**Кыргызско-Российский Славянский университет**

**Кафедра информационных и вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**по дисциплине «РАЗРАБОТКА КОРПОРАТИВНЫХ ПРИЛОЖЕНИЙ»**

**Тема: *«Веб-сайт для учета личного бюджета»***

**Выполнил: студент группы ЕПИ-2-20 Мирлан уулу Алинур.**

**Руководитель:**

Оценка:

Дата:

Бишкек 2023

Оглавление

[1 Введение 3](#_Toc124406352)

[2 Разработка программы 4](#_Toc124406353)

[2.1 Технологии разработки. 4](#_Toc124406354)

[2.2 Дизайн и структура 6](#_Toc124406355)

[2.3 Модель данных и алгоритм решения задачи. 11](#_Toc124406356)

[3 Руководство программиста 13](#_Toc124406357)

[4 Руководство пользователя 14](#_Toc124406358)

[5 Выводы 15](#_Toc124406359)

# Введение

В наше современное время управление личными финансами становится все более важной задачей для людей различных профессий и финансовых возможностей. Независимо от того, являетесь ли вы предпринимателем, работником или студентом, эффективное планирование и контроль над финансами играют ключевую роль в достижении финансовых целей и обеспечении финансовой устойчивости.

В рамках этого контекста был разработан веб-сайт на базе ASP.NET Core MVC, предназначенный для помощи пользователям в управлении и отслеживании их финансовых данных. Это веб-приложение предоставляет пользователю возможность следить за текущим балансом, контролировать свои расходы и доходы, а также эффективно планировать свои бюджеты на различные категории трат.

Основной целью этого отчета является предоставление детального описания разработанного веб-сайта, его функциональности и возможностей, а также обзора ключевых характеристик и преимуществ, которые он предоставляет пользователям. В отчете также будут рассмотрены основные технологии и инструменты, использованные при разработке веб-приложения.

# Содержательная постановка задачи

**Название проекта**: Приложение учета бюджета

Цель проекта:

Разработать веб-приложение, которое позволит пользователям отслеживать свои финансы, управлять бюджетом и анализировать свои расходы и доходы.

**Задачи проекта:**

1. Регистрация и аутентификация пользователей:

Предоставить возможность пользователям создать учетную запись и войти в систему с помощью логина и пароля.

Обеспечить безопасность и защиту персональных данных пользователей.

1. Управление категориями доходов и расходов:

Позволить пользователям создавать и управлять категориями для классификации своих доходов и расходов.

Дать возможность пользователям добавлять, редактировать и удалять категории.

1. Ввод и отслеживание доходов и расходов:

Разработать форму, которая позволит пользователям вводить информацию о своих доходах и расходах, указывая категорию, тэги сумму и описание.

Сохранять введенные данные в базе данных и связывать их с соответствующими пользователями.

Предоставить возможность сортировать транзакции по дате и сумме как по убыванию, так и по возрастанию.

1. Отображение баланса и сводной информации:

Показывать текущий баланс пользователя, отражающий разницу между его доходами и расходами.

Выводить сводную информацию о доходах и расходах по выбранному периоду времени или категориям.

1. Управление бюджетом:

Позволить пользователям создавать и управлять бюджетами на определенные категории расходов.

Отслеживать выполнение бюджета и обеспечить сохранение закрытых бюджетов в архиве.

Предоставить возможность сортировать бюджеты по дате создания и категории как по убыванию, так и по возрастанию.

1. Анализ бюджетов, расходов и доходов:

Дать возможность фильтровать данные и сравнивать их для получения более детальной информации и анализа.

**Технические требования:**

* Приложение должно быть разработано с использованием ASP.NET Core MVC (или другого выбранного вами фреймворка).
* Для хранения данных о пользователях, доходах, расходах, категориях и бюджетах использовать реляционную базу данных (например, SQL Server).
* Реализовать пользовательский интерфейс, который будет интуитивно понятным и удобным в использовании.
* Обеспечить безопасность данных и защиту от несанкционированного доступа.

# Обоснование выбора СУБД и языка программирования

**СУБД SQL Server:**

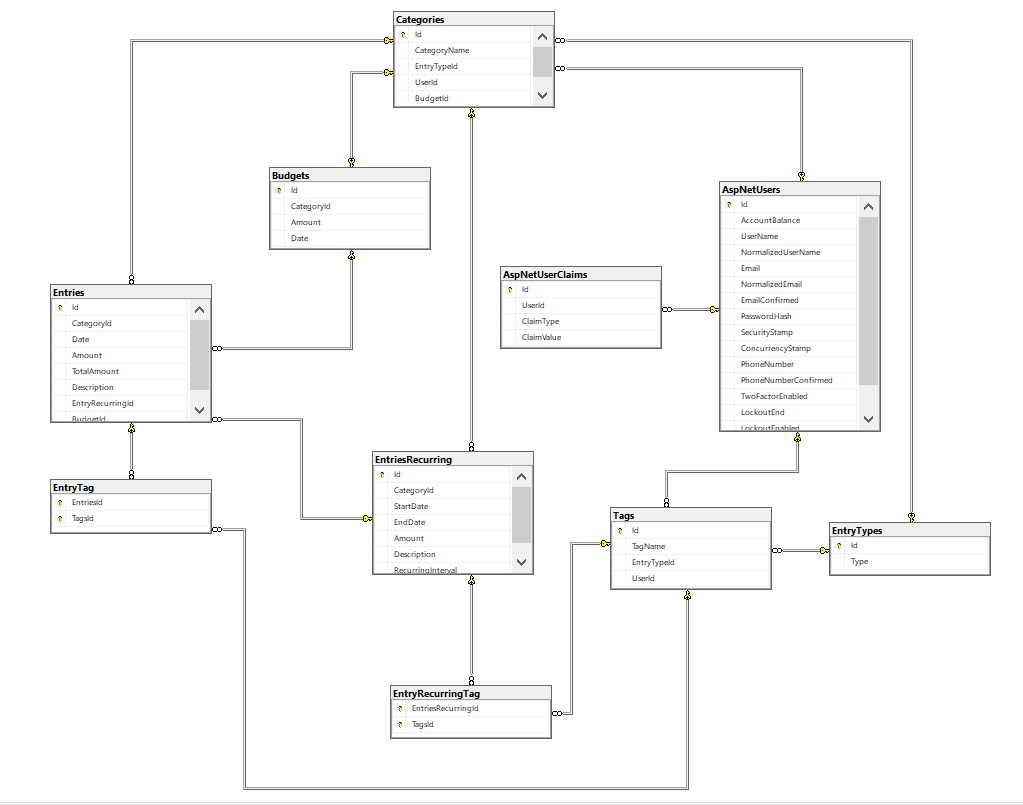
* SQL Server является одной из ведущих реляционных баз данных на рынке и предоставляет широкий набор функциональных возможностей, которые соответствуют требованиям проекта учета бюджета. Вот основные аргументы, подтверждающие выбор SQL Server:
* Надежность и производительность: SQL Server обладает высокой стабильностью и отличной производительностью, что является критическим фактором для приложения учета бюджета, где требуется эффективная обработка большого объема данных.
* Богатый набор функций: SQL Server предлагает широкий набор функциональных возможностей, таких как поддержка транзакций, индексы, процедуры и триггеры, что обеспечивает гибкость и эффективность при работе с данными приложения учета бюджета.
* Интеграция с платформой Microsoft: SQL Server плотно интегрирован с другими технологиями Microsoft, такими как .NET Framework и Azure, что облегчает разработку, развертывание и масштабирование приложения учета бюджета.
* Широкое сообщество и поддержка: SQL Server имеет большое сообщество разработчиков и обширную документацию, что обеспечивает доступ к ресурсам, поддержке и обновлениям, необходимым для успешного развития приложения учета бюджета.

**Язык программирования C#:**

* C# является основным языком программирования для разработки приложений на платформе .NET и обладает рядом преимуществ, которые подтверждают его выбор для данного проекта:
* Широкая поддержка и инструментарий: C# является основным языком разработки в экосистеме Microsoft, что обеспечивает его широкую поддержку и наличие разнообразных инструментов разработки, таких как Visual Studio, ReSharper и других.
* ООП-подход: C# является объектно-ориентированным языком программирования, что облегчает организацию и структурирование кода приложения учета бюджета, а также обеспечивает возможности наследования, полиморфизма и инкапсуляции.
* Интеграция с платформой .NET: C# тесно интегрирован с платформой .NET, что позволяет использовать множество библиотек и фреймворков, упрощает разработку и обеспечивает высокую производительность приложения.
* Большое сообщество разработчиков: C# имеет широкое сообщество разработчиков, что обеспечивает доступ к информации, помощи, обновлениям и расширениям, способствующим эффективной разработке приложения учета бюджета.

В результате, выбор СУБД SQL Server и языка программирования C# обоснован и соответствует требованиям проекта учета бюджета, обеспечивая надежность, производительность, функциональность, интеграцию и поддержку необходимых для успешной разработки и функционирования приложения.

# Описание структуры разработанный базы данных



*Рис. 1. Диаграмма отношений между сущностями (SQL Server)*

Структура разработанной базы данных для веб-приложения BudgetTracker, основанной на представленных моделях, выглядит следующим образом:

1. Таблица "Budget" представляет бюджеты и содержит следующие поля:

* Id: идентификатор бюджета.
* CategoryId: идентификатор категории, к которой принадлежит бюджет.
* Amount: сумма бюджета.
* Date: дата создания бюджета.
* Entries: список записей, связанных с данным бюджетом.
* Category: связанная категория.

