Содержание

[Введение 4](#_Toc184397907)

[1 Анализ предметной области и постановка задачи 6](#_Toc184397908)

[1.1 Анализ предметной области 6](#_Toc184397909)

[1.2 Постановка задачи 6](#_Toc184397910)

[2 Разработка структуры страниц 8](#_Toc184397911)

[2.1 Выбор и обоснование метода построение структуры страницы 8](#_Toc184397912)

[2.2 Выбор и обоснование доступа к содержимому страницы 8](#_Toc184397913)

[2.3 Разработка логической структуры главной страницы 9](#_Toc184397914)

[3 Разработка страниц 11](#_Toc184397915)

[3.1 Разработка шапки страницы 11](#_Toc184397916)

[3.2 Разработка меню навигации по сайту 11](#_Toc184397917)

[3.3 Настройка компонентов взаимодействия с базой данных 12](#_Toc184397918)

[4 Разработка структуры приложения и динамического содержимого страниц 13](#_Toc184397919)

[4.1 Разработка каскадных таблиц стилей CSS 13](#_Toc184397920)

[4.2 Разработка бизнес-логики работы с базой данных 17](#_Toc184397921)

[5 Тестирование сайта 20](#_Toc184397922)

[5.1 Отчёт о тестировании 20](#_Toc184397923)

[5.2 Руководство пользователя 21](#_Toc184397924)

[Заключение 26](#_Toc184397925)

[Список литературы 27](#_Toc184397926)

# Введение

Современные интернет-приложения для учета и анализа финансовых средств становятся неотъемлемой частью повседневной жизни как для частных пользователей. В условиях постоянного роста финансовой грамотности и растущего интереса к эффективному управлению деньгами, такие приложения предоставляют пользователю мощные инструменты для мониторинга доходов, расходов и составления финансовых отчетов. Удобство и функциональность интерфейса играют решающую роль в успехе таких решений, позволяя легко и быстро анализировать данные и принимать обоснованные финансовые решения.

Одним из таких решений является веб-приложение, построенное с использованием архитектуры MVC (Model-View-Controller). Эта архитектура эффективно разделяет бизнес-логику, данные и пользовательский интерфейс, что способствует упрощению разработки, тестирования и поддерживаемости приложения. Важнейшим компонентом в контексте этой архитектуры является frontend, который отвечает за взаимодействие пользователя с приложением и его визуальное представление. Использование frontend-технологий, таких как HTML, CSS и JavaScript, позволяет создавать интуитивно понятные, динамичные и визуально привлекательные интерфейсы.

HTML — это язык разметки, который используется для структурирования содержания веб-страниц. Он определяет элементы, такие как заголовки, абзацы, списки, ссылки и изображения, с помощью тегов. HTML создает основную структуру страницы и позволяет браузерам отображать контент.CSS — это язык стилей, который отвечает за визуальное оформление веб-страниц. С помощью CSS можно изменять цвет, шрифт, размеры, отступы и расположение элементов на странице. CSS позволяет создавать адаптивные и отзывчивые дизайны, которые выглядят хорошо на различных устройствах. JavaScript — это язык программирования, который добавляет интерактивность и динамичность на веб-страницы. С его помощью можно управлять элементами на странице, обрабатывать события (например, клики и ввод данных), а также взаимодействовать с сервером для загрузки данных без перезагрузки страницы. JavaScript позволяет создавать сложные веб-приложения, такие как игры, формы и интерактивные интерфейсы. HTML, CSS и JavaScript работают вместе, чтобы создать полноценный пользовательский опыт. Эти три технологии являются основой веб-разработки и позволяют создавать современные, функциональные и привлекательные веб-приложения.

Сочетание этих технологий с архитектурой MVC позволяет создать мощное и гибкое приложение, которое будет легко масштабировать и модифицировать. В результате пользователи получат не только эффективный инструмент для учета своих финансов, но и удобный интерфейс, который будет интуитивно понятен и доступен для пользователей с разным уровнем технической подготовки.

1 Анализ предметной области и постановка задачи

1.1 Анализ предметной области

Анализ предметной области интернет-приложений для учёта финансовых средств показывает, что в последние годы наблюдается значительный рост интереса к таким инструментам как со стороны индивидуальных пользователей, так и со стороны малых и средних предприятий. Это связано с необходимостью более эффективного управления финансами.

Основная структура таких приложений включает несколько ключевых компонентов. Во-первых, это интерфейс пользователя, который должен быть интуитивно понятным и доступным. Пользователи ожидают, что они смогут быстро вводить данные о доходах и расходах, а также получать отчёты о финансовом состоянии. Интерфейсы интернет-приложений обычно интуитивно понятны, что делает их доступными для пользователей с разным уровнем финансовой грамотности.

Во-вторых, важной частью является система анализа данных. Она должна поддерживать создание графиков, которые позволяют пользователям визуализировать свои финансовые потоки и выявлять тенденции. Это способствует более осознанному финансовому планированию. Графики и диаграммы в реальном времени позволяют быстро оценивать свои финансовые результаты и выявлять тенденции в расходах.

