Муниципальное автономное общеобразовательное учреждение «Лицей № 17»

Тема проекта:

Разработка веб-приложения для учета финансов и автоматического распределения долгов между участниками мероприятий

Выполнил

Пацай Фёдор Андреевич

Обучающийся 10Б класса

МАОУ «Лицей № 17»

Научный руководитель:  
Паршев Александр Анатольевич

Учитель информатики

МАОУ «Лицей № 17»

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ…………………………………………………………………3

Глава 1 МЕТОДИКА ИССЛЕДОВАНИЯ……………………...…………5

* 1. Используемые технологии……………………………………….5
  2. Этапы создания проекта………………………………………….5
  3. Процесс тестирования……………………………………………6
  4. Обоснование выбора технологий………………………………..6
  5. Выбор архитектуры приложения………………………………..7

Глава 2 Процесс получения данных………………………………………9

2.1 Методы сбора данных……………………………………………9

2.2 Взаимодействие с пользователем……………………………….9

2.3 Структура данных………………………………………………10

2.4 Проблемы и ограничения………………………………………11

Глава 3 Р**еализация функционала**

**автоматического распределения долгов…………………………………13**

**3.1** Задачи автоматического распределения долгов……………….13

3.2 Автоматическое распределение долгов………………………..13

3.3 Ручное изменение долгов……………………………………….13

3.4 Пример работы с долгами………………………………………14

3.5 Визуализация данных………...…………………………………14

3.6 Проблемы при расчёте долгов………………………………….14

3.7 Итоги реализации………………………………………………..14

ЗАКЛЮЧЕНИЕ……………………………………………………………16

БИБЛИОГРАФИЧЕСКИЙ СПИСОК…………………………………….17

ПРИЛОЖЕНИЕ 1………………………………………………………….18

ВВЕДЕНИЕ

Проблема: в командных поездках, совместных мероприятиях и покупках часто возникают трудности с учётом общих расходов. Из-за отсутствия удобных инструментов появляются ошибки в расчётах, путаница и неравномерное распределение затрат. Например, участникам приходится вручную вести записи в блокнотах или таблицах, что не всегда удобно и приводит к недопониманию. В итоге процесс затягивается, вызывает лишние споры и негативные эмоции в группе.

Актуальность: совместные мероприятия и покупки давно стали неотъемлемой частью повседневной жизни, особенно среди молодых людей и активных групп. В таких ситуациях часто возникает необходимость чётко и справедливо распределять расходы между участниками. Однако традиционные способы учёта затрат, такие как бумажные записи и таблицы, устарели. Они требуют времени, неудобны и подвержены ошибкам. Современным пользователям нужен более эффективный и автоматизированный инструмент, который решит эти проблемы и упростит распределение расходов.

Цель: разработать удобное и интуитивно понятное веб-приложение для автоматизированного учёта совместных расходов. Приложение позволит: автоматически рассчитывать и распределять затраты между участниками, упрощать процесс расчётов и минимизировать трудозатраты, избегать ошибок и недопонимания при подсчётах. Решение сделает процесс управления общими расходами быстрым, прозрачным и комфортным для всех участников.

Задачи:

* Разработать веб-приложение
* Обеспечить удобный интерфейс
* Обеспечить защиту данных пользователей
* Провести тестирование и выявить возможные ошибки
* Реализовать систему аутентификации - чтобы доступ к данным был только у авторизированных пользователей

Объект исследования: процессы учета и распределения общих расходов в групповых мероприятиях и совместных покупках.

Предмет исследования: возможности автоматизации учета совместных расходов с использованием веб-приложения для повышения точности и удобства распределения затрат между участниками.

Методы исследования:

* Анализ
* Описание
* Сравнение
* Программирование

### ГЛАВА 1 МЕТОДИКА ИССЛЕДОВАНИЯ

### 1.1 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

Для разработки веб-приложения для учёта финансов использовались следующие технологии:

* + Django — фреймворк для разработки серверной части приложения. Он обеспечил быстрый процесс разработки, встроенные механизмы работы с базами данных и безопасность.
  + HTML, CSS, JavaScript — для создания клиентской части и обеспечения интерактивности интерфейса.
  + SQLite — использовалась для хранения данных в локальной базе данных.
  + Bootstrap — для упрощения разработки адаптивного дизайна.
  + Django ORM — для работы с базой данных через объектно-ориентированное программирование.