1. Таблица "Category" представляет категории расходов и доходов и содержит следующие поля:

* Id: идентификатор категории.
* CategoryName: название категории (обязательное поле).
* EntryTypeId: идентификатор типа записи (доход или расход), связанного с категорией.
* EntryType: связанный тип записи.
* UserId: идентификатор пользователя, которому принадлежит категория.
* User: связанный пользователь.
* Entries: список записей, связанных с данной категорией.
* BudgetId: идентификатор связанного бюджета (необязательное поле).
* Budget: связанный бюджет.
* TotalExpenses: общая сумма расходов в данной категории (вычисляемое поле, не отображается в таблице).

1. Таблица "Entry" представляет записи доходов и расходов и содержит следующие поля:

* Id: идентификатор записи.
* CategoryId: идентификатор категории, к которой относится запись.
* Date: дата создания записи.
* Amount: сумма записи.
* TotalAmount: общая сумма записи (не отображается в таблице).
* Description: описание записи.
* Tags: список тегов, связанных с записью.
* Category: связанная категория.
* EntryRecurringId: идентификатор связанной повторяющейся записи (необязательное поле).
* EntryRecurring: связанная повторяющаяся запись.
* BudgetId: идентификатор связанного бюджета (необязательное поле).
* Budget: связанный бюджет.
* StringTags: строковое представление списка тегов (вычисляемое поле, не отображается в таблице).

1. Таблица "EntryRecurring" представляет повторяющиеся записи и содержит следующие поля:

* Id: идентификатор повторяющейся записи.
* CategoryId: идентификатор категории, к которой относится повторяющаяся запись.
* StartDate: дата начала повторения записи.
* EndDate: дата окончания повторения записи.
* Amount: сумма повторяющейся записи.
* Description: описание повторяющейся записи.
* Tags: список тегов, связанных с повторяющейся записью.
* Category: связанная категория.
* RecurringInterval: интервал повторения записи.
* Entries: список записей, связанных с повторяющейся записью.
* StringTags: строковое представление списка тегов (вычисляемое поле, не отображается в таблице).

1. Таблица "EntryType" представляет типы записей и содержит следующие поля:

* Id: идентификатор типа записи.
* Type: наименование типа записи (Income - доход, Expense - расход).

1. Таблица "Tag" представляет теги и содержит следующие поля:

* Id: идентификатор тега.
* TagName: название тега (обязательное поле).
* EntryTypeId: идентификатор типа записи, связанного с тегом.
* EntryType: связанный тип записи.
* Entries: список записей, связанных с тегом.
* EntriesRecurring: список повторяющихся записей, связанных с тегом.
* UserId: идентификатор пользователя, которому принадлежит тег.
* User: связанный пользователь.

1. Таблица "User" представляет пользователей и содержит следующие поля:

* Id: идентификатор пользователя (унаследовано от IdentityUser).
* AccountBalance: баланс аккаунта пользователя (сумма денежных средств).
* Categories: список категорий, принадлежащих пользователю.
* Tags: список тегов, принадлежащих пользователю.

1. Таблица "EntryTag" представляет отношение между записями и тегами (отношение many-to-many) и содержит следующие поля:

* EntryId: идентификатор записи (внешний ключ к таблице "Entry").
* TagId: идентификатор тега (внешний ключ к таблице "Tag").

1. Таблица "EntryRecurringTag" представляет отношение между повторяющимися записями и тегами (отношение many-to-many) и содержит следующие поля:

* EntryRecurringId: идентификатор повторяющейся записи (внешний ключ к таблице "EntryRecurring").
* TagId: идентификатор тега (внешний ключ к таблице "Tag").

1. Таблица "AspNetCoreUserClaims" является встроенной таблицей в Identity Framework и представляет утверждения (claims) пользователей и содержит следующие поля:

* Id: идентификатор утверждения.
* UserId: идентификатор пользователя (внешний ключ к таблице "User").
* ClaimType: тип утверждения.
* ClaimValue: значение утверждения.

# Описание программы (руководство программиста)

Программа, разработанная с использованием ASP.NET MVC и ASP.NET Core Identity, представляет собой веб-приложение для учета бюджета, которое позволяет пользователям отслеживать свой текущий баланс, расходы и доходы, а также создавать и управлять бюджетами на траты.

В разработке приложения использовались следующие ключевые технологии и библиотеки:

1. ASP.NET MVC: Фреймворк ASP.NET MVC был выбран для разработки веб-интерфейса приложения. Он предоставляет удобную модель разработки, основанную на шаблоне MVC (Model-View-Controller), что позволяет разделить логику приложения на компоненты, упрощает поддержку и расширение кода.
2. ASP.NET Core Identity: Библиотека ASP.NET Core Identity использовалась для обеспечения аутентификации и авторизации пользователей. Она предоставляет готовые инструменты для работы с пользователями, ролями, аутентификацией через различные провайдеры и управлением доступом.
3. Библиотека Hangfire C#: Библиотека Hangfire C# была использована для выполнения повторяющихся задач, таких как расходы и доходы, в фоновом режиме. Она позволяет запланировать и автоматически выполнять задачи в заданное время, что упрощает процесс автоматизации и повторения операций.
4. Проверка данных с помощью ModelState: Для обеспечения верификации данных, введенных пользователем в формы, был использован механизм ModelState, предоставляемый ASP.NET Core. Он позволяет проверить корректность и целостность данных перед их обработкой и сохранением.
5. AutoMapper: Библиотека AutoMapper была задействована для упрощения и повторного использования преобразования данных между объектами модели и объектами представления. Она автоматически выполняет сопоставление полей и свойств между объектами различных типов, что сокращает объем кода и улучшает его читаемость.
6. Паттерн Post/Redirect/Get: Для обеспечения безопасного и правильного обновления страниц после отправки данных формы был использован паттерн Post/Redirect/Get. Он гарантирует, что после успешной обработки данных формы пользователь будет перенаправлен на другую страницу, предотвращая повторное выполнение действий при обновлении страницы.
7. Entity Framework Core: В проекте был использован Entity Framework Core в качестве объектно-реляционного отображения (ORM). Он предоставляет удобные инструменты для работы с базой данных, позволяет выполнять операции CRUD (создание, чтение, обновление, удаление) с объектами модели, а также обеспечивает управление миграциями и взаимодействие с базой данных.
8. Сессии: Для хранения данных между запросами была использована технология сессий, предоставляемая ASP.NET Core. Сессии позволяют сохранять состояние информации на сервере между запросами клиента, что позволяет удобно передавать данные и сохранять контекст выполнения.
9. ASP.NET Core Areas: Для логического разделения функциональности приложения, в частности для отделения доходов от расходов, была использована функциональность ASP.NET Core Areas. Она позволяет организовать код приложения в раздельные области (Areas), каждая из которых может содержать собственные контроллеры, представления и другие компоненты.

В результате использования указанных технологий и библиотек было достигнуто эффективное разделение обязанностей, обеспечена аутентификация и авторизация пользователей, выполнение повторяющихся задач, верификация данных в формах, преобразование данных, безопасное обновление страниц, управление базой данных и хранение состояния между запросами. Это позволяет программисту разрабатывать и поддерживать приложение учета бюджета с высокой эффективностью и надежностью.

# Руководство пользователя

1. Регистрация:

1. Откройте приложение и перейдите на страницу регистрации.
2. Введите свою электронную почту, пароль и желаемую начальную сумму.
3. Нажмите кнопку "Register", чтобы создать свою учетную запись.

2. Главная страница:

1. После успешной регистрации вы будете перенаправлены на главную страницу.
2. На главной странице отображается история ваших трат и доходов.
3. Вы можете отсортировать транзакции по дате и сумме транзакции в порядке убывания или возрастания.
4. Такая же сортировка доступна и на отдельных страницах для доходов и расходов.
5. Используйте форму на правой стороне истории транзакций для поиска и фильтрации транзакций по следующим параметрам: начальная дата, конечная дата, теги и описание.

3. Доходы и расходы:

* 1. В навигационной панели сверху страницы есть ссылки на доходы “Income” и расходы “Expense”, с их соответствующими категориями “Categories”, повторяющимися доходами “Income” и расходами “Expense”.
  2. На данных страницах вы найдете форму для добавления новых транзакций.
  3. Заполните форму с необходимыми данными, такими как сумма, категория, теги и описание, а затем нажмите кнопку “Add”.

4. Бюджеты:

1. В навигационной панели сверху страницы есть ссылка на бюджеты “Budgets”.
2. На странице бюджетов вы можете добавлять, удалять, изменять и закрывать бюджеты.
3. Используйте форму на странице для ввода необходимых данных бюджета, таких категория и сумма бюджета.
4. Вы также можете использовать фильтры на странице для поиска и отображения бюджетов по начальной дате, конечной дате и категории бюджета.

5. Категории доходов и расходов:

1. В навигационной панели сверху страницы есть ссылка на категории доходов и расходов.
2. Щелкните на соответствующую ссылку, чтобы перейти на страницу управления категориями доходов или расходов.
3. На странице категорий вы можете добавлять и удалять категории.
4. Введите название категории в форму на странице и нажмите кнопку "Add", чтобы добавить новую категорию.

Тестирование

В процессе разработки программы, я провел ряд ручных тестов, чтобы убедиться в правильной функциональности и соответствии требованиям. Ниже представлены основные тестовые сценарии, которые были протестированы:

1. Создание бюджета:

* Проверка возможности создания нового бюджета с заданными параметрами.
* Убедиться, что созданный бюджет корректно отображается в списке бюджетов.

2. Создание категории:

* Проверка возможности создания новой категории с указанными данными.
* Убедиться, что созданная категория отображается в списке категорий.

