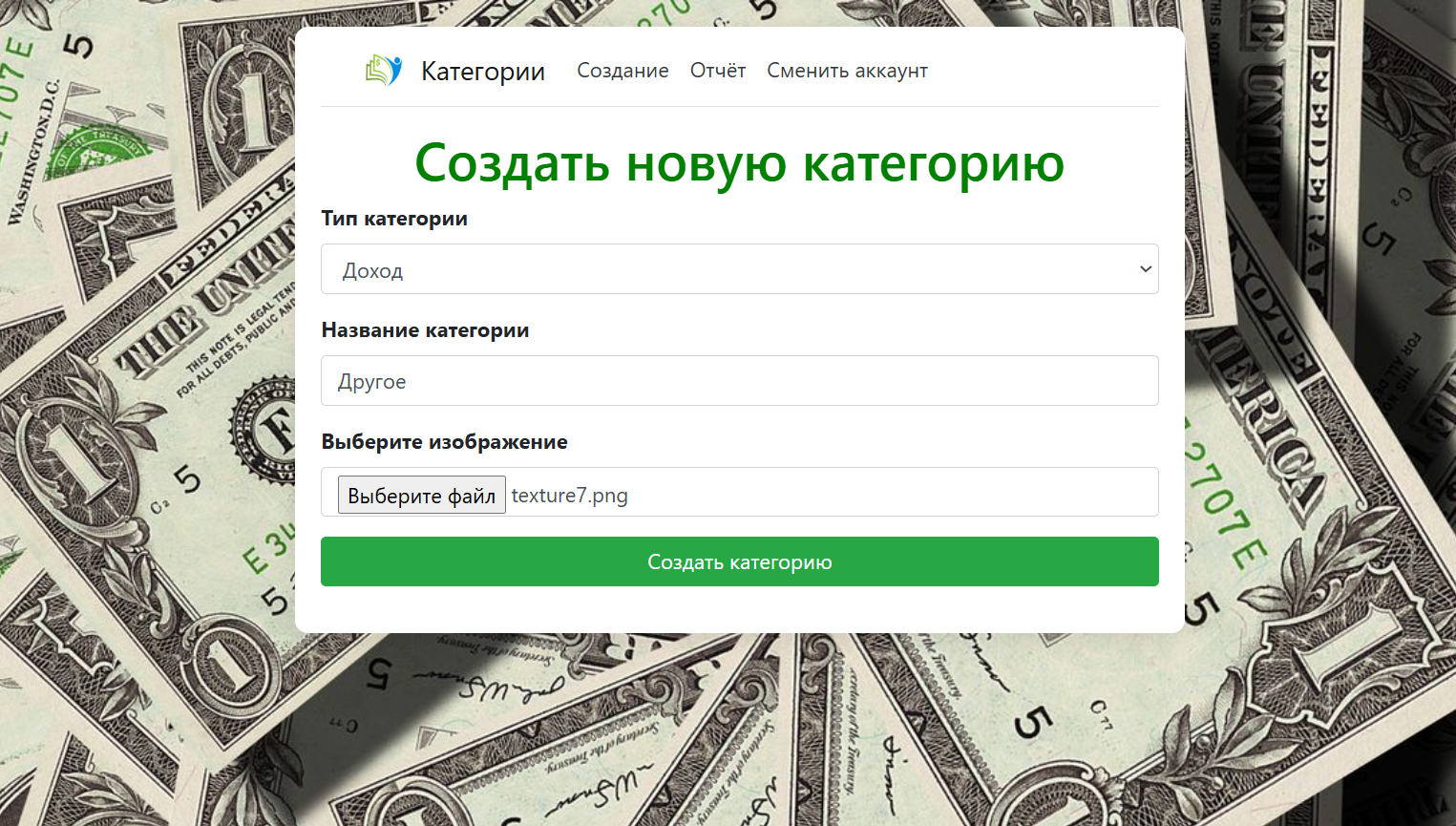


Рисунок 5 – Страница для добавления новой категории

Для редактирования категории необходимо нажать по кнопке редактирования (рисунок 6). После этого откроется страница для внесения изменений (рисунок 7). Необходимо заполнить на ней данные и нажать на кнопку «Сохранить». Изменения внесутся.