### 1.2 ЭТАПЫ СОЗДАНИЯ ПРОЕКТА

Процесс создания приложения был разделён на несколько ключевых этапов:

* Анализ требований: на этом этапе были собраны требования к функционалу приложения. Определены основные задачи, такие как учет расходов, создание отчетов и управление пользователями.
* Проектирование интерфейса и архитектуры: разработка макета интерфейса, определение ключевых элементов, таких как форма для ввода данных и таблицы для отображения расходов.
* Разработка: написание серверной логики на Django, моделирование базы данных, создание шаблонов для отображения информации.
* Тестирование и исправление ошибок: после разработки приложение тестировалось с реальными пользователями, которые предоставляли обратную связь по функциональности и удобству.
* Оптимизация и улучшения: на основе отзывов пользователей были внесены изменения в интерфейс и логику работы приложения.

### 1.3 ПРОЦЕСС ТЕСТИРОВАНИЯ

В процессе разработки приложения была проведена серия тестов для проверки его функциональности и стабильности. Написаны тесты для ключевых частей системы, включая:

* Проверку корректности добавления и отображения транзакций.
* Тестирование правильности формирования отчетов.
* Оценка безопасности данных при вводе и сохранении информации.

Тесты помогли выявить ошибки и несоответствия в логике работы приложения, что позволило улучшить стабильность и удобство использования системы.

### 1.4 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

Django был выбран за его высокую продуктивность и гибкость в создании веб-приложений. Он предоставляет множество встроенных инструментов, таких как системы аутентификации и авторизации, ORM для работы с базой данных и средства защиты от распространённых угроз, что ускоряет разработку и снижает количество ошибок. HTML, CSS, JavaScript используются для создания клиентской части приложения. HTML отвечает за структуру страниц, CSS — за их оформление, а JavaScript — за динамическое взаимодействие с пользователем. Эта комбинация технологий позволяет создать удобный и интуитивно понятный интерфейс для пользователя. SQLite была выбрана в качестве базы данных, так как она является легким и простым в использовании решением для небольших приложений. Это позволяет сократить сложность конфигурации базы данных на этапе разработки. В будущем, при необходимости масштабирования проекта, можно рассмотреть более мощные СУБД, такие как PostgreSQL. Django ORM использовался для работы с базой данных через объектно-ориентированное программирование, что делает код более читабельным и простым для поддержки. ORM позволяет работать с данными через модели, что упрощает взаимодействие с базой данных и сокращает количество ошибок.

### 1.5 ВЫБОР АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

Приложение построено по принципу архитектуры **Model-View-Template (MVT)**, которая является аналогом популярной архитектуры **MVC (Model-View-Controller)** и применяется в Django. В этой архитектуре:

* **Model (Модель)** — это компоненты, отвечающие за логику работы с данными. В Django модели описываются в виде классов и управляют взаимодействием с базой данных. В данном проекте используются модели для хранения данных о событиях, пользователях, расходах и долгах.
* **View (Представление)** — это части приложения, которые отвечают за обработку пользовательских запросов и отображение данных. В данном проекте представления реализованы как функции или классы, которые получают данные из модели и передают их в шаблоны для отображения.
* **Template (Шаблон)** — это компоненты, отвечающие за отображение информации пользователю. Django использует систему шаблонов для генерации HTML-страниц, которые затем отправляются клиенту.

Также стоит отметить использование встроенной системы маршрутизации в Django, которая позволяет эффективно связывать URL-адреса с нужными представлениями и шаблонами, а также механизмы безопасности, которые позволяют предотвращать распространенные атаки на приложение, такие как CSRF.