3. Добавление записи:

* Проверка возможности добавления новой записи с заданными параметрами.
* Убедиться, что добавленная запись отображается в списке записей и связана с соответствующей категорией и бюджетом.

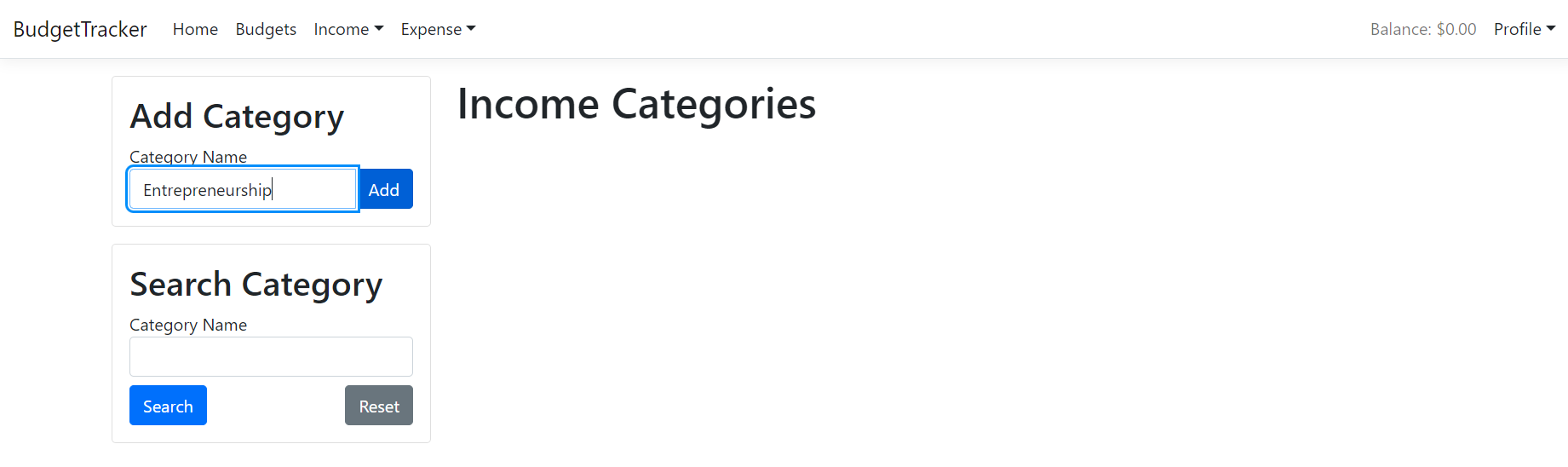
4. Редактирование записи:

* Изменение параметров существующей записи (сумма, описание, теги) и проверка сохранения изменений.
* Проверка связи измененной записи с соответствующей категорией и бюджетом.

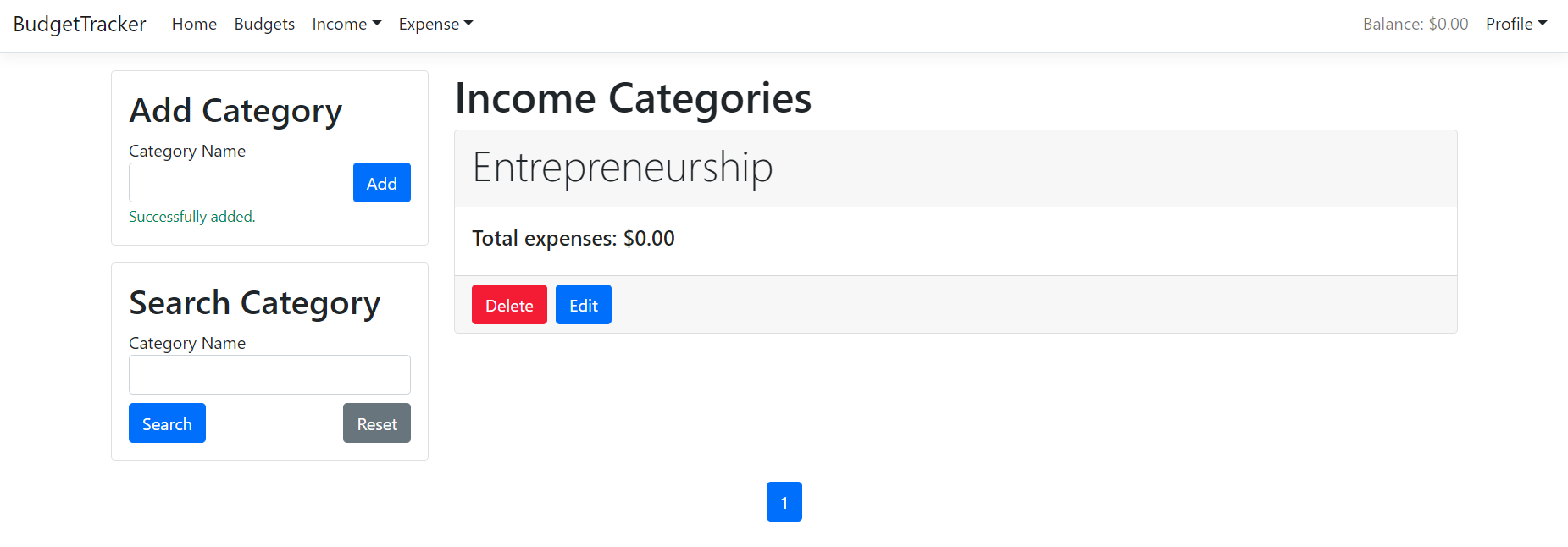
5. Удаление записи:

* Проверка возможности удаления существующей записи.
* Убедиться, что удаленная запись больше не отображается в списке записей и не связана с категорией и бюджетом.

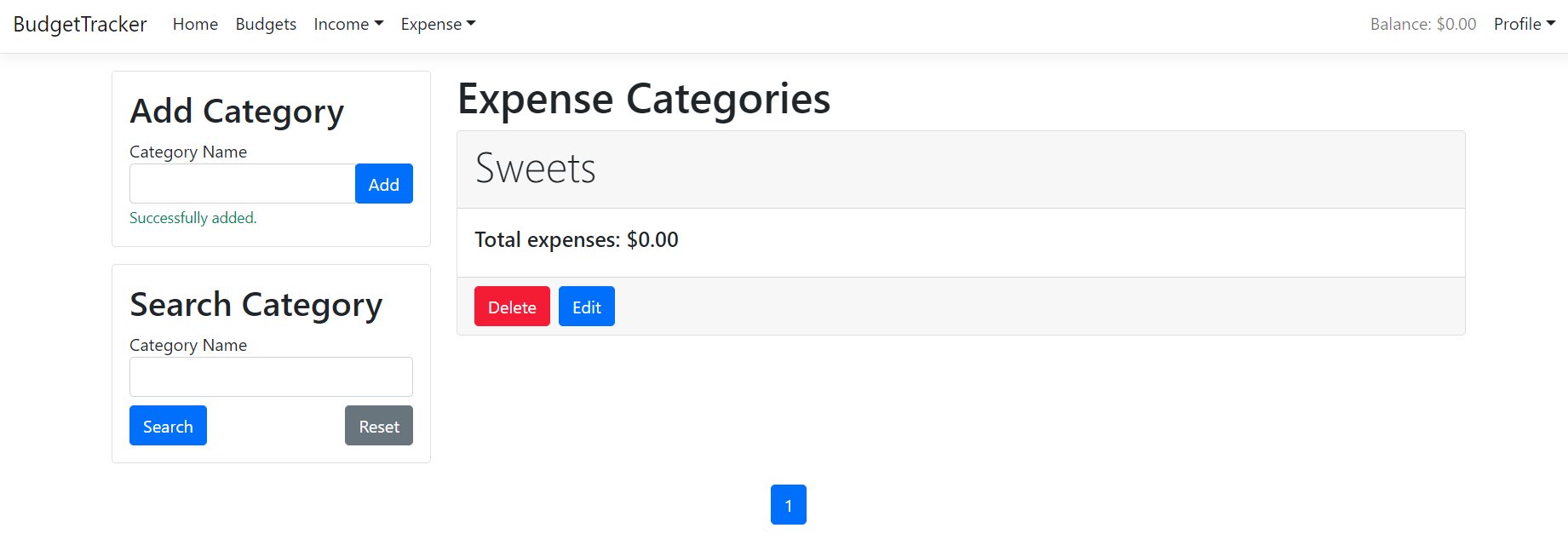
Выполненные тесты были документированы с помощью скриншотов, чтобы обеспечить визуальное подтверждение каждого шага тестирования. Вот некоторые примеры скриншотов, иллюстрирующих успешное прохождение тестов:



