

**Министерство образования и науки Российской Федерации**  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

---

Институт Кибернетики  
Направление подготовки Информатика и вычислительная техника  
Кафедра Оптимизации систем управления

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту**

по дисциплине «Технологии разработки программного обеспечения»  
на тему «Сервис учета личных финансов "MoneyTalks"»

Выполнили студенты гр. <u>8ВМ44</u>	_____	<u>В.В. Асмоловский</u>
	_____	<u>Д.А. Логинов</u>
	_____	<u>В.Н. Мухаметшин</u>
	_____	<u>А.Ю. Пилецкая</u>
	_____	<u>Д.В. Цыбин</u>

Дата сдачи пояснительной записки преподавателю \_\_\_\_\_ 20\_\_ г.

Руководитель ассистент каф. ОСУ И.А. Заикин

\_\_\_\_\_  
(Оценка руководителя)

\_\_\_\_\_  
(Подпись)

\_\_\_\_\_ 20\_\_ г.

(Дата проверки)

Курсовой проект студенты В.В. Асмоловский, Д.А. Логинов, В.Н. Мухаметшин, А.Ю. Пилецкая, Д.В. Цыбин выполнили и защитили  
с оценкой \_\_\_\_\_.

\_\_\_\_\_ 20\_\_ г.

(дата защиты )

Томск 2014 г.

## СОДЕРЖАНИЕ

Введение.....	4
1 Требования к программе.....	6
1.1 Требования к программе .....	6
1.2 Варианты использования .....	9
1.3 Спецификации вариантов использования .....	10
1.3.1 Спецификация «Управление счетами».....	10
1.3.2 Спецификация «Управление категориями».....	11
1.3.3 Спецификация «Управление местами» .....	12
1.3.4 Спецификация «Управление транзакциями».....	13
2 Анализ.....	16
2.1 Пользователь .....	16
2.2 Место .....	17
2.3 Категория .....	17
2.4 Счёт.....	17
2.5 Транзакция .....	17
2.6 Ассоциации между классами.....	18
3 Проектирование.....	20
3.1 Проектные классы.....	20
3.2 Диаграмма пакетов системы .....	22
3.3 Диаграммы последовательностей для операций проектных классов	23
4 Реализация.....	33
4.1 Тестирование .....	33

4.1.1	Модульное тестирование .....	33
4.2	Покрывтие кода.....	36
4.3	Непрерывная интеграция .....	37
4.4	Развёртывание .....	39
5	Документация .....	40
5.1	Назначение программы .....	40
5.2	Условия выполнения программы .....	40
5.3	Выполнение программы.....	41
5.3.1	Регистрация уникального пользователя.....	41
5.3.2	Авторизация пользователя.....	42
5.3.3	Создание нового уникального счета.....	43
5.3.4	Создание нового уникального места .....	43
5.3.5	Создание новой уникальной категории.....	44
5.3.6	Создание подкатегории, уникальной в рамках родительской категории .....	45
5.3.7	Создание записи о списании средств со счета с указанием места, категории, суммы и комментария.....	46
5.3.8	Изменение или удаление счетов, мест, категорий и транзакций .....	47
5.4	Сообщения программы.....	48
	Заключение .....	49
	Список использованных источников .....	50
	Приложение А .....	52

## ВВЕДЕНИЕ

Экономика нашего времени основывается на товарно-денежных отношениях. Общество потребления, возникшее в результате развития капитализма, захватывает все большее количество неокрепших умов современной России [1].

Расходы на товары и услуги во всем мире возросли более чем в четыре раза за последние 40 лет [2]. Вместе с расходами увеличивается и количество мест хранения накопленных непосильным трудом сбережений: зарплатная карта, стипендиальная карта, множество карт по специальным предложениям от банков, банковская ячейка или конверт с деньгами в ящике стола, полученный вчера от начальника, – их большое количество затрудняет осознание общих активов и, соответственно, адекватности осуществляемых расходов.

При таком положении дел нередко возникают ситуации, когда на утро после хорошей вечеринки потраченная вчера сумма превышает недельные расходы на продукты. Или купленный в подарок девушке iPhone™ вдруг обращается походом в некую печально известную контору «Деньги в долг». Науке известны и другие случаи, однако большинство из них можно было бы избежать, зная и адекватно оценивая свои финансовые возможности.