Таким образом, на приложения для анализа финансов есть спрос, а также интерес у пользователей.

1.2 Постановка задачи

Разработка интернет-приложений для учета и анализа финансов требует четкой постановки задач, особенно на этапе проектирования frontend. Основная цель — создать интуитивно понятный и удобный интерфейс, который позволит пользователям без особых усилий управлять своими финансами.

Также важна реализация эффективной навигации. Пользователи должны легко находить необходимые функции, такие как добавление доходов и расходов, создание отчетов и анализ финансовых данных. Для этого необходимо продумать структуру меню и взаимодействие с пользователем.

Кроме того, следует обратить внимание на визуализацию данных. Графики и диаграммы помогают пользователям лучше понять свои финансовые потоки.

При разработке frontend интернет-приложения для учета и анализа финансов в рамках MVC ASP.NET следует учитывать множество технических требований. Приложение должно быть высокопроизводительным и безопасным, что требует тщательной проработки архитектуры и выбора инструментов.

Для реализации пользовательского интерфейса рекомендуется использовать Razor Views, которые позволяют динамически генерировать HTML-код на основе данных модели. Это упрощает интеграцию между фронтендом и бэкендом, обеспечивая плавное взаимодействие между компонентами приложения.

Кроме того, необходимо реализовать функционал для визуализации данных. Графики и диаграммы помогут пользователям лучше понять свои финансовые потоки. Для этого можно использовать JavaScript-библиотеки, такие как Chart.js.

2 Разработка структуры страниц

2.1 Выбор и обоснование метода построение структуры страницы

В процессе создания веб-приложений для учета и анализа финансов одним из самых действенных способов организации структуры страниц является применение Razor Pages в ASP.NET. Этот метод предоставляет разработчикам возможность создавать динамичные и интерактивные веб-страницы с помощью кода C#, что существенно упрощает задачу отображения сложных объектов и управления данными.

Основные преимущества Razor Pages:

1. Интуитивно понятная структура;

2. Интеграция C# и HTML: Razor позволяет использовать C# код непосредственно в HTML-разметке. Это позволяет разработчикам легко добавлять динамическое содержимое, например, выводить список транзакций, используя циклы, такие как for или foreach, для перебора коллекций объектов.

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, использование Razor Pages в ASP.NET для построения структуры страниц предоставляет разработчикам мощный инструмент для создания динамичных веб-приложений. Возможность интеграции C# кода и HTML разметки делает разработку более гибкой и удобной. Применение ViewModel, ViewBag и ViewData позволяет эффективно управлять передачей данных между сервером и клиентом, обеспечивая четкую и структурированную работу с информацией. Это сочетание методов позволяет создать интуитивно понятный и функциональный интерфейс, что особенно важно для приложений, связанных с учетом и анализом финансов.

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. В рамках ASP.NET MVC это можно реализовать с помощью атрибутов авторизации, таких как [Authorize], которые применяются к контроллерам или отдельным методам. 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 клиента будет предоставлен доступ для управления своими финансами. Клиент сможет просматривать свои доходы и расходы. Управления доходами и расходами будет осуществляться с помощью соответствующих кнопок, клиенты смогут удалять и добавлять доходы и расходы. Кроме того, клиенту будет предоставлена возможность просмотра графиков по доходам и расходам.

Таким образом, определён доступ и взаимодействие с содержимым страницы для клиента.

2.3 Разработка логической структуры главной страницы

При создании главной страницы интернет-приложения для учета и анализа финансов необходимо тщательно продумать ее логическую структуру. Это обеспечит пользователю 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 доходы и расходы.

3. Карточки элементов. Будут отображать всю информацию о доходе или расходе, включая дату, сумму, категорию, название категории, описание.

4. Кнопки управления. Кнопки для управления содержимым, то есть кнопки для удаления, создания нового дохода или расхода. Также на страницы будут находиться кнопки для перехода к просмотру графиков.

Таким образом, определена логическая структура страницы пользователя.

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 клиента (рисунок 1).

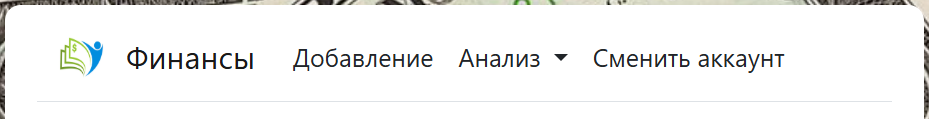


Рисунок 1 – Шапка страниц клиента

Таким образом, была создана шапка для страниц клиента, 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 клиента были созданы кнопки навигации:

1. Финансы – переход на главную страницу клиента;

2. Добавление – переход на страницу для добавления записи о доходе или расходе;

3. Анализ – элемент раскрывается, при этом пользователь может выбрать доход или расход. При нажатии осуществляется переход на страницу для отображения графика.

4. Сменить аккаунт – осуществляет переход на страницу входа.