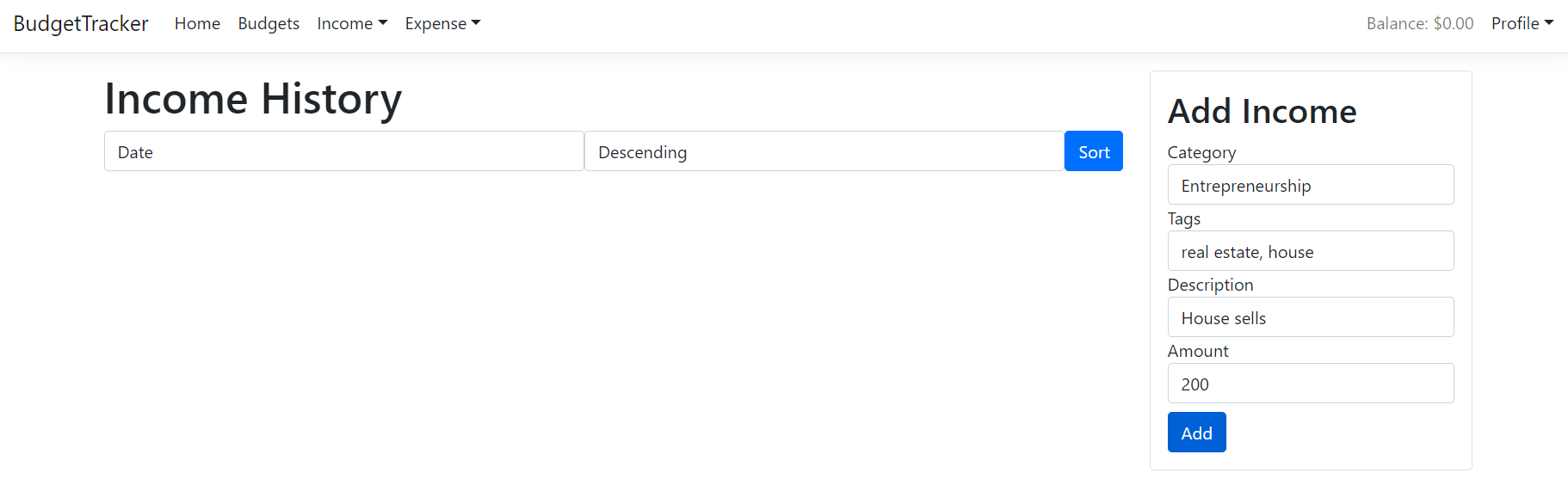
*Рис. 2. Добавление категории доходов*



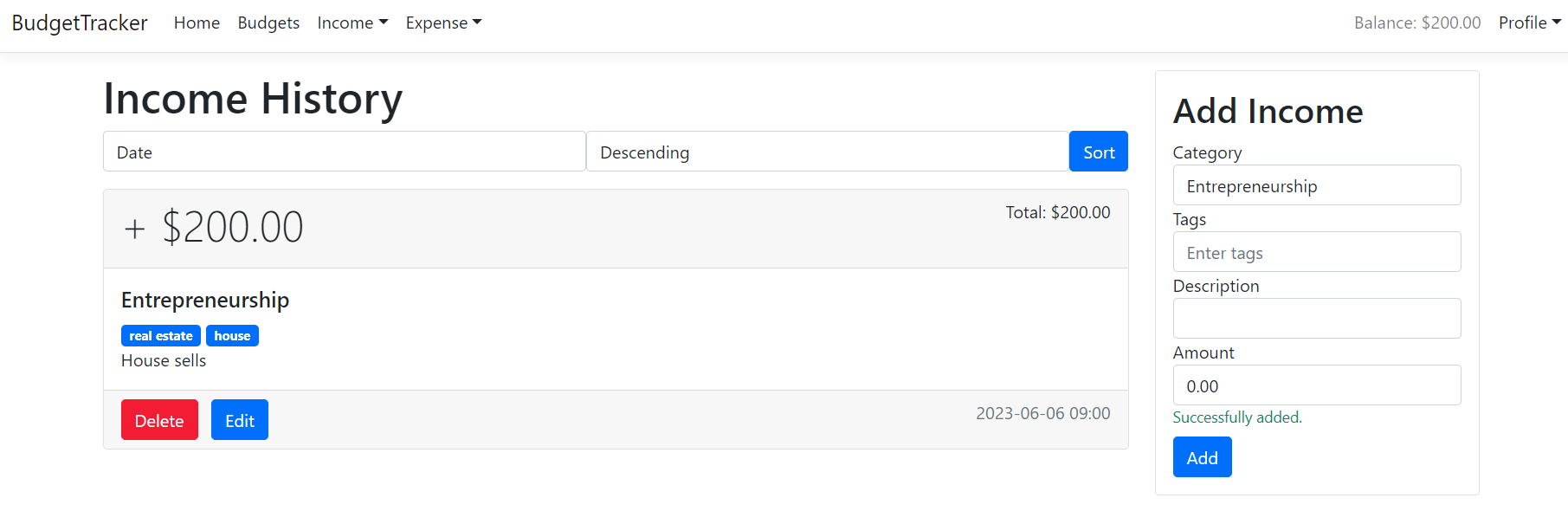
*Рис. 3. Успешное добавление категории доходов\*



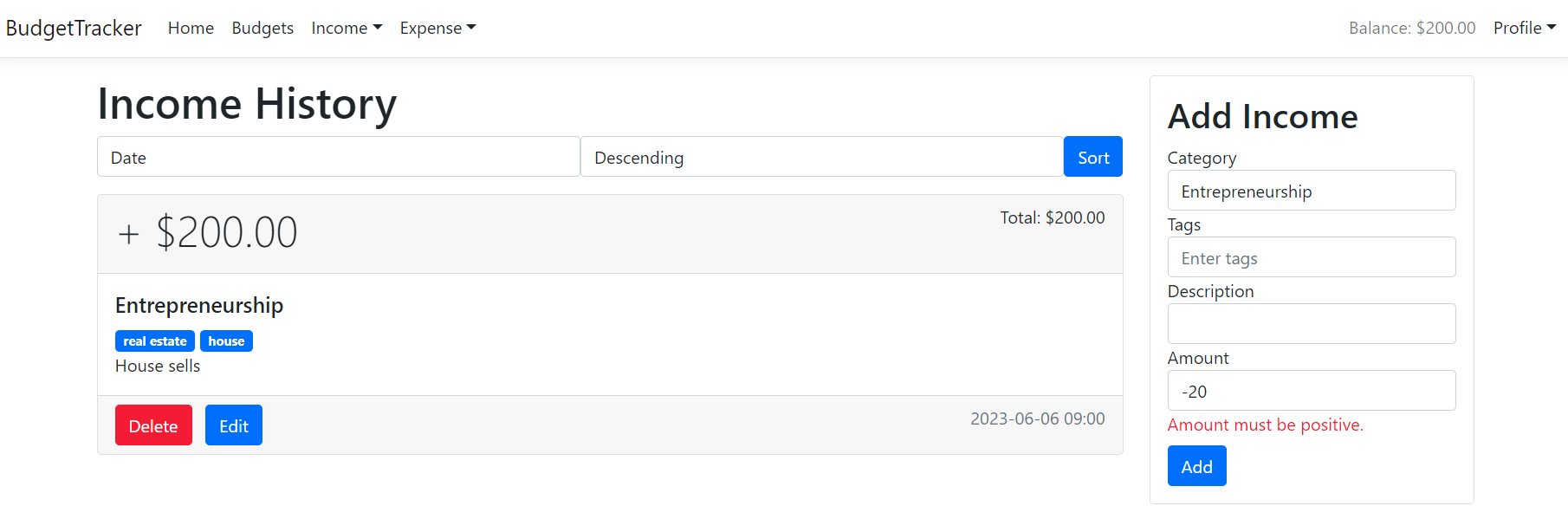
*Рис. 4. Успешное добавление категории трат*