### 1.6. БЕЗОПАСНОСТЬ ПРИЛОЖЕНИЯ

Безопасность приложения была важным аспектом на всех этапах разработки. Были реализованы следующие меры безопасности:

* **Хеширование паролей**: для защиты паролей пользователей используется встроенная система аутентификации Django, которая обеспечивает хеширование паролей перед их сохранением в базе данных с использованием алгоритма PBKDF2.
* **Использование HTTPS**: все взаимодействие с сервером осуществляется через защищенное соединение HTTPS, что предотвращает утечку данных через незащищенные каналы связи.
* **Защита от SQL-инъекций**: Django ORM автоматически предотвращает SQL-инъекции, обрабатывая все запросы через параметризацию, что исключает возможность внедрения вредоносного кода в запросы.
* **CSRF защита**: В Django реализована защита от атак Cross-Site Request Forgery (CSRF), что исключает возможность выполнения нежелательных действий от имени пользователя без его согласия.

### ГЛАВА 2 ПРОЦЕСС ПОЛУЧЕНИЯ ДАННЫХ

### 2.1 МЕТОДЫ СБОРА ДАННЫХ

Данные о расходах собирались через форму, доступную только авторизованным пользователям. После входа в систему, пользователь мог добавить транзакцию о расходах, заполнив обязательные поля. В процессе ввода данных пользователь указывал название события, сумму расхода и участников, которые принимали участие в затратах. Важно, что если пользователь забывал ввести какую-либо информацию (например, сумму или участников), система предупреждала его соответствующими сообщениями, чтобы обеспечить корректность ввода данных.

Данные о расходах вводились вручную, и каждый расход привязывался к конкретному событию, в котором участвовали пользователи. Пользователи могли добавлять информацию о транзакциях, таких как:

* + Название события (например, «Пикник» или «Вечеринка»).
  + Сумма расхода (например, 1000 рублей).
  + Участники (пользователи, которые принимали участие в расходах, с возможностью указания доли каждого). Кроме того, пользователи могли указывать, кто является инициатором расхода (создателем), чтобы система могла правильно распределить долг между участниками.

### 2.2 ВЗАИМОДЕЙСТВИЕ С ПОЛЬЗОВАТЕЛЕМ

Интерфейс системы был разработан так, чтобы максимально упростить процесс ввода данных для пользователей. Пользователь может выбрать участников для расхода из списка участников мероприятия, к которому привязан расход. Этот список автоматически генерируется на основе информации о пользователях, зарегистрированных в конкретном событии. Если пользователь вводит некорректные данные (например, отрицательную сумму или не указывает обязательные поля), система использует стандартные механизмы проверки данных, предоставляемые Django. В частности, для проверки правильности данных используются такие поля, как:

* + CharField, DecimalField и другие поля Django, которые автоматически проверяют корректность формата данных (например, чтобы сумма была положительным числом).
  + blank=False и null=False для обязательных полей, чтобы гарантировать, что пользователи не смогут оставить важные поля пустыми. В случае возникновения ошибок, система выводит соответствующие сообщения об ошибке, объясняя пользователю, что именно нужно исправить. Пользователи могут редактировать введенные данные с помощью тех же форм, которые использовались для их первоначального ввода. Это позволяет им корректировать информацию в случае ошибок или изменения условий (например, добавления новых участников или изменения суммы).

### 2.3 СТРУКТУРА ДАННЫХ

В проекте используется несколько моделей для организации данных в базе данных, которые тесно связаны между собой. Ниже приведены ключевые модели:

1. Модель Events:

* + title — название события.
  + user — ссылка на пользователя, создавшего событие.

2. Модель Users:

* + username — имя пользователя.
  + event — ссылка на событие, к которому принадлежит пользователь.

3. Модель Expenses:

* + event — ссылка на событие, к которому принадлежит расход.
  + name — название расхода (например, «Покупка еды»).
  + creator — ссылка на пользователя, создавшего расход.
  + price — сумма расхода.
  + members — ссылка на пользователей, которые участвуют в расходе.