Таким образом, клиенты 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 клиентской части интернет-приложения для учёта и анализа финансовых средств будут добавлены такие элементы управления, как кнопка для добавления информации о доходе или расходе на выбор и кнопка для удаления определённой записи.

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 записи с информацией о доходе или расходе. Запись будет содержать нужную информацию: изображение, название, описание, дату, сумму.

Также для анализа данных будет реализован просмотр графиков, графики будут создаваться по данным из БД.

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 клиента были добавлены стили для новых элементов, а также изменена стилизация старых элементов: button, button:hover.

Добавлены новые стили:

date – для отображения даты получения дохода или расхода;

add-button, add-button img – для отображения кнопки добавления дохода или расхода;

chart-button, chart-button img – для отображения кнопки просмотра графика;

item-list, item-list li, item-list li:last-child – для отображения записи дохода или расхода;

category-image – для отображения изображения категории дохода или расхода;

category-image – для отображения изображения категории дохода или расхода;

category-info – для отображения информации о категории;

category-info – для отображения информации о категории;

amount – для отображения суммы дохода или расхода;

delete-button, delete-button img – для отображения кнопки удаления определённой записи;

total – для отображения общего баланса клиента.

button {

background-color: white;

color: white;

}

button:hover {

background-color: white;

}

.date {

display: block;

margin-top: 5px;

font-size: 0.9em;

color: #6c757d;

}

.add-button {

position: absolute;

right: 20px;

top: 10px;

background: none;

border: none;

cursor: pointer;

}

.add-button img {

width: 30px;

height: 30px;

}

.chart-button {

background: none;

border: none;

cursor: pointer;

margin-left: 5px;

}

.chart-button img {

width: 25px;

height: 25px;

}

.item-list {

list-style-type: none;

padding: 0;

}

.item-list li {

display: flex;

align-items: flex-start;

justify-content: space-between;

padding: 10px;

border-bottom: 1px solid #e9ecef;

}

.item-list li:last-child {

border-bottom: none;

}

.category-image {

width: 45px;

height: 45px;

margin-right: 10px;

border-radius: 70%;

}

.category-info {

display: flex;

flex-direction: column;

}

.category-info span:first-child {

color: #000;

font-weight: bold;

}

.amount {

margin-left: auto;

font-weight: bold;

}

.delete-button {

background: none;

border: none;

cursor: pointer;

padding: 0;

margin-left: 10px;

}

.delete-button img {

width: 23px;

height: 23px;

}

.total {

font-weight: bold;

margin-top: 20px;

font-size: 1.2em;

}

При разработке страницы для добавления новых записей добавлены стили для form-group label для отображения жирным шрифтом label:

.form-group label {

font-weight: bold;

}

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 клиента реализовано чтение записей о доходах и расходах:

public async Task<IActionResult> Index()

{

var user = await \_userManager.GetUserAsync(User);

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 viewModel = new HomeViewModel

{

Incomes = incomes,

Expenses = expenses

};

return View(viewModel);

}

Также клиент может удалять данные:

[HttpPost]

public IActionResult DeleteIncome(int id)

{

var income = \_context.Incomes.Find(id);

if (income != null && income.UserId == \_userManager.GetUserId(User))

{

\_context.Incomes.Remove(income);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

[HttpPost]

public IActionResult DeleteExpense(int id)

{

var expense = \_context.Expenses.Find(id);

if (expense != null)

{

\_context.Expenses.Remove(expense);

\_context.SaveChanges();

}

return RedirectToAction("Index");

}

Для клиента предоставлена возможность добавления данных:

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(CreateTransactionViewModel model)

{

if(model.Description == null)

{

model.Description = " ";

}

var user = await \_userManager.GetUserAsync(User);

if (model.IsIncome)

{

var income = new Income

{

Amount = model.Amount,

Date = model.Date,

Description = model.Description,

CategoryId = model.CategoryId,

UserId = user.Id

};

\_context.Incomes.Add(income);

}

else

{

var expense = new Expense

{

Amount = model.Amount,

Date = model.Date,

Description = model.Description,

CategoryId = model.CategoryId,

UserId = user.Id

};

\_context.Expenses.Add(expense);

}

try

{

await \_context.SaveChangesAsync();

return RedirectToAction("Index");

}

catch (Exception ex)

{

ModelState.AddModelError("", "Ошибка при сохранении данных в базе.");

Console.WriteLine($"Ошибка при сохранении: {ex.Message}");

}

return View(model);

}

Кроме того, при создании графика происходит выборка данных:

[HttpGet]

public async Task<IActionResult> IncomeChart()

{

var user = await \_userManager.GetUserAsync(User);

var incomes = await \_context.Incomes

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

.Include(i => i.Category)

.ToListAsync();

var model = new ChartViewModel

{

Incomes = incomes

};

return View(model);

}

[HttpGet]

public async Task<IActionResult> ExpenceChart()

{

var user = await \_userManager.GetUserAsync(User);

var expenses = await \_context.Expenses

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

.Include(e => e.Category)

.ToListAsync();

var model = new ChartViewModel

{

Expenses = expenses

};

return View(model);