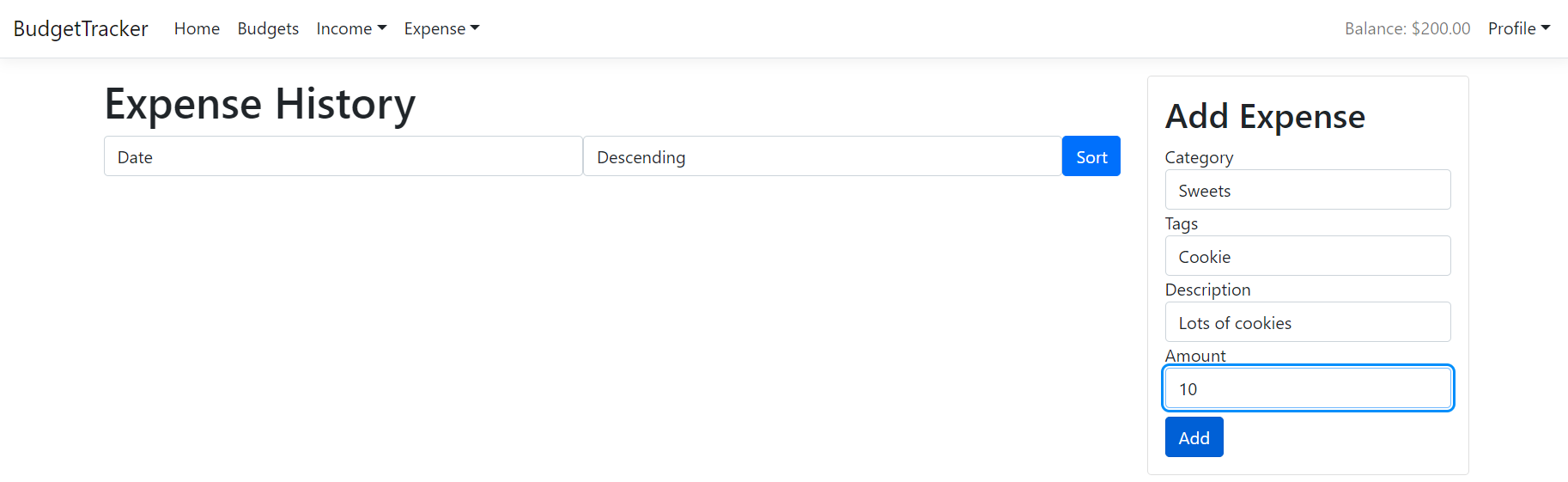
*Рис. 5. Процесс добавления дохода*



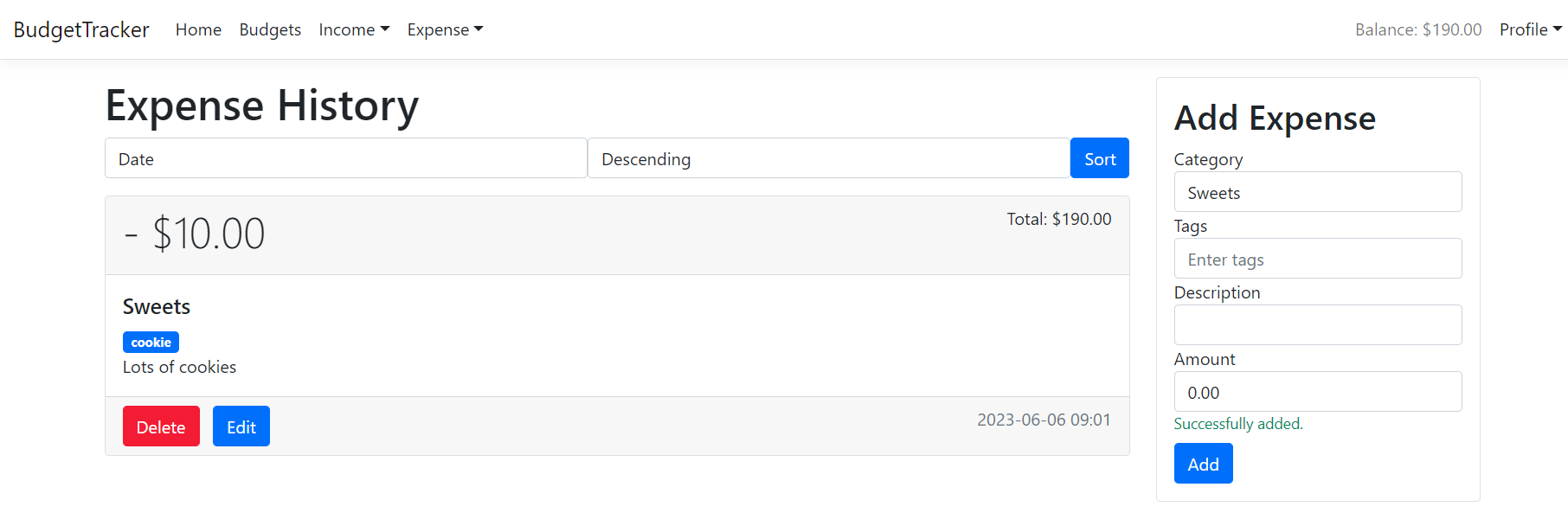
*Рис. 6. Успешное добавление дохода*



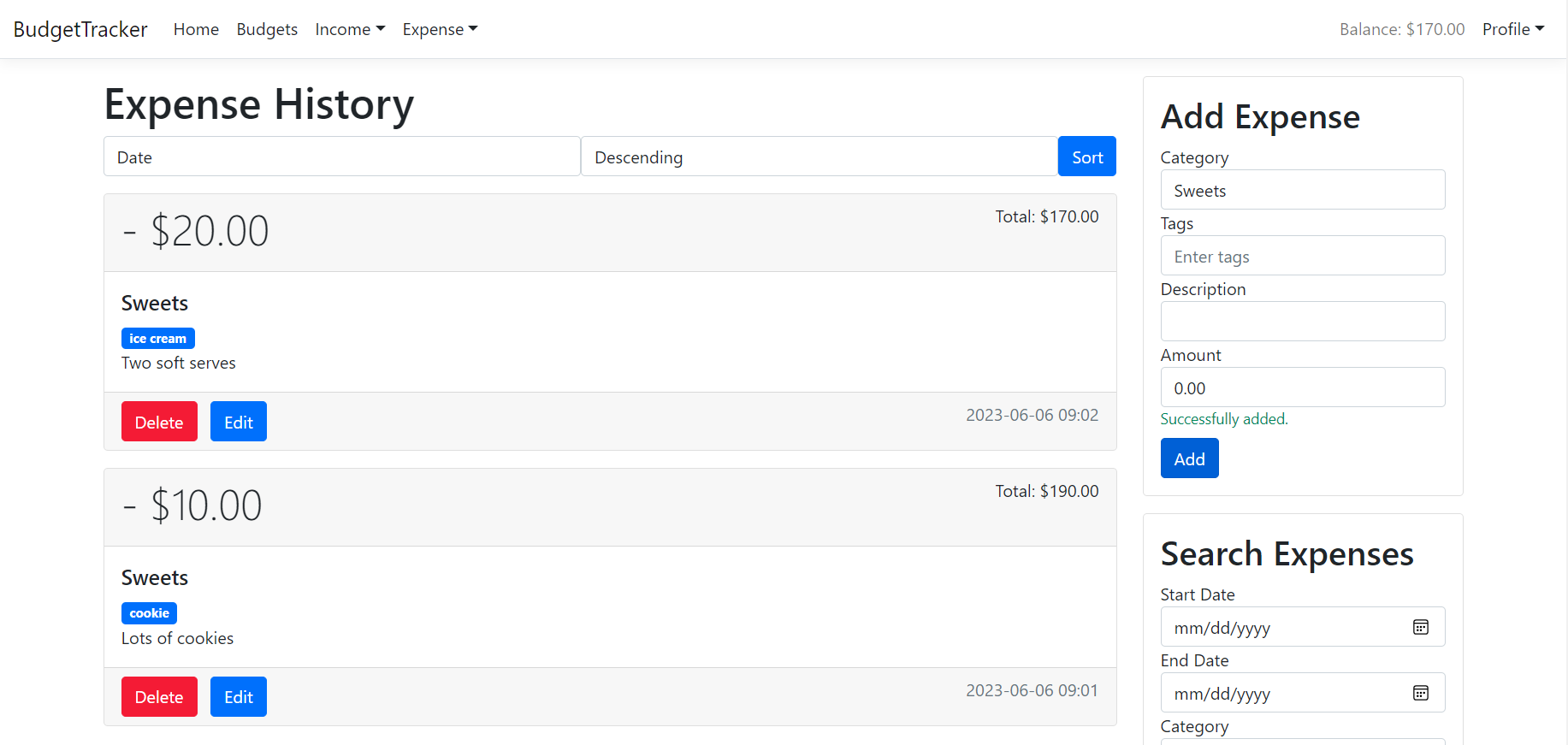
*Рис. 7. Проверка суммы дохода на положительность*