4. Модель Debts:

* + debtor — ссылка на пользователя, который должен деньги.
  + payer — ссылка на пользователя, который оплатил расход.
  + debt — сумма долга.
  + purchase — ссылка на расход, из-за которого возник долг.
  + event — ссылка на событие, к которому относится долг.

Модель Expenses связана с моделями Events и Users и отображает сам расход, его создателя и участников. Модель Debts вычисляет, сколько каждый пользователь должен заплатить в зависимости от того, кто создал расход и кто участвовал в нем. В системе применяется метод create\_debts\_for\_members, который автоматически генерирует долги между участниками. Эта структура данных позволяет эффективно управлять расходами и долгами, а также поддерживать связь между пользователями и событиями. Также важно отметить, что система позволяет гибко управлять распределением долгов между участниками. Например, если сумма расхода не равномерно распределяется между участниками (например, 1000 рублей делятся на 3 человека, но в зависимости от вклада каждого участника доли могут быть разными), создатель события может вручную изменить суммы долгов для каждого участника. Это даёт возможность точнее отражать реальное распределение расходов, особенно если один из участников понёс большую часть расходов, чем другие. В случае изменений долга, система автоматически пересчитывает остаточные суммы для всех участников.[1]

### 2.4 ПРОБЛЕМЫ И ОГРАНИЧЕНИЯ

В процессе разработки веб-приложения возникли несколько проблем и ограничений, которые необходимо учитывать:

* **Ограничения SQLite**: для небольших проектов SQLite является удобной и быстрой базой данных, однако она имеет ограничения на масштабируемость. При увеличении количества данных или пользователей система может начать работать медленно. В будущем планируется перейти на более мощную СУБД, например, PostgreSQL или MySQL, для улучшения производительности.
* **Проблемы с валидацией данных**: В процессе разработки возникали трудности с проверкой и валидацией данных, особенно при вводе больших объемов информации. Это потребовало дополнительных усилий на этапе тестирования и отладки системы, чтобы избежать ошибок ввода и обработки данных.
* **Использование ограниченных средств UI/UX**: Приложение было разработано с минималистичным интерфейсом, чтобы ускорить процесс разработки. Однако в будущем планируется улучшить пользовательский интерфейс, добавив больше визуальных и функциональных элементов для улучшения опыта пользователя.
* **Многозадачность и поддержка множества пользователей**: Система пока не оптимизирована для работы с большим количеством одновременных пользователей. Для повышения производительности и масштабируемости можно будет использовать технологии, такие как кэширование и распределенные базы данных.

### **ГЛАВА 3 РЕАЛИЗАЦИЯ ФУНКЦИОНАЛА АВТОМАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ДОЛГОВ**

3.1 ЗАДАЧИ АВТОМАТИЧЕСКОГО РАСПРЕДЕЛЕНИЯ ДОЛГОВ

Основной задачей функционала автоматического распределения долгов является корректное вычисление долга каждого участника на основе внесённых данных о расходах. Система должна автоматически учитывать вклад каждого участника в общий расход, а также возможность изменения этих данных вручную в случае необходимости.

3.2 АВТОМАТИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ДОЛГОВ

Когда пользователь добавляет новый расход, система рассчитывает долги для всех участников, пропорционально их участию в расходах. Для этого используется метод create\_debts\_for\_members, который рассчитывает долю каждого пользователя на основе суммы расхода и количества участников. Система автоматически распределяет сумму между участниками и создаёт соответствующие записи о долгах в базе данных.

3.3 РУЧНОЕ ИЗМЕНЕНИЕ ДОЛГОВ

Если распределение долга не соответствует реальности (например, расход был неравномерно распределён между участниками), создатель события может вручную изменить сумму долга каждого участника. Это позволяет более точно отразить реальную ситуацию, если один из участников заплатил больше, чем другие, или если возникли другие обстоятельства, влияющие на распределение расходов. Для изменения долга создатель события может зайти в раздел управления расходами и отредактировать сумму долга для каждого пользователя, внося нужные корректировки. После внесения изменений система автоматически пересчитывает все задолженности и обновляет информацию о долгах.