}

# 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. Модуль входа в систему | | | | |
| 1.1 | Запуск приложения | Отображается окно авторизации. | | + |
| 1.2 | В форме поля логин или пароль пользователя пустые | Появилась подсказка внизу незаполненного поля:    Загрузка главной страницы не произошла | | + |
| 1.3 | Введены неверные логин или пароль и  нажатие кнопки «Войти». | Загрузка главной страницы не происходит. Пароль сброшен | | + |
| 1.4 | В форме авторизации пользователь ввёл верные данные и  нажал кнопку «Войти». | Загрузится Главная страница приложения. | | + |
| 2. Добавление записи | | | | |
| 2.1 | Нажатие кнопки добавления записи на главной странице клиента или «Добавление» на меню | Отображается страница для добавления | | + |
| 2.2 | На странице для добавления заполнены необходимые поля и нажата кнопка «Сохранить» | Появилась подсказка    Затем отображается главная страница клиента | | + |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 3. Удаление записи | | |  |
| 3.1 | Нажатие на кнопку удаления на главной странице клиента | Запись удалена и обновляется страница | + |
| 4. Работа с графиком | | | |
| 4.1 | Нажатие на кнопку просмотра графика на главной странице или нажатие на кнопку «Анализ» на навигационном меню | Открывается страница для просмотра графика | + |
| 4.2 | Наведение на область на графике | Отображается подсказка с информацией | + |

Таким образом, были протестированы основные функции приложения.

5.2 Руководство пользователя

Перед началом работы пользователь приложения должен войти в систему или зарегистрироваться.

Работа пользователя в приложении начинается с авторизации (рисунок 2).

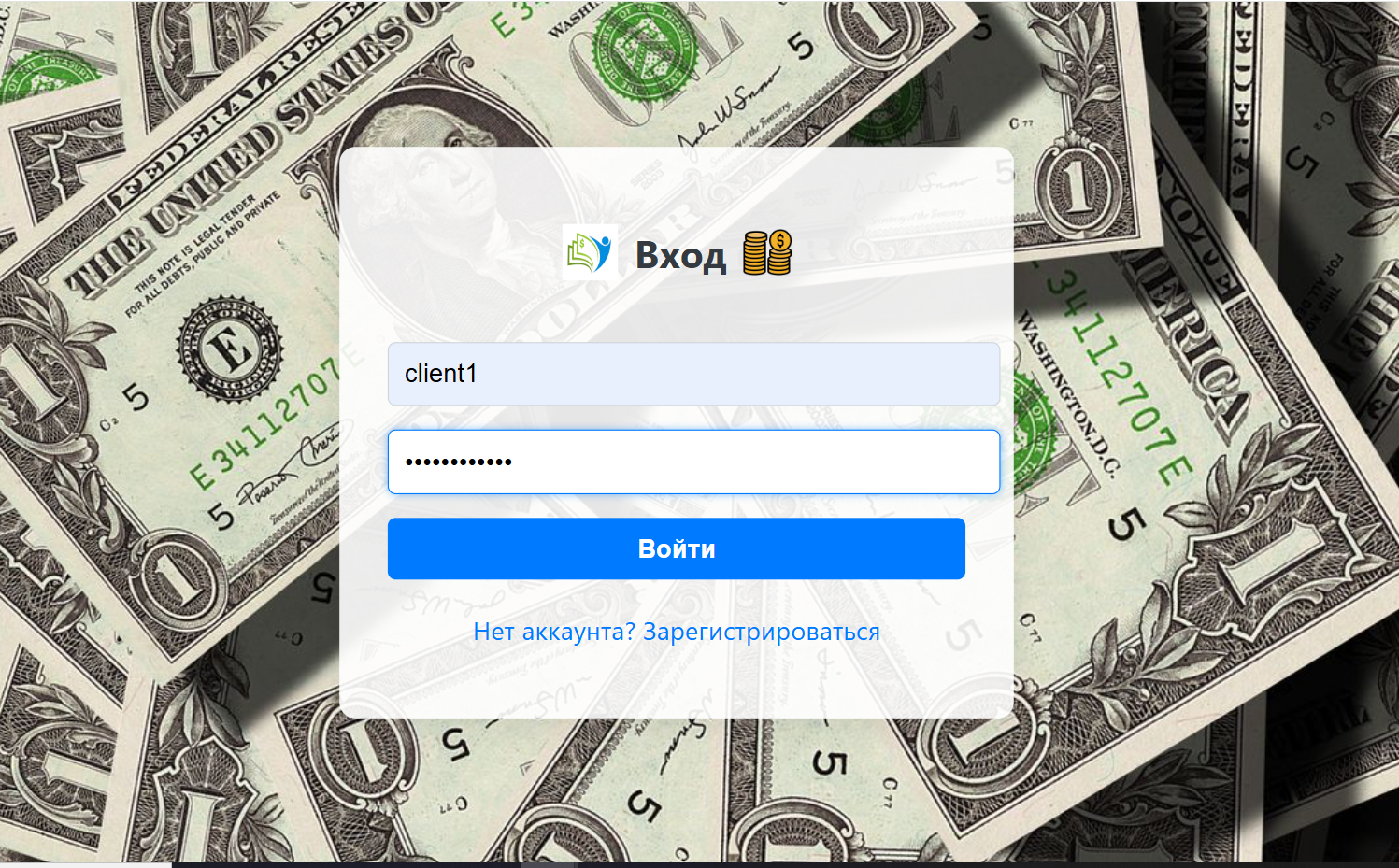


Рисунок 2 – Авторизация пользователя

После входа клиента отображается главная страница, которая показывает доходы и расходы клиента (рисунок 3).