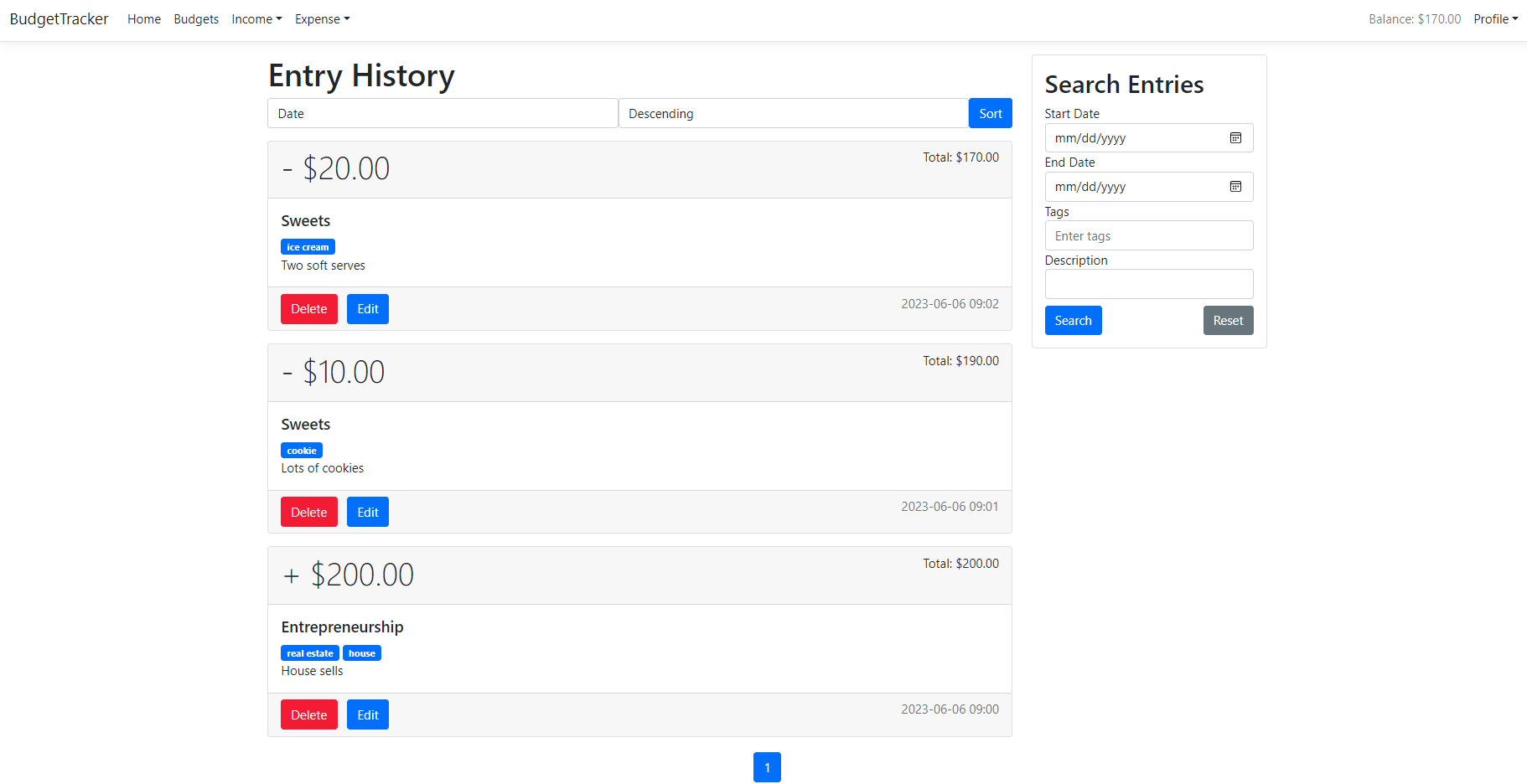
*Рис. 8. Процесс добавления затраты*



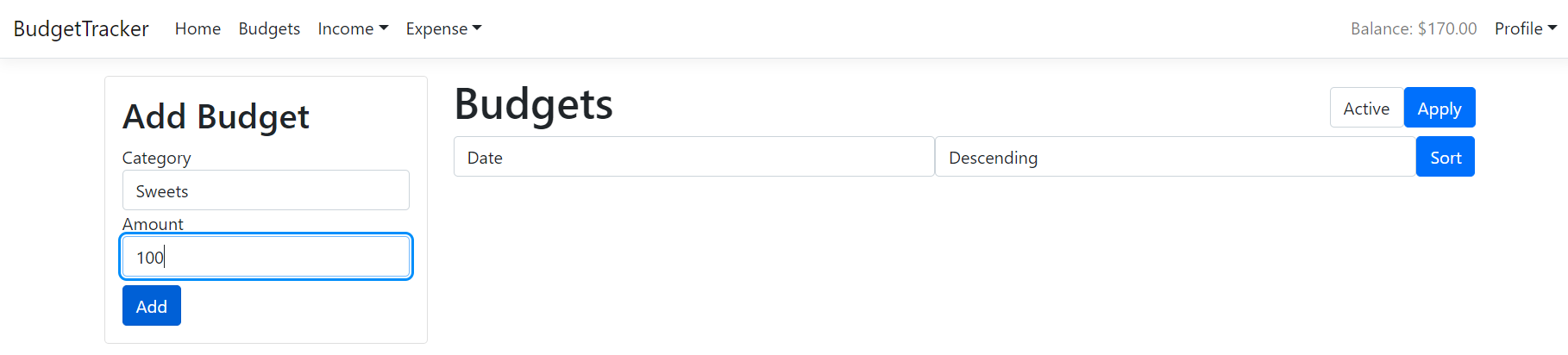
*Рис. 9. Успешное добавление затраты*



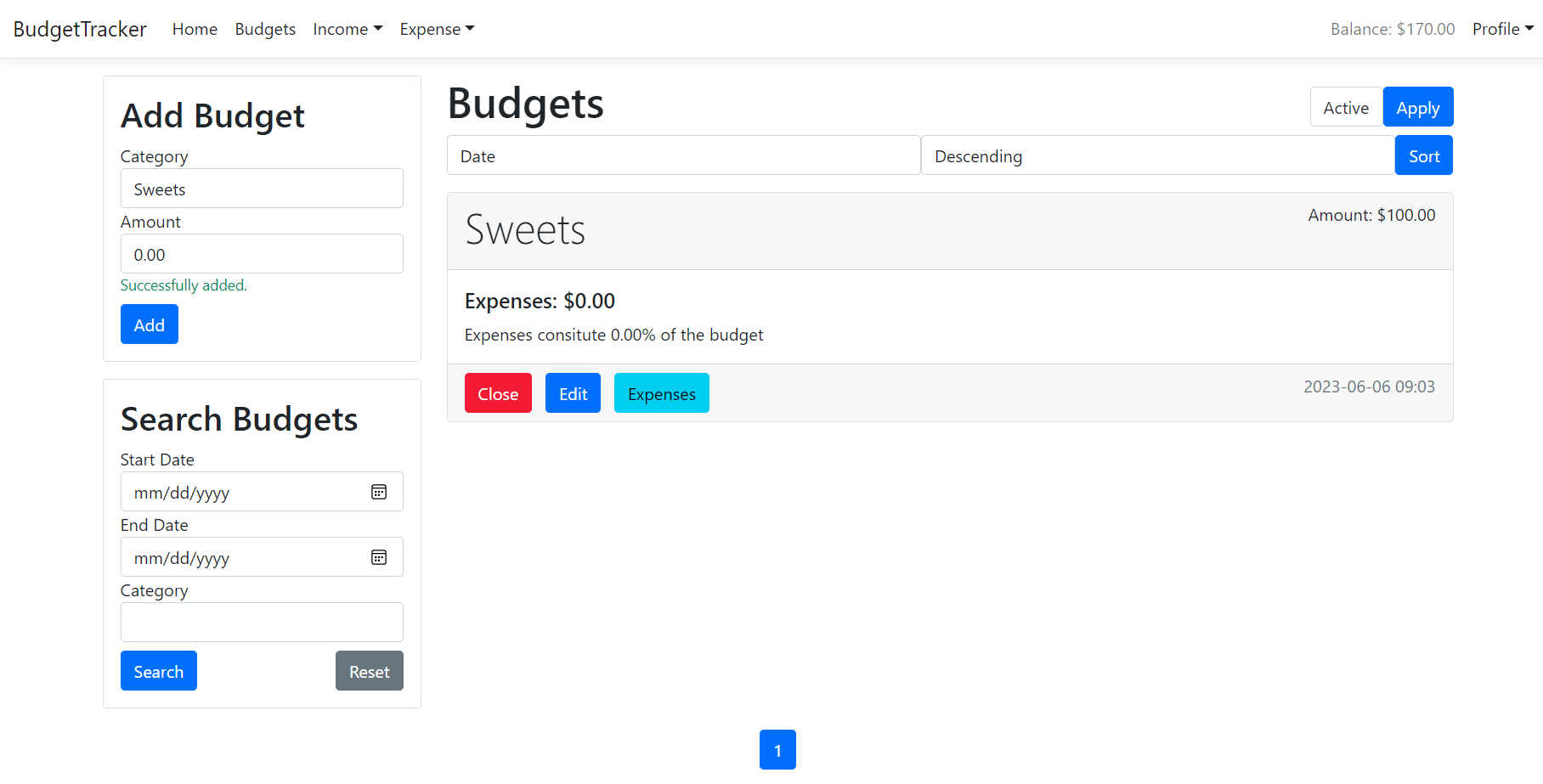
*Рис. 10. Успешное добавление еще одной затраты*



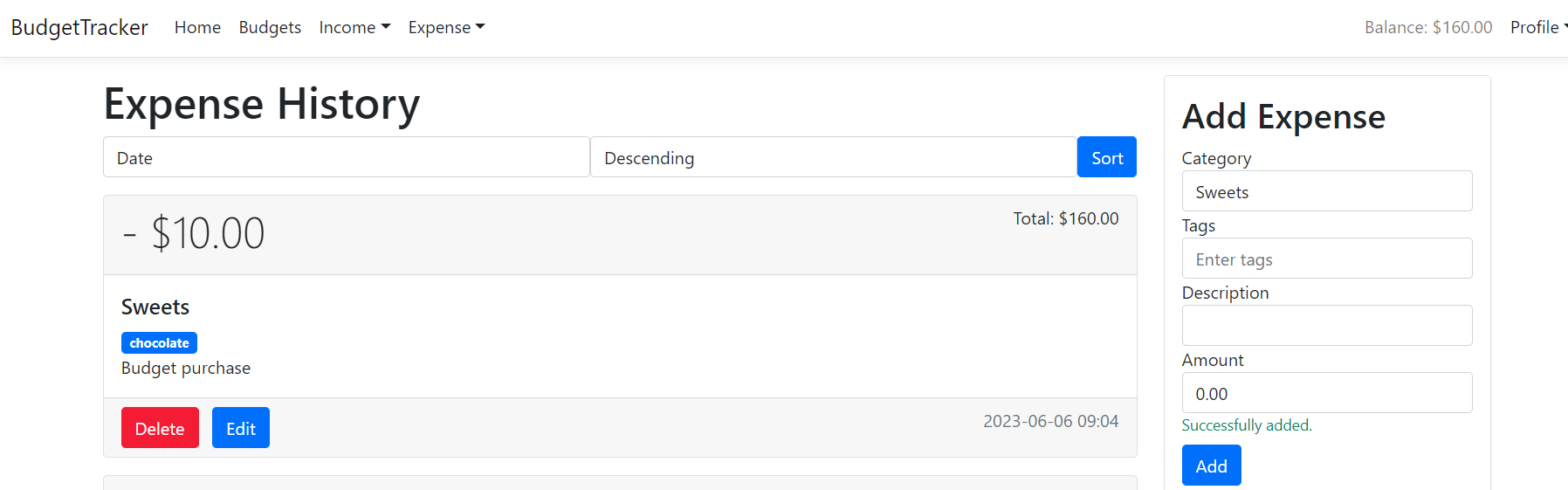
*Рис. 11. Отображение истории транзакций*