Один из способов решения этой задачи – учет личных финансов, наиболее эффективным видом которого представляется учет с помощью электронных устройств: будь то персональный компьютер, планшет или мобильное устройство.

Целью данного курсового проекта является разработка простого и удобного для ежедневного использования сервиса учета личных финансов, доступного с различных устройств.

В соответствии с поставленными целями курсового проекта были выбраны наиболее подходящие средства разработки:

- язык программирования Python, как активно развивающийся язык, призванный повысить скорость разработки и читаемость кода;
- каркас веб-приложений Django, использующий шаблон проектирования MVC;
- интегрированная среда разработки PyCharm, поддерживающая разработку на Django.

Перечисленные средства разработки позволяют реализовать поставленную задачу и достичь установленной цели.

# **1 ТРЕБОВАНИЯ К ПРОГРАММЕ**

Сервис MoneyTalks предназначен для учета личных финансов пользователя.

Основным целевым потребителем решения является среднестатистический молодой (16-30 лет) человек с регулярным доходом и необходимостью контролировать свои финансы.

Задачи, решаемые сервисом:

- 1) Сохранение информации о движении средств (транзакциях);
- 2) Привязка транзакций к месту и времени;
- 3) Разграничение затрат по категориям и счетам пользователя;
- 4) Детализация затрат за счет введения подкатегорий;
- 5) Максимальная персонализация.

## **1.1 Требования к программе**

- 1) Требования к функциональным характеристикам.
  - Сервис должен выполнять следующие функции:
    - Регистрация уникального пользователя;
    - Авторизация пользователя;
    - Создание нового уникального счета;
    - Создание нового уникального места;
    - Создание новой уникальной категории;
    - Создание подкатегории, уникальной в рамках родительской категории;
    - Создание записи о списании средств со счета с указанием места, категории, суммы и комментария;
    - Создание записи о поступлении средств на счет с указанием места, времени, категории, суммы и комментария;
    - Перевод средств со счета на счет;
    - Изменение/удаление счетов, мест, категорий и транзакций.

Ввод данных должен осуществляться посредством клавиатуры (физической или экранной) и манипулятора (компьютерная мышь, тачпад, сенсорный экран и т.п.).

## 2) Требования к надежности.

Для обеспечения надежного функционирования системы требуется развернуть ее на сервере с бесперебойной подачей энергии. Сервер должен работать под управлением последней стабильной версии Windows Server, либо под управлением последней стабильной версии Linux.

При вводе некорректной информации сервис должен выдать предупреждающее сообщение и предложить пользователю повторить ввод.

Основные сценарии ввода некорректных данных:

- Ввод неуникального имени;
- Отсутствие введенной информации.

Восстановление после некритических ошибок должно происходить меньше, чем за 30с. Восстановление после критических ошибок, приведших к неработоспособности системы, должно происходить не более чем за сутки.

## 3) Условия эксплуатации

Компьютер предназначен для работы в закрытом отапливаемом помещении при следующих условиях окружающей среды:

- температура окружающего воздуха от +10°C до +35°C;
- атмосферное давление от 630 до 800 мм ртутного столба;
- относительная влажность воздуха не более 80%;
- запыленность воздуха не более 0,75 мг/м<sup>3</sup>;
- кроме этого, в воздухе не должно быть паров агрессивных жидкостей и веществ, вызывающих коррозию [3].

Обслуживающий персонал должен состоять из как минимум одного специалиста по системному администрированию. Персонал должен обеспечить бесперебойную работу сервера и своевременное исправление ошибок.

#### 4) Требования к составу и параметрам технических средств

Сервис должен выполняться на следующем аппаратном и программном обеспечении (минимальные требования):

- Для серверов под управлением Windows Server 2008 и позднее:
  - Процессор – 1 ГГц;
  - Память ОЗУ – 512 Мб;
  - Свободное место на диске – 8 Гб;
  - Привод – устройство чтения DVD дисков;
  - Дисплей и периферийные устройства – монитор Super VGA (800x600) или более высокого разрешения, клавиатура, мышь Microsoft или совместимое указывающее устройство [4].
- Для серверов под управлением Ubuntu Server:
  - Процессор – 300 МГц;
  - Память ОЗУ – 64-128 Мб;
  - Свободное место на диске – 1 Гб;
  - Видеокарта и монитор с разрешением 640x480;
  - CD и DVD привод, USB порт [5].