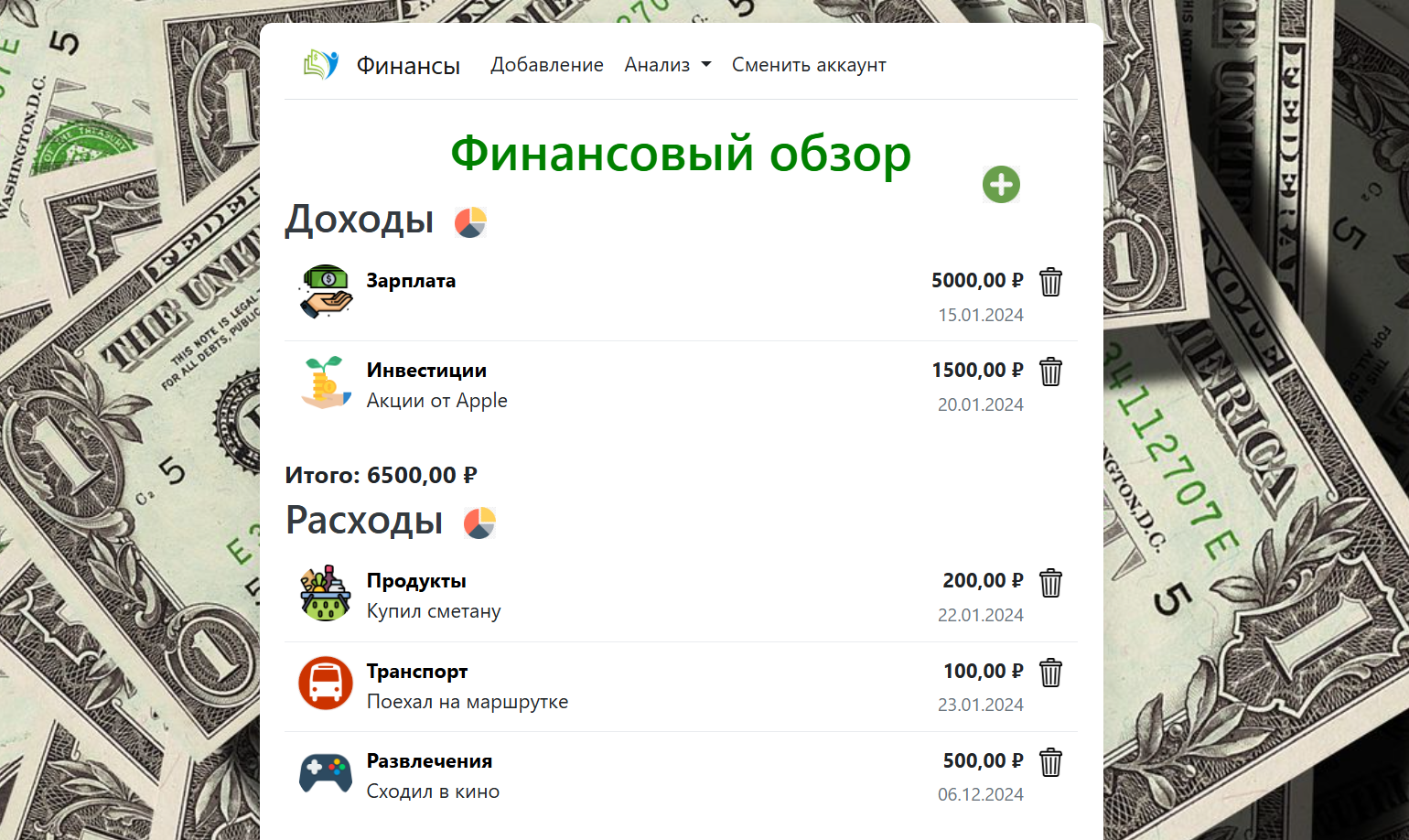


Рисунок 3 – Главная страница клиента

Для удаления какой-либо записи необходимо нажать на кнопку для удаления, которая отображается справа от записи в виде рисунка «мусорка» (рисунок 4). После нажатия соответствующая запись будет удалена.



Рисунок 4 – Кнопка для удаления

Для добавления записи необходимо щёлкнуть по кнопке для добавления, которая находится справа от заголовка «Финансовый обзор» и выглядит в виде «Плюса» (рисунок 5) или по элементу меню «Добавление». После щелчка откроется страница (рисунок 6) для добавления, которую нужно заполнить и щёлкнуть по кнопке «Сохранить». Получим уведомление (рисунок 7).