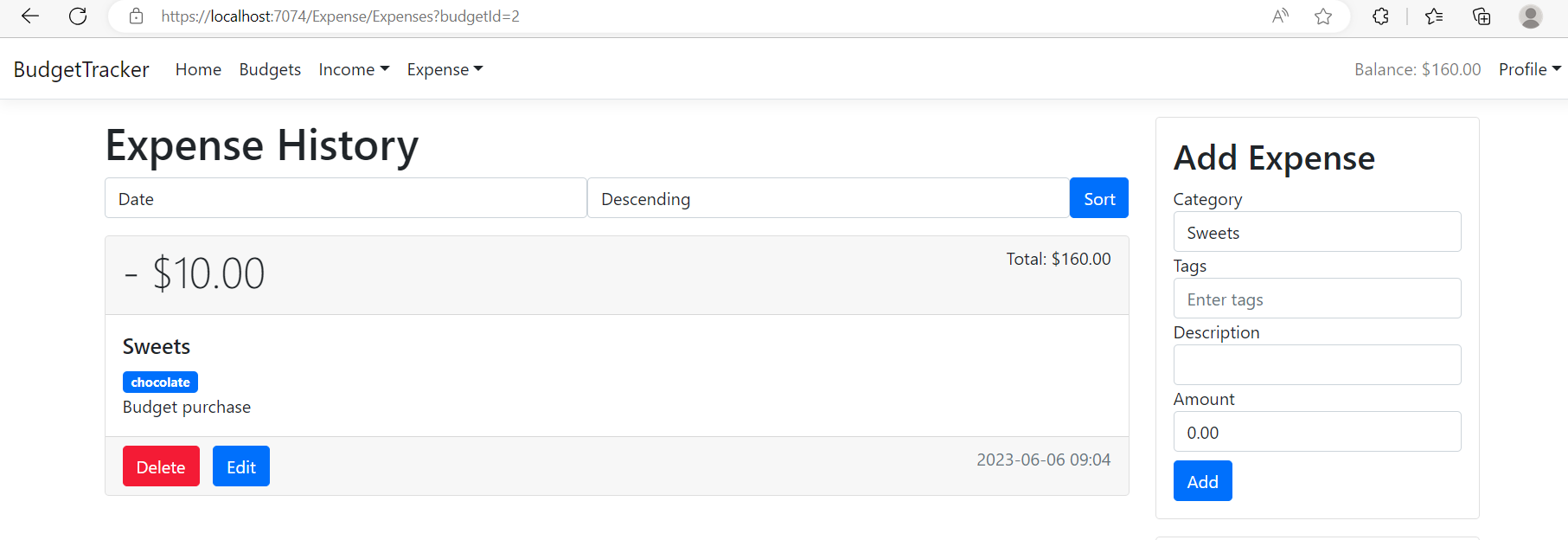
*Рис. 12. Процесс создания бюджета на конкретную категорию затрат*



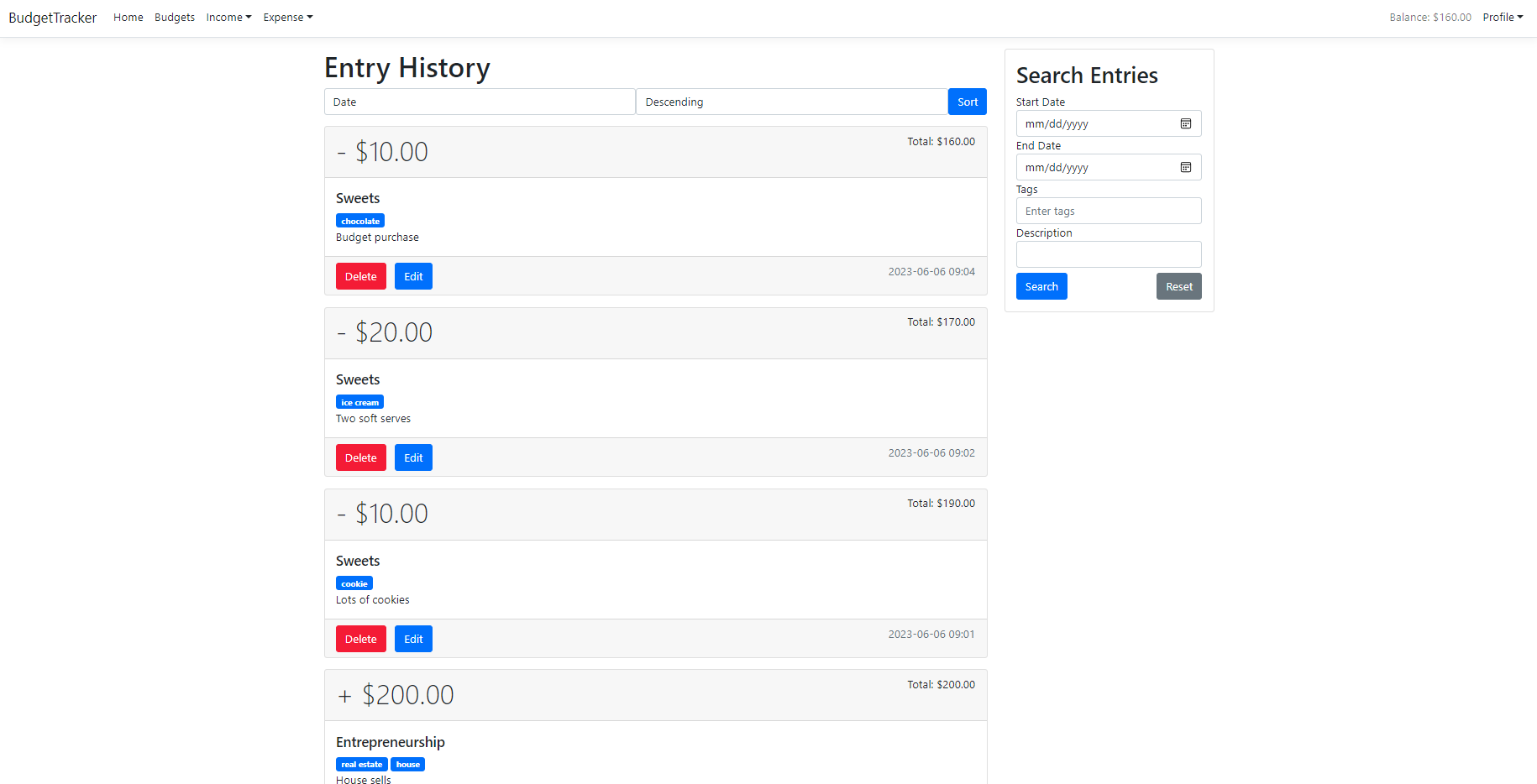
*Рис. 13. Успешное создание бюджета*



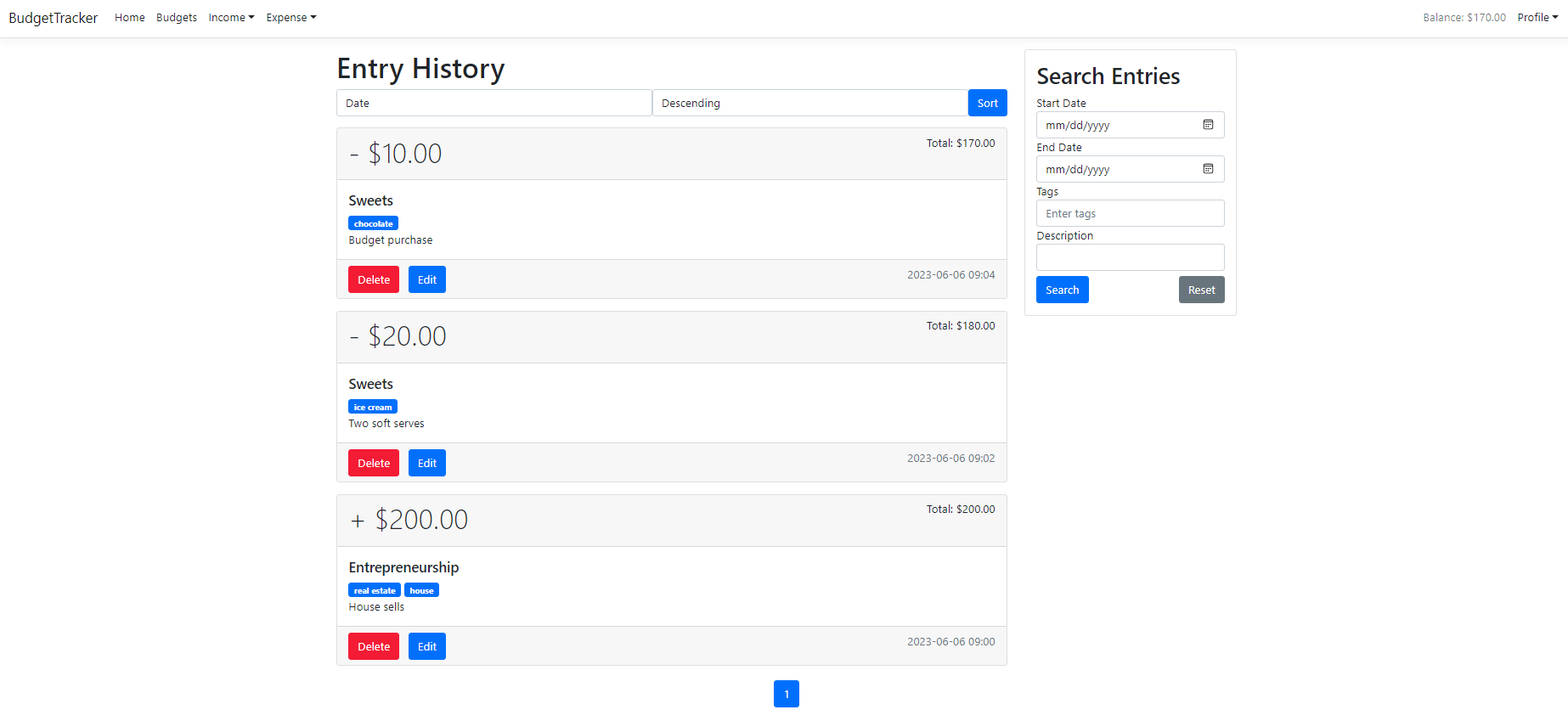
*Рис. 14. Добавление затраты на категорию, имеющую выше-добавленный бюджет*