Рисунок 6 – Кнопка для редактирования категории

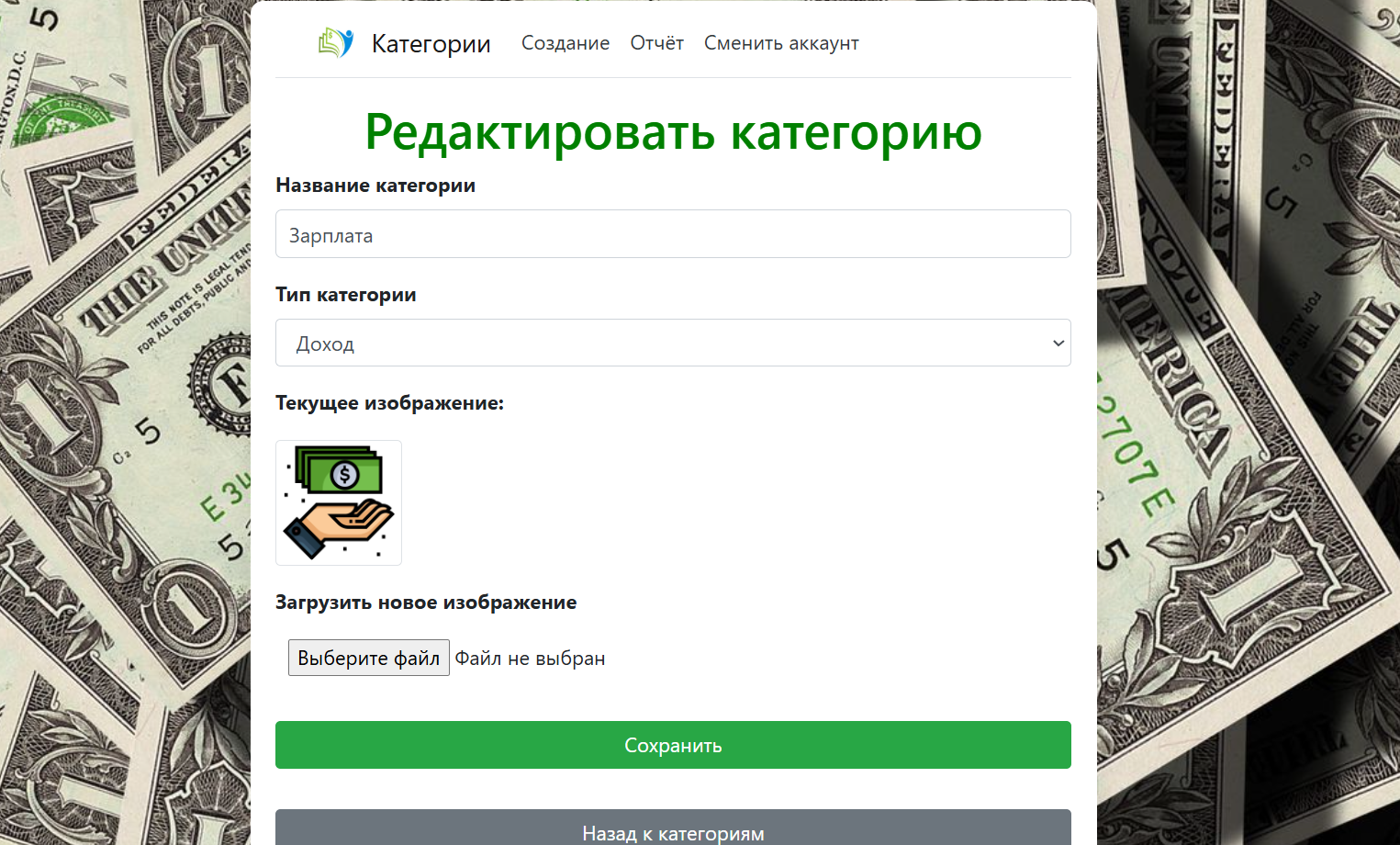


Рисунок 7 – Страница для редактирования категории

Для формирования отчёта на главной странице необходимо нажать «Создать отчёт» или по элементу меню «Отчёт», после этого он будет скачан (рисунок 8).