Рисунок 5 – Кнопка для добавления новой записи

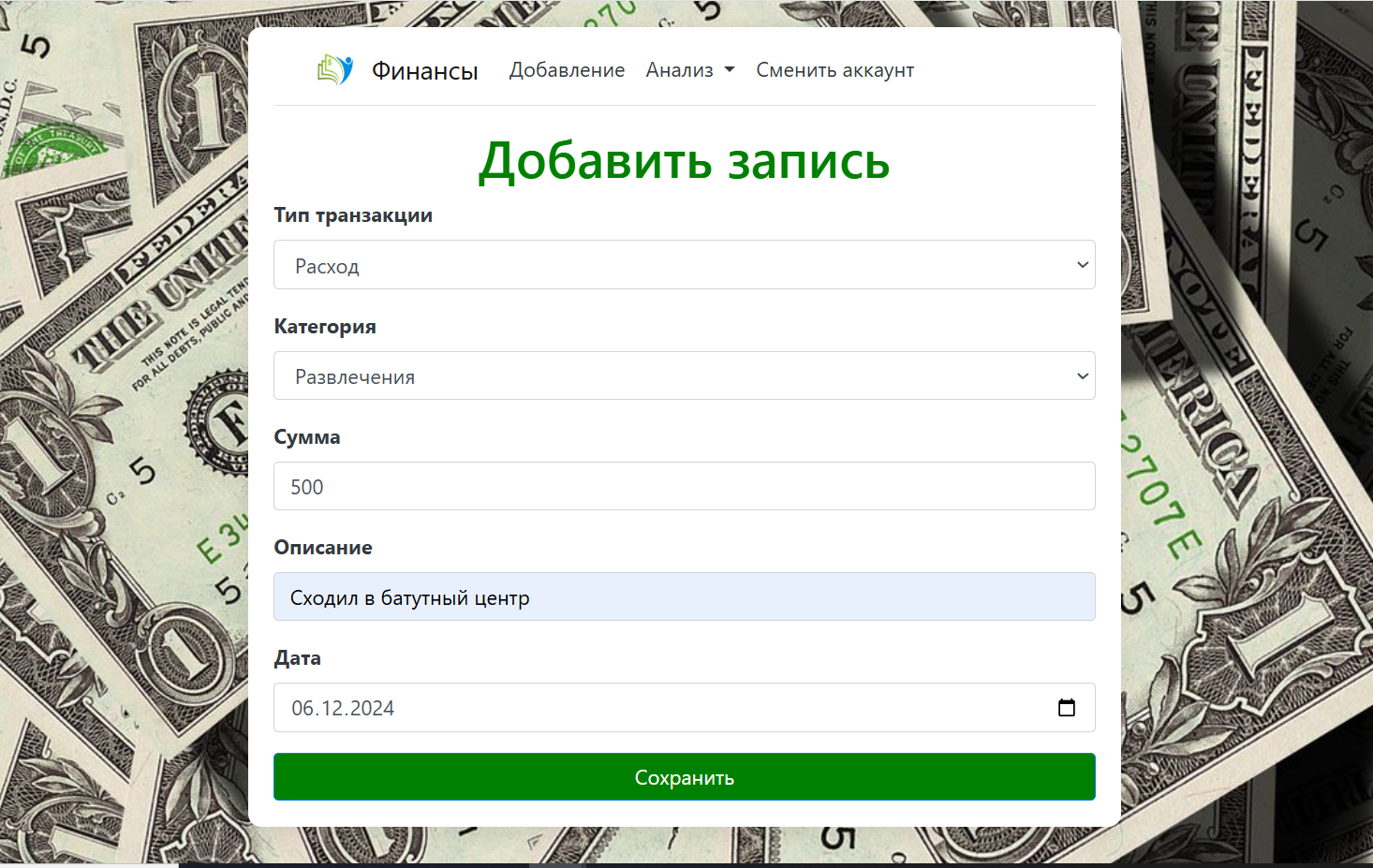


Рисунок 6– Страница для добавления записи

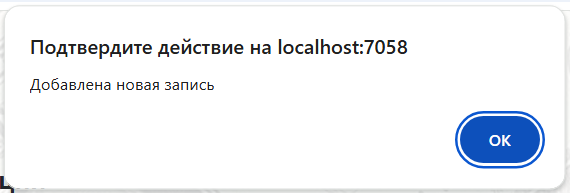


Рисунок 7 – Уведомление о добавлении

После добавления клиент попадает обратно на главную страницу (рисунок 3).

Для просмотра графика необходимо нажать на кнопку просмотра графика (рисунок 8) справа от заголовка «Доходы» или «Расходы» или выбрать меню «Анализ», далее «Доход» или «Расход». Отобразится соответствующая страница (рисунок 9). При наведении на график отобразится информация о категории (рисунок 10).