#### 5) Требования к информационной и программной совместимости

Реализация сервиса должна осуществляться на языке Python, в среде PyCharm (Community или Professional версия) или Visual Studio с установленным фреймворком. Версия интерпретатора Python не ниже 3.2.

Для разработки системы следует использовать фреймворк Django версии не ниже 1.7.



При разработке следует придерживаться стандартов форматирования кода, принятого для языка Python. При проектировании приложения необходимо придерживаться шаблона MVC (Model-View-Controller).

#### 6) Требования к программной документации

Документация представляет отчет по курсовой работе, состоящий из следующих частей:

- Титульный лист;
- Содержание;
- Введение;
- Требования к программе;
- Анализ;
- Проектирование;
- Реализация;
- Документация;
- Заключение;
- Список используемых источников.

#### 7) Стадии и этапы разработки

Сроки разработки: с 1 сентября по 6 декабря 2014г.

Основные этапы разработки сервиса:

- Описание требований и проектирование. Документы на выходе:
  - UML диаграммы вариантов использования, проектных классов, последовательностей.
- Реализация. На выходе должен быть рабочий прототип.
- Тестирование. Требуется покрыть тестами 75% кода.
- Настройка система непрерывной интеграции [6].

## 1.2 Варианты использования

На основе требований к системе, рассмотренных выше, было проведено моделирование диаграммы вариантов использования (рисунок 1Рисунок 1).