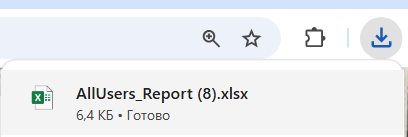


Рисунок 8 – Отчёт скачан

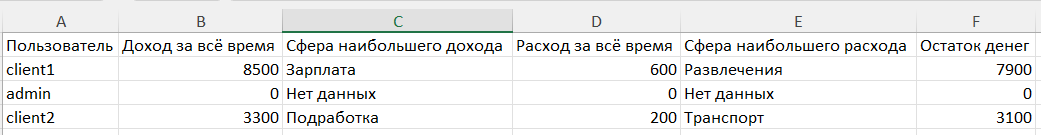


Рисунок 9 – Содержимое отчёта

# Заключение

В заключение, разработка веб-приложения для учета и анализа личных финансов с использованием ASP.NET MVC представляет собой важный шаг к обеспечению финансовой грамотности и стабильности пользователей. Интуитивно понятный интерфейс и надежная архитектура делают это приложение мощным инструментом для управления финансами.

Приложение обеспечит удобство использования и функциональность, включая возможности добавления и учета записей, а также детального анализа финансовых потоков. ASP.NET MVC предоставляет мощные инструменты для реализации механизмов аутентификации и авторизации, что критически важно для защиты конфиденциальной информации пользователей. Безопасность данных становится приоритетом в финансовых приложениях, и использование современного фреймворка позволяет эффективно реализовать эти меры.

Работа с базами данных через Entity Framework и SQL Server Management Studio позволяет не только эффективно управлять данными, но и обеспечивает высокую производительность операций CRUD. Это гарантирует, что пользователи будут иметь доступ к актуальной информации о своих финансах в любое время. Кроме того, настройка маршрутизации и система валидации данных обеспечивают корректную обработку запросов и защиту от ошибок ввода.

Таким образом, данное приложение не только упрощает процесс учета финансов, но и способствует более осознанному подходу к ним, помогая пользователям достигать их финансовых целей и обеспечивая долгосрочное благополучие. Эффективная архитектура backend, построенная на ASP.NET MVC, позволяет создавать масштабируемые и надежные решения, которые могут адаптироваться к растущим потребностям пользователей и технологическим изменениям.

# Список литературы

1. **Алекс, Д.** Асинхронное программирование в C# 5.0**,** 2015 – 120 с.

2. **Вагнер, Б.** Эффективное программирование на C#. 50 способов улучшения кода, 2017 – 224 с.

3. **Полуэктова, Н. Р.** - Разработка веб-приложений. — М.: Юрайт, 2024. — 205 c.

4. **Симан,** М. Внедрение зависимостей в .NET. — М.: ТНТ, 2021. — 534 c.

5. **Нагаева, И. А.; Фролов, А. Б.; Кузнецов, И. А.** Основы web-дизайна. Методика проектирования. — М.: Директ-Медиа, 2021. — 184 c.

6. **Фриман, А., Робинс, А. –** Pro ASP.NET Core MVC 2 / Алан Фриман, Джереми Робинс. – М.: Аспресс, 2021. — 672 с.

7. **Пустохина, И.** – ASP.NET Core MVC: Руководство для начинающих / Ирина Пустохина. – М.: Паккт Паблишинг, 2021. — 350 с.