3.4 ПРИМЕР РАБОТЫ С ДОЛГАМИ

Возьмём пример мероприятия, на котором было потрачено 1000 рублей. Участниками являются три человека: пользователь A, B и C. Система автоматически рассчитывает, что каждый должен заплатить 333,33 рубля. Однако пользователь A, по договорённости, заплатил 500 рублей, а остальные — 250 рублей каждый. В такой ситуации создатель события может вручную изменить долги, установив для пользователя A долг в размере 0 рублей, для пользователя B — 250 рублей, а для пользователя C — 500 рублей.

3.5 ВИЗУАЛИЗАЦИЯ ДАННЫХ

Для удобства пользователя все долги отображаются в интерфейсе приложения в виде таблицы. В таблице указываются: название события, участники мероприятия, сумма долга каждого участника, суммарная сумма расходов и доля каждого участника. Это помогает пользователю наглядно отслеживать долговые обязательства и вовремя вносить изменения, если ситуация меняется.

3.6 ПРОБЛЕМЫ ПРИ РАСЧЁТЕ ДОЛГОВ

Несмотря на продуманную логику распределения долгов, были выявлены некоторые проблемы, с которыми столкнулись пользователи на этапе тестирования:  
**Некорректный ввод данных** — если участники не точно указывали суммы расходов или доли участия, система могла неправильно распределить долги.  
**Ошибки в пересчёте долгов** — при внесении изменений вручную в систему часто не учитывалось правильное обновление информации для всех участников.  
Для устранения этих проблем в ходе тестирования были доработаны алгоритмы вычисления долгов и добавлены дополнительные проверки данных.

3.7 ИТОГИ РЕАЛИЗАЦИИ

Функционал автоматического распределения долгов, с возможностью ручного изменения суммы долга, значительно упрощает процесс учёта расходов в мероприятиях. Пользователи могут как полагаться на автоматический расчёт долгов, так и вручную скорректировать их в случае необходимости. Это позволяет более гибко подходить к учёту финансов и устранять потенциальные ошибки в процессе расчётов.

### ЗАКЛЮЧЕНИЕ

В ходе работы над проектом были достигнуты следующие результаты: д**остижение поставленных целей:** приложение успешно реализует систему, позволяющую автоматически делить расходы и распределять долги между участниками групповых мероприятий с возможностью ручной корректировки данных. Использование Django, HTML, CSS, JavaScript и SQLite обеспечило высокую безопасность и удобство эксплуатации системы. **Практическое использование:** разработанное решение имеет высокую практическую ценность для организаторов групповых мероприятий, где важен точный учёт расходов и распределение долгов. Приложение способствует прозрачному управлению финансовыми потоками при организации корпоративных встреч, поездок, семейных торжеств и других мероприятий. **Направления дальнейшего развития:** в будущем целесообразно улучшить пользовательский интерфейс для повышения удобства работы, а также интегрировать дополнительные аналитические инструменты для расширенного анализа и отчетности по финансовым операциям.

Таким образом, результаты проекта находятся в логической связи с поставленными задачами и полностью соответствуют заявленной цели, демонстрируя эффективность выбранных технологий и открывая перспективы для дальнейших разработок в области автоматизации учета расходов на групповых мероприятиях.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Винсент У. С. Django for Beginners: Build websites with Python and Django */* У. С. Винсент. – М.: Питер, 2018. – 350 с.
2. Винсент У. С. Django for Professionals: Production websites with Django/ У. С. Винсент. – М.: Питер, 2020. – 400 с.
3. Гринфилд Д. Р., Гринфилд А. Р. Two Scoops of Django 3.x: Best Practices for Django/ Д. Р. Гринфилд, А. Р. Гринфилд. – М.: Питер, 2019. – 550 с.
4. Меле А. Django 3 By Example: Build powerful and reliable websites with Django 3 / А. Меле. – М.: Питер, 2019. – 500 с.
5. Персиваль Г. Разработка через тестирование с Python: использование Django, Selenium и JavaScript */* Г. Персиваль. – М.: Питер, 2017. – 624 с.

ПРИЛОЖЕНИЕ 1

Пример кода модели