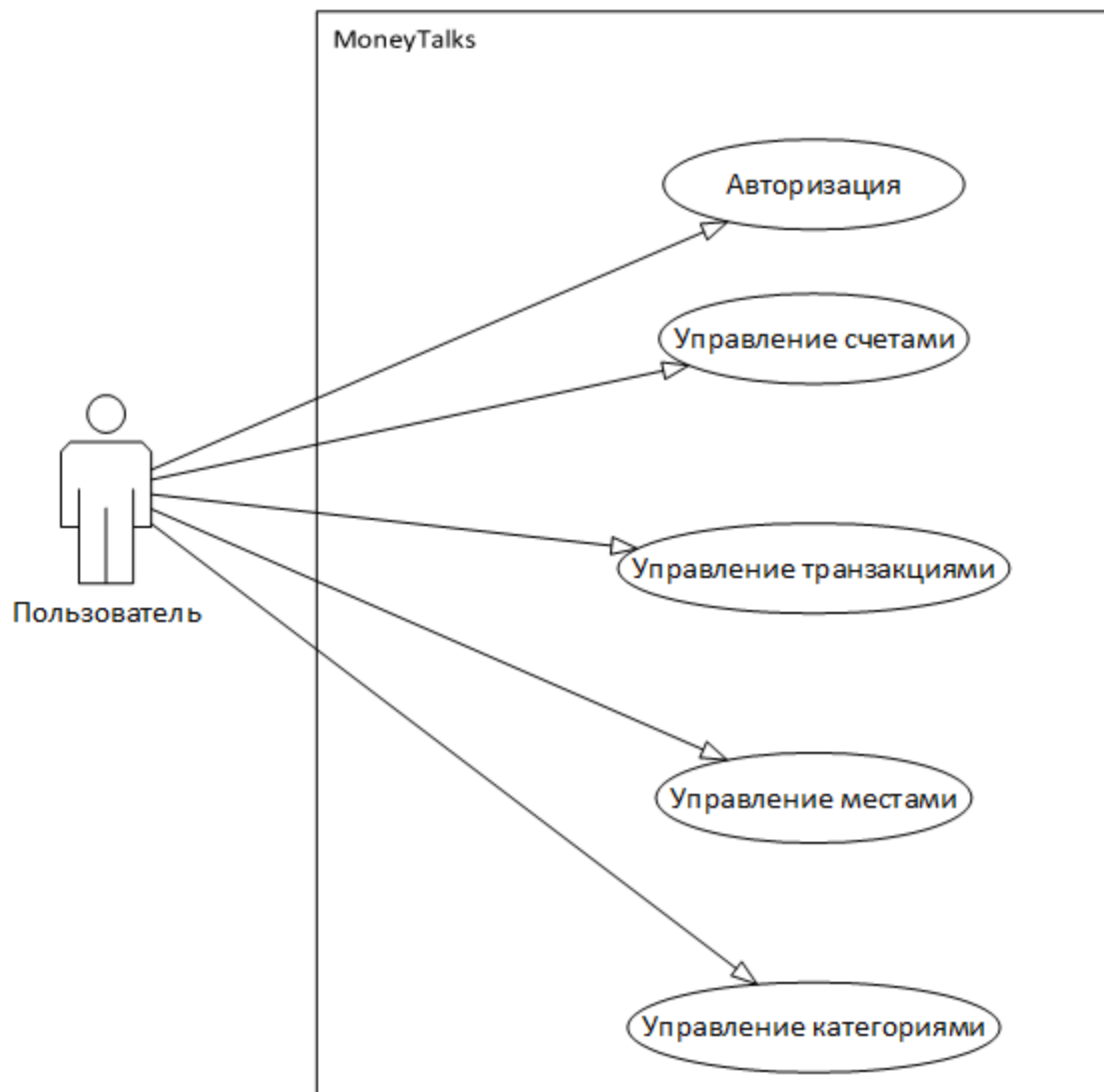


Рисунок 1 – Диаграмма вариантов использования системы MoneyTalks

### 1.3 Спецификации вариантов использования

Далее представлены спецификации ранее описанных вариантов использования.

#### 1.3.1 Спецификация «Управление счетами»

Краткое описание:

Управление счетами пользователем.

Основное действующее лицо:

Пользователь.

Второстепенные действующие лица:

Сервис учёта личных финансов.

Предусловия:

1. Открытие главной страницы Сервиса учёта личных финансов.

Основной поток:

1. Вариант использования начинается, когда Пользователь открывает страницу управления счетами.
2. Пользователь выбирает действие со счетом.
3. Сервис отображает список измененных счетов.

Альтернативные потоки:

- 2.1. Пользователь выбирает опцию «Добавить».

- 2.1.1. Пользователь вводит название счета.

- 2.1.2. Сервис проверяет уникальность названия счета среди счетов Пользователя.

- 2.1.3. Пользователь указывает начальный баланс.

- 2.1.4. Сервис добавляет новый счет.

- 2.2. Пользователь выбирает опцию «Изменить».

- 2.2.1. Пользователь переименовывает счет.

- 2.2.2. Сервис проверяет уникальность названия счета среди счетов Пользователя.

- 2.2.3. Сервис сохраняет изменения.

- 2.3. Пользователь выбирает опцию «Удалить».

- 2.3.1. Сервис запрашивает подтверждение действия пользователя.

- 2.3.2. Пользователь подтверждает удаление счета.

- 2.3.3. Сервис удаляет счет.

Постусловия:

Нет.

### **1.3.2 Спецификация «Управление категориями»**

Краткое описание:

Управление категориями пользователя.

Основное действующее лицо:

Пользователь.

Второстепенные действующие лица:

Сервис учёта личных финансов.

Предусловия:

1. Открытие главной страницы Сервиса учёта личных финансов.

Основной поток:

1. Вариант использования начинается, когда Пользователь выбирает опцию «Категории».
2. Выбор действий с категориями.
3. Сервис отображает дерево категорий.

Альтернативные потоки:

- 2.1. Пользователь выбирает опцию «Создать».

- 2.1.1. Пользователь вводит название категории.

- 2.1.2. Пользователь выбирает родительскую категорию.

- 2.1.3. Система создает категорию.

- 2.2. Пользователь выбирает опцию «Удалить».

- 2.2.1. Система запрашивает подтверждение удаления.

- 2.2.2. Система удаляет категорию.

Постусловия:

Нет.

### **1.3.3 Спецификация «Управление местами»**

Краткое описание:

Управление местами пользователя.

Основное действующее лицо:

Пользователь.

Предусловие:

Пользователь открыл главную Страницу сервиса личных финансов.

1. Пользователь открыл страницу управления местами.

2. Пользователь выбирает действие с местом.
3. Если пользователь выбирает добавить место, то:
  - 3.1. Сервис запрашивает имя места.
  - 3.2. Пользователь вводит название места.
  - 3.3. Сервис проверяет уникальность названия места среди мест пользователя.
  - 3.4. Если название места уникально,
    - 3.4.1. то сервис добавляет новое место
  - 3.5. Иначе:
    - 3.5.1. Переход к шагу 3.1.
4. Если пользователь выбирает (опцию) изменить место, то:
  - 4.1. Сервис запрашивает новое имя места.
  - 4.2. Пользователь вводит имя места.
  - 4.3. Сервис проверяет уникальность названия места среди мест пользователя.
  - 4.4. Если название места уникально,
    - 4.4.1. то Сервис изменяет название места,
  - 4.5. Иначе:
    - 4.5.1. Переход к шагу 4.1.
5. Если пользователь выбирает удалить место, то
  - 5.1. Сервис запрашивает подтверждение действия пользователя.
  - 5.2. Пользователь подтверждает удаление места.
  - 5.3. Сервис удаляет место.
6. Сервис отображает список изменённых мест.

Постусловия:

Нет.

#### **1.3.4 Спецификация «Управление транзакциями»**

Краткое описание:

Управление транзакциями пользователя.

Основное действующее лицо:

Пользователь.

Второстепенное действующее лицо:

Система учета личных финансов.

Предусловие:

1. Открытые главной страницы системы учета личных финансов.
2. Наличие хотя бы одного счета пользователя.

Основной поток:

1. Вариант использования начинается, когда Пользователь выбирает опцию «Транзакции».
2. Пользователь выбирает действие с транзакциями.
3. Система отображает список измененных транзакций.

Альтернативные потоки:

- 2.1. Пользователь выбирает опцию «Создать».
  - 2.1.1. Пользователь выбирает исходный счет.
  - 2.1.2. Пользователь выбирает целевой счет.
  - 2.1.3. Пользователь выбирает место.
  - 2.1.4. Пользователь выбирает категорию.
  - 2.1.5. Пользователь вводит сумму транзакции.
  - 2.1.6. Пользователь вводит комментарии.
  - 2.1.7. Система проверяет исходный и целевой счет на одинаковость.
  - 2.1.8. Система сохраняет изменения.
- 2.2. Пользователь выбирает опцию «Все».
  - 2.2.1. Система отображает все транзакции.
- 2.3. Пользователь выбирает опцию «Перевод».
  - 2.3.1. Система отображает транзакции между личными счетами Пользователя.
- 2.4. Пользователь выбирает опцию «Доход».

- 2.4.1. Система отображает поступления на счета Пользователя.
- 2.5. Пользователь выбирает опцию «Расход».
  - 2.5.1. Система отображает списания со счетов Пользователя.
- 2.6. Пользователь выбирает опцию «Изменить» для конкретной транзакции.
  - 2.6.1. Система открывает окно ввода данных с заполненными полями.
  - 2.6.2. Пользователь меняет параметры существующей транзакции.
  - 2.6.3. Система сохраняет изменения.
- 2.7. Пользователь выбирает опцию «Удалить» для конкретной транзакции.
  - 2.7.1. Система запрашивает подтверждение удаления.
  - 2.7.2. Система удаляет транзакцию.

Постусловие:

Нет [7, с. 101-103].

## 2 АНАЛИЗ

После постановки цели и определения требований к программе осуществляется анализ предметной области и создание аналитической модели.

В ходе этого процесса необходимо выявить классы анализа, представляющие четкую, точно определенную абстракцию предметной области. Классы анализа должны проецироваться на реальные бизнес понятия.

Классы анализа отличаются от классов проектирования тем, что последние вытекают из области решения и ограничены выбранными средствами разработки.

Класс анализа должен обладать именем, атрибутами и операциями, при этом точно и лаконично моделировать один аспект предметной области с точки зрения создаваемой системы [7].

В ходе выполнения курсового проекта были выявлены следующие классы:

- Пользователь;
- Место;
- Категория;
- Счёт;
- Транзакция.

### 2.1 Пользователь

Пользователь сервиса, осуществляющий учет личных финансов. Обладает набором счетов, управляет уникальными для него местам, категориями. Пользователи могут зарегистрироваться или войти в сервис, если уже зарегистрированы.



## **2.2 Место**

Место – это организация или человек, который является контрагентом совершаемой транзакции. Это может быть как магазин, в котором совершается покупка, так и место работы пользователя, где он получил заработную плату, например: «Лента, гипермаркет», «ТПУ».

Места различаются названиями и могут быть изменены или удалены, а также добавлены новые по усмотрению пользователя.

## **2.3 Категория**

Категория транзакции – это логическая группа доходов или расходов, например: «Продукты», «Работа». Также может быть подгруппа – например «Молочные продукты» в группе «Продукты».

Категории, также как и места, различаются названием и управляются пользователем. Для ускорения начала работы с сервисом должны быть предусмотрены стандартные или популярные категории. Также к категориям могут быть добавлены подкатегории, обладающие теми же свойствами.

## **2.4 Счёт**

Счет пользователя, к которому относятся осуществляемые доходы или расходы. К примеру, это может быть «Кошелек», «Сберкнижка», «Карта Газпромбанк».

## **2.5 Транзакция**

Транзакция – это, непосредственно, операция добавления или списания денег со счета. Также это может быть операция перевода денег с одного счета на другой. Ее можно отнести к той или иной категории, связать с тем или иным местом.

На рисунке 2 представлена диаграмма описанных классов и ассоциации между ними.

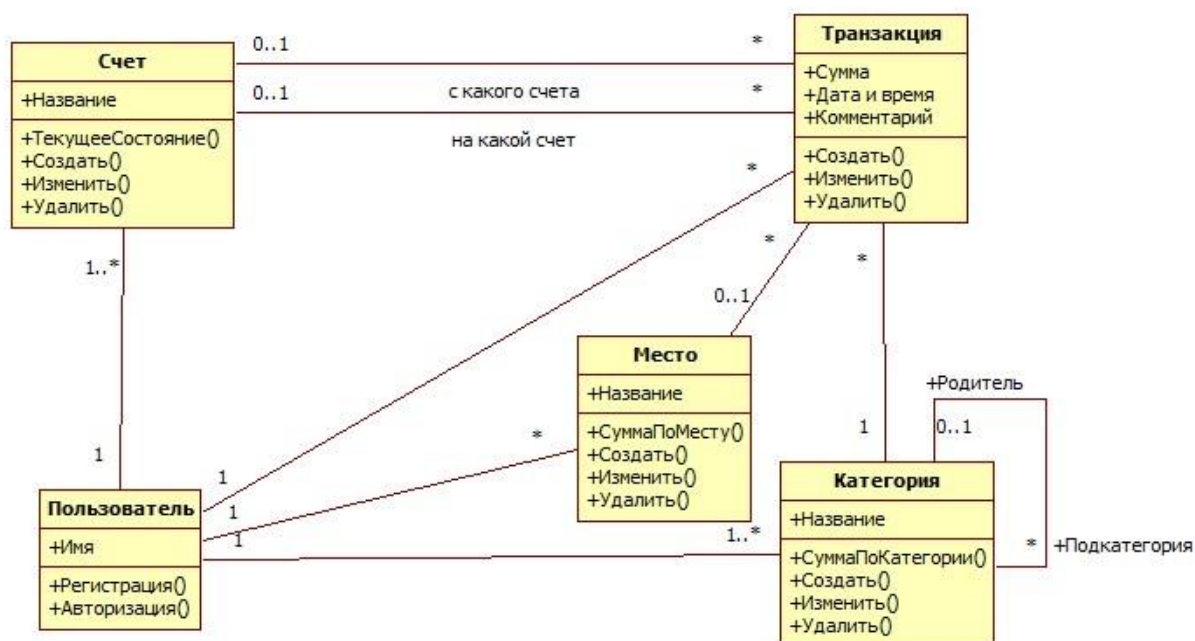


Рисунок 2 – Диаграмма классов анализа

## 2.6 Ассоциации между классами

Ассоциация между «Пользователь» и «Счёт» необходима для привязки одного или нескольких счетов к конкретному пользователю системы.

Ассоциация между «Пользователь» и «Место» необходима для создания каждому пользователю системы уникальных мест и организаций-контрагентов. Причем у пользователя может и не быть мест вовсе.

Ассоциация между «Пользователь» и «Категория» необходима для создания каждому пользователю уникального дерева категорий. У пользователя должна быть как минимум одна категория.

Рефлексивная ассоциация класса «Категория» необходима для создания иерархической структуры категорий. В данном случае один объект выступает в роли родителя другого и может обладать от 0 до бесконечности потомков. Тогда как потомок может или иметь родителя, при этом только одного, или не иметь их вовсе.

Ассоциация между «Место» и «Транзакция» необходима для отображения места совершения транзакции, при этом указание места

совершения транзакции не обязательно. Если место указывается, то только одно.

Ассоциация между «Категория» и «Транзакция» необходимо для отображения категории совершённой транзакции. Это обязательная связь для транзакции и для одной транзакции указывается только одна категория.

Ассоциация «с какого счета» между «Транзакция» и «Счёт» необходима для связи со счётом, с которого была списана сумма транзакции.

Ассоциация «на какой счет» между «Транзакция» и «Счёт» необходима для связи со счетом, на который была записана сумма транзакции.

У объекта транзакции может быть или только отношение «с какого счета», или только отношение «на какой счет», или оба отношения сразу.

В первом случае принимается, что транзакция является расходом, во втором – доходом, в третьем – переводом.

Ассоциация между «Транзакция» и «Пользователь» необходима для привязки транзакций к конкретному пользователю.

Полученная аналитическая модель хорошо описывает выбранную предметную область, при этом оставаясь простой и интуитивно понятной. Имена выявленных классов отражают их назначение и точно проецируются в аспекты предметной области.

### 3 ПРОЕКТИРОВАНИЕ

Анализ предметной области завершён, создана аналитическая модель. Следующий этап в разработке – проектирование. Цель проектирования – определить в полном объеме, как будет реализовываться функциональность системы, удовлетворяющая требованиям пользователя [7].

#### 3.1 Проектные классы

Посредством уточнения классов анализа реализуются проектные классы, это включает в себя добавление деталей реализации. При проектировании моделируется то, как реализовать функции, которые должна выполнять система. На рисунке 3 представлена диаграмма проектных классов, реализуемых в системе.

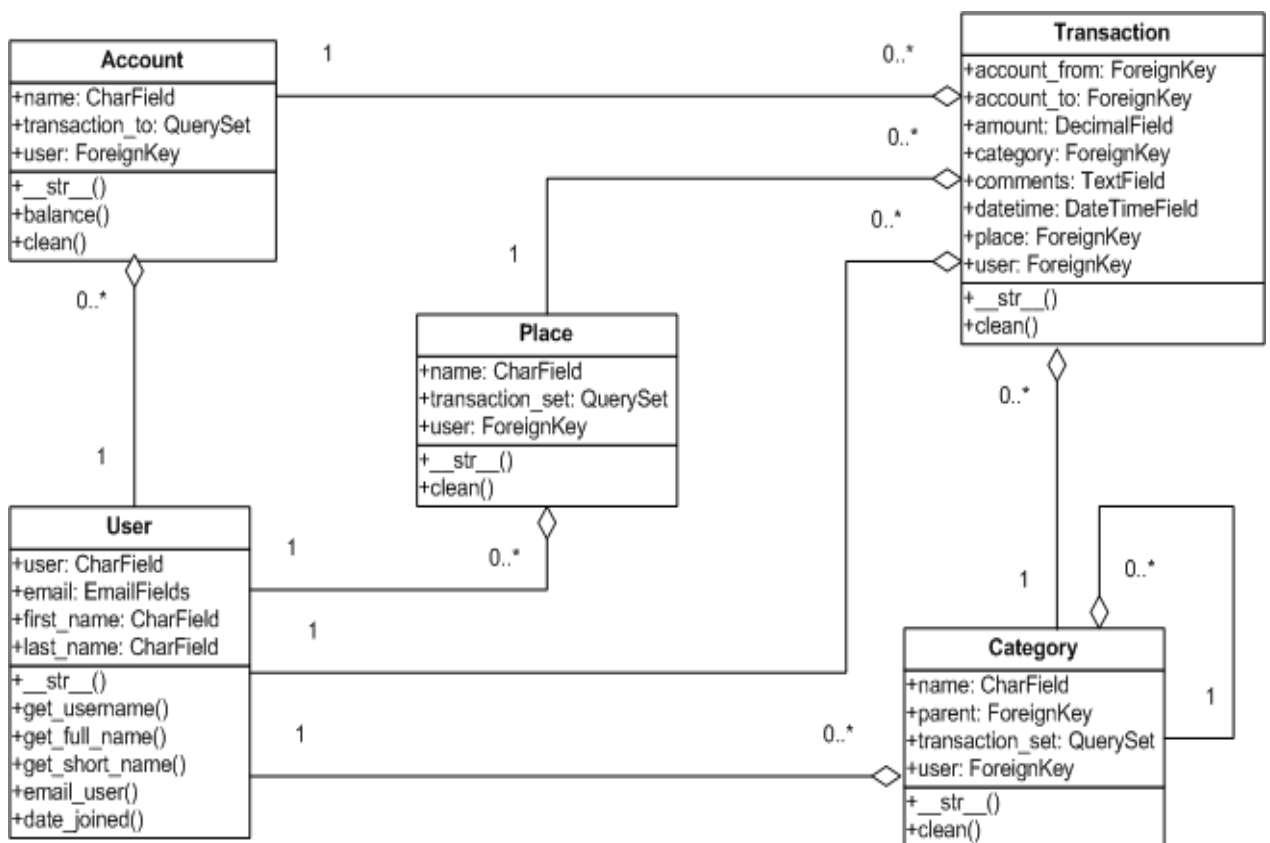


Рисунок 3 – Диаграмма проектных классов

Диаграмма классов с этапа анализа не претерпела сильных изменений. Описание классов анализа совпадает с описанием проектных классов (раздел

2), но стоит отметить, что в ходе реализации было решено использовать один класс, встроенный в Django – User.

Стоит выделить 3 метода, используемых в данной диаграмме, два из которых – `clean()` и `__str__()` – переопределённые методы базового класса, а метод `balance()` является новым:

- 1) `__str__()` – данный метод определяет поведение функции `str()`, вызванной для экземпляра класса. В свою очередь, функция преобразует данные в строку, то есть из любого типа данных можно получить строковое представление.

Код метода:

```
def __str__(self):  
    return self.name
```

- 2) `balance()` – метод используется для подсчёта текущего количества средств на счёте, учитывая все осуществлённые транзакции.

Код метода:

```
def balance(self):  
    value = Decimal(0.00)  
    for transaction in self.transactions_from.all():  
        value -= transaction.amount  
    for transaction in self.transactions_to.all():  
        value += transaction.amount  
    return value
```

- 3) `clean()` – метод используется для проверки всего объекта, любое исключение `ValidationError` вызванное в `clean()` будет сохранено со специальным ключом в словаре ошибок, `NON_FIELD_ERRORS`, который используется для ошибок относящихся ко всей модели, а не конкретному полю.

Код метода:

```
def clean(self):  
    cleaned_data = super().clean()  
    if self.user.places.filter(name=self.name).exists():  
        raise ValidationError(_('Название места должно быть уникальным'))  
    return cleaned_data
```

### 3.2 Диаграмма пакетов системы

Пакет основной способ организации элементов модели в языке UML. Каждый пакет владеет всеми своими элементами, т. е. теми элементами, которые включены в него. Про соответствующие элементы пакета говорят, что они принадлежат пакету или входят в него. При этом каждый элемент может принадлежать только одному пакету.

Тем самым для элементов модели задается отношение вложенности пакетов, которое представляет собой иерархию. Диаграммы пакетов отображают зависимости между пакетами, составляющими модель [8].

На рисунке 4 представлена диаграмма пакетов. Стоит отметить, что назначения пакетов соответствует их названию.

Пакет UI содержит в себе визуальное представление системы. Отвечает за взаимодействие пользователя с системой. Содержит в себе пакет Templates с различными шаблонами страниц системы, а также пакет Static со статическими файлами (изображения, каскадные таблицы стилей и др). Шаблоны используют статические файлы для формирования графического представления.

В свою очередь, одни пакеты могут быть вложены в другие пакеты.

Пакет Models реализует основную логику системы. Отвечает за работу с категориями, местами, счетами и транзакциями пользователя, а также за их взаимодействие между собой.