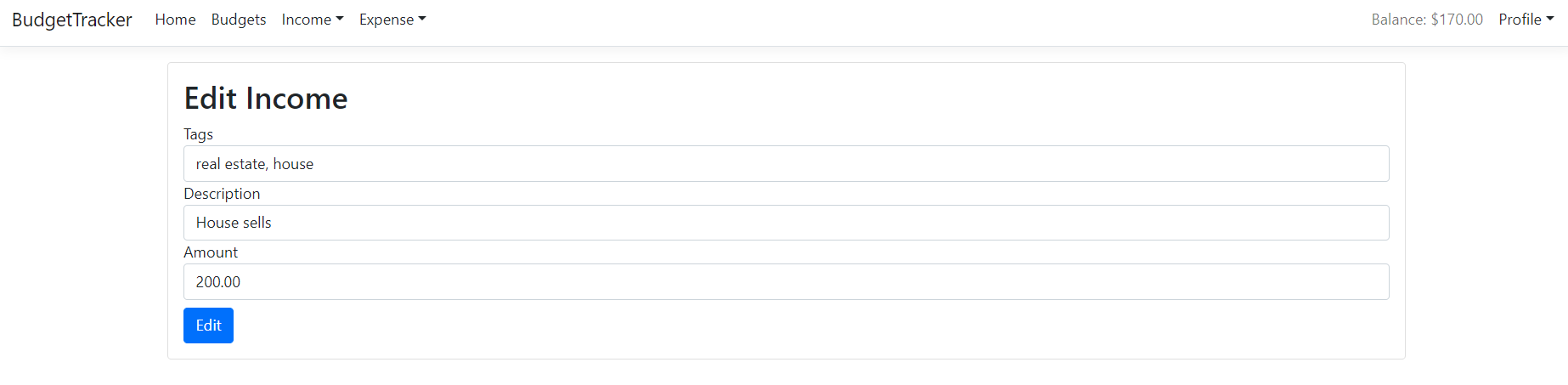
*Рис. 15. Отображение затрат бюджета (нужно перейти по кнопке бюджета - Expenses)*



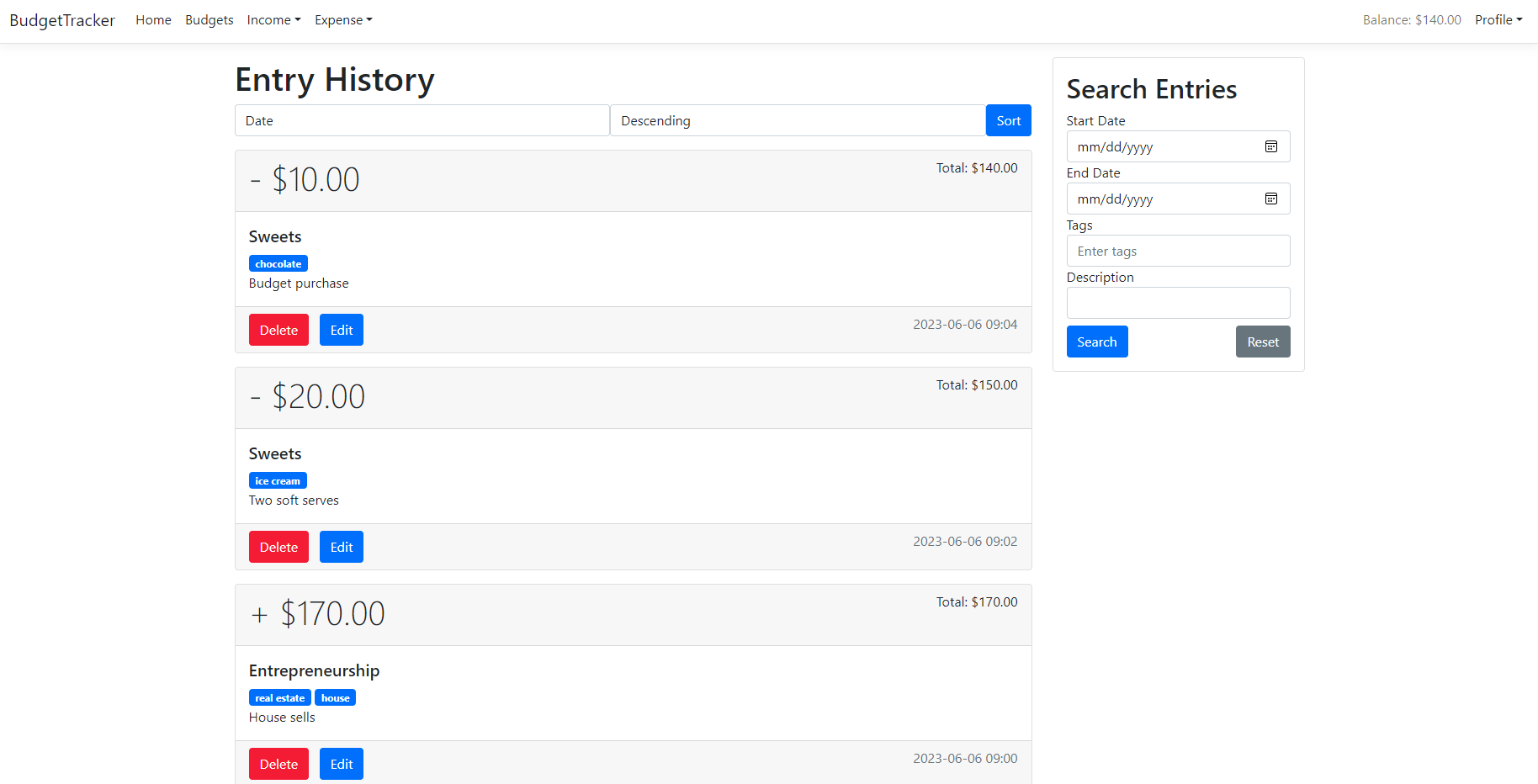
*Рис. 16. История транзакций до удаления записи.*



*Рис. 17. История транзакций после удаления записи*



*Рис. 18. Редактирование дохода “Enterpreneurship +200”*



*Рис. 19. История транзакций после изменения записи*

# Заключение

В ходе разработки веб-приложения BudgetTracker на основе ASP.NET Core MVC были достигнуты следующие результаты. Был создан функциональный и удобный в использовании веб-сайт, который позволяет пользователям отслеживать свой баланс, управлять тратами и доходами, а также управлять бюджетами.

В процессе разработки были сделаны следующие выборы:

* СУБД: Для хранения данных была выбрана SQL Server. SQL Server обеспечивает надежное хранение данных, обладает мощными возможностями для работы с реляционными базами данных и является хорошо поддерживаемым продуктом в экосистеме ASP.NET Core.
* Язык программирования: Для реализации логики приложения был выбран язык программирования C#. C# является официальным языком разработки для платформы .NET и предоставляет широкий набор инструментов и библиотек для разработки веб-приложений.

В целом, разработка веб-приложения BudgetTracker была успешной. Полученное приложение предоставляет пользователю возможность эффективно управлять своими финансами, отслеживать баланс, контролировать расходы и доходы, а также планировать бюджеты. Приложение обладает интуитивно понятным интерфейсом и предоставляет необходимую функциональность для удобного использования.

В заключение, разработка BudgetTracker была интересным и полезным опытом. Полученное приложение представляет собой удобный инструмент для управления финансами и может быть использовано как основа для дальнейшего развития и расширения функциональности.

Воспользоваться веб-приложением вы можете по ссылке: <https://alinurbudgettracker.azurewebsites.net/>

# Использованная литература

Онлайн-ресурсы:

1. ASP.NET MVC: <https://learn.microsoft.com/en-us/aspnet/mvc/>
2. ASP.NET Core Identity: <https://www.tutorialspoint.com/asp.net_core/asp.net_core_identity_overview.htm>

Книги:

1. Head First Design Patterns: A Brain-Friendly Guide by Eric Freeman (Author), Bert Bates (Author), Kathy Sierra (Author), Elisabeth Robson (Author)

**Задача:**

Создание приложения учета финансов

**Цель проекта:**

Создание подробного отчета, который предоставит обзор и анализ данных, связанных с учетом бюджета в вашем приложении. Отчет будет содержать информацию о текущем балансе пользователя, его тратах, заработках и бюджетных ограничениях.

**Задачи проекта:**

1. Вывести общую информацию:

Отчет должен предоставлять полезную информацию о финансовом состоянии пользователя и его бюджетных показателях.

Отчет должен быть структурирован и легко читаем.

1. Общий баланс:

Отчет должен содержать информацию о текущем балансе пользователя.

Пользователь должен видеть общую сумму доступных средств на своем счету.

1. Траты и доходы:

Отчет должен включать информацию о последних транзакциях пользователя, включая траты и доходы.

Для каждой транзакции должны быть указаны дата, категория, описание и сумма.

Транзакции могут быть отсортированы как по дате, так и по сумме, по убыванию и разрастанию.

1. Бюджеты на траты:

Отчет должен включать информацию о бюджетах пользователя на траты.

Для каждого бюджета должны быть указаны категория и доступная сумма.

Пользователь должен видеть, сколько из бюджета уже потрачено.

1. Дополнительные функции (по желанию):
2. По желанию можно добавить дополнительные функции, такие как фильтрация транзакций по дате или категории, возможность изменения бюджетов и т.д.