Рисунок 8 – Кнопка для просмотра графика

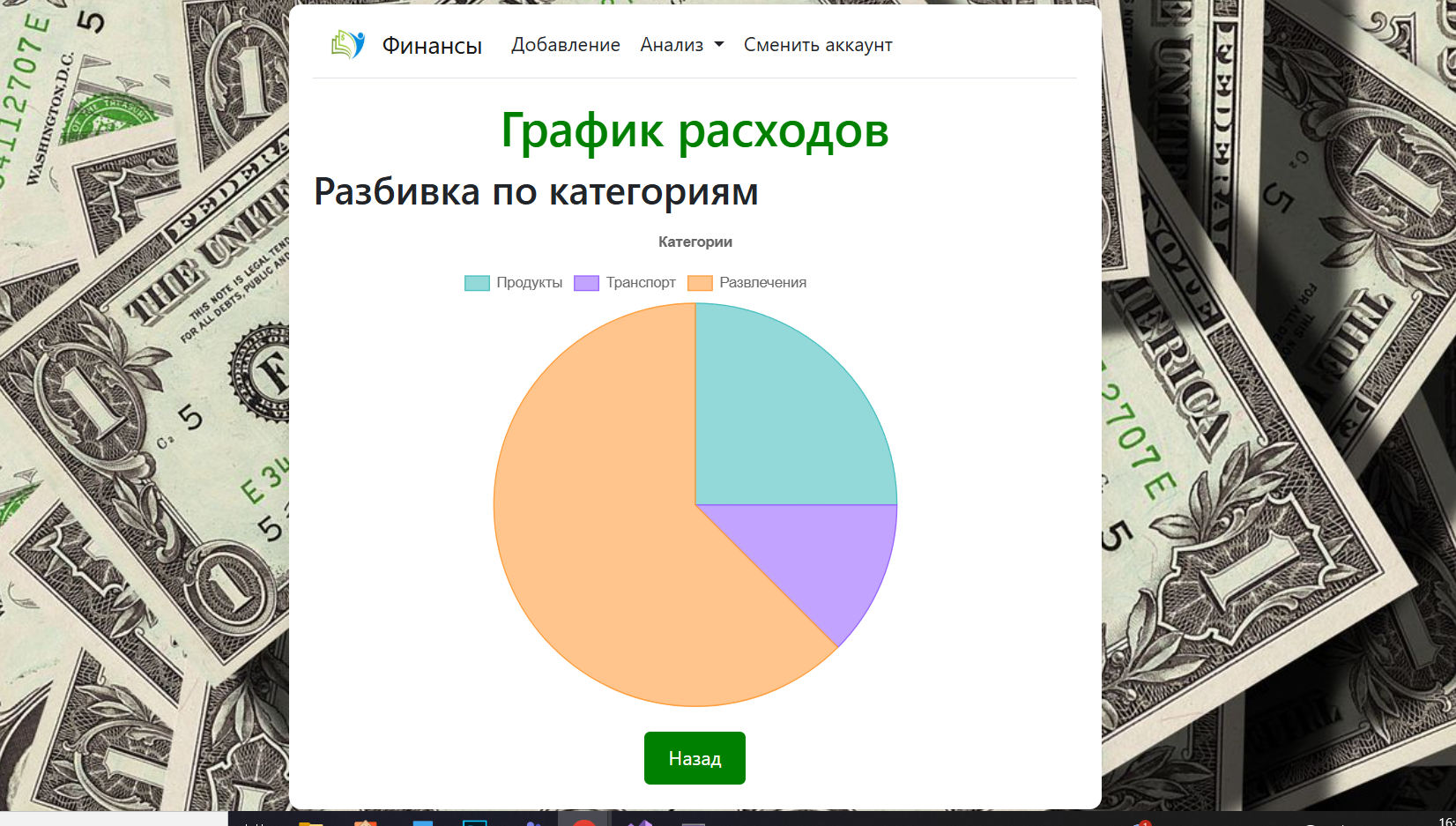


Рисунок 9 – Страница просмотра графика

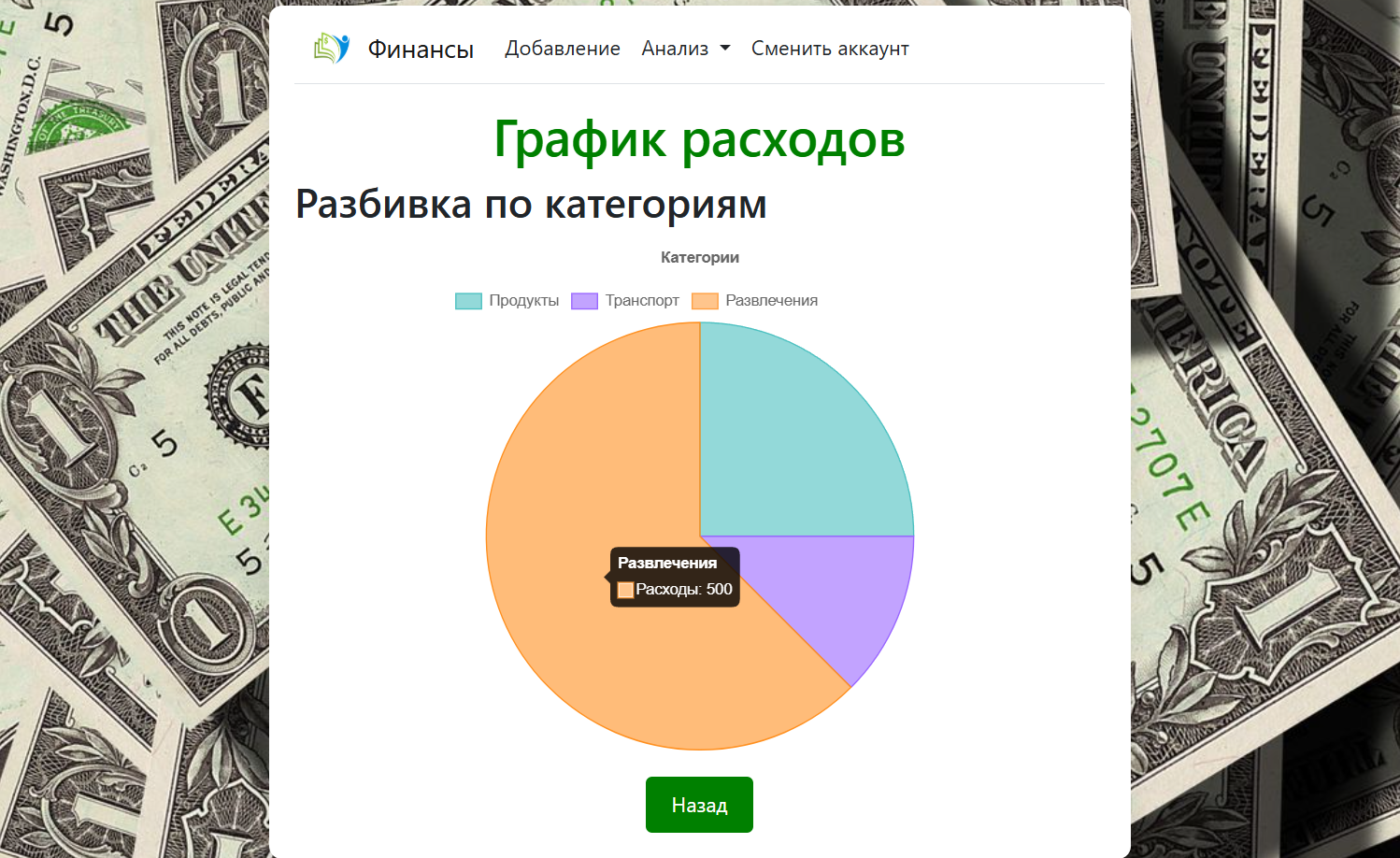


Рисунок 10 – Отображение информации о категории

# Заключение

В условиях растущего интереса к финансовой грамотности и необходимости эффективного управления личными средствами, современные интернет-приложения становятся важным инструментом для пользователей, стремящихся контролировать свои финансы. Одним из ключевых аспектов успешности таких решений является удобный и функциональный интерфейс, который помогает пользователю быстро и легко анализировать данные, принимать обоснованные решения и управлять своими финансами.

Использование архитектуры MVC (Model-View-Controller) в разработке таких приложений позволяет эффективно разделять бизнес-логику, данные и визуальное представление, что делает приложение более масштабируемым и поддерживаемым. Frontend, как неотъемлемая часть архитектуры, отвечает за взаимодействие с пользователем, обеспечивая интуитивно понятный и динамичный интерфейс, реализованный с использованием HTML, CSS и JavaScript. Эти технологии обеспечивают как структуру страницы, так и её визуальное оформление и интерактивность, что делает веб-приложение функциональным и удобным в использовании.

В результате сочетания MVC и передовых frontend-технологий, пользователи получают мощный и гибкий инструмент для управления своими финансовыми средствами. Такой подход обеспечивает не только эффективный учет, но и адаптивный интерфейс, подходящий для людей с различным уровнем технической подготовки. Таким образом, создание веб-приложения для учета и анализа финансов становится важной составляющей современных технологий, ориентированных на улучшение качества жизни пользователей.

# Список литературы

1. **Хеник, Б.** - HTML и CSS. Путь к совершенству (HTML и CSS: The Good Parts), 2017 – 284 с.

2. **Флэнаган, Д.** - "JavaScript. Подробное руководство (JavaScript. The Definitive Guide)", 2018. – 992 с.

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

4. **Кангин,** В. В. Интернет. Языки HTML и JavaScript. — М.: ТНТ, 2021. — 488 c.

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

6. **Бэнкс, А.** – React и Redux. Функциональная веб-разработка. Руководство / Бэнкс Алекс. - М.: Питер, 2018. - 458 c.

7. **Хоган, Б. П.** – HTML5 и CSS3. Веб-разработка по стандартам нового поколения / Брайан П. Хоган. - М.: Питер, 2013. — 128 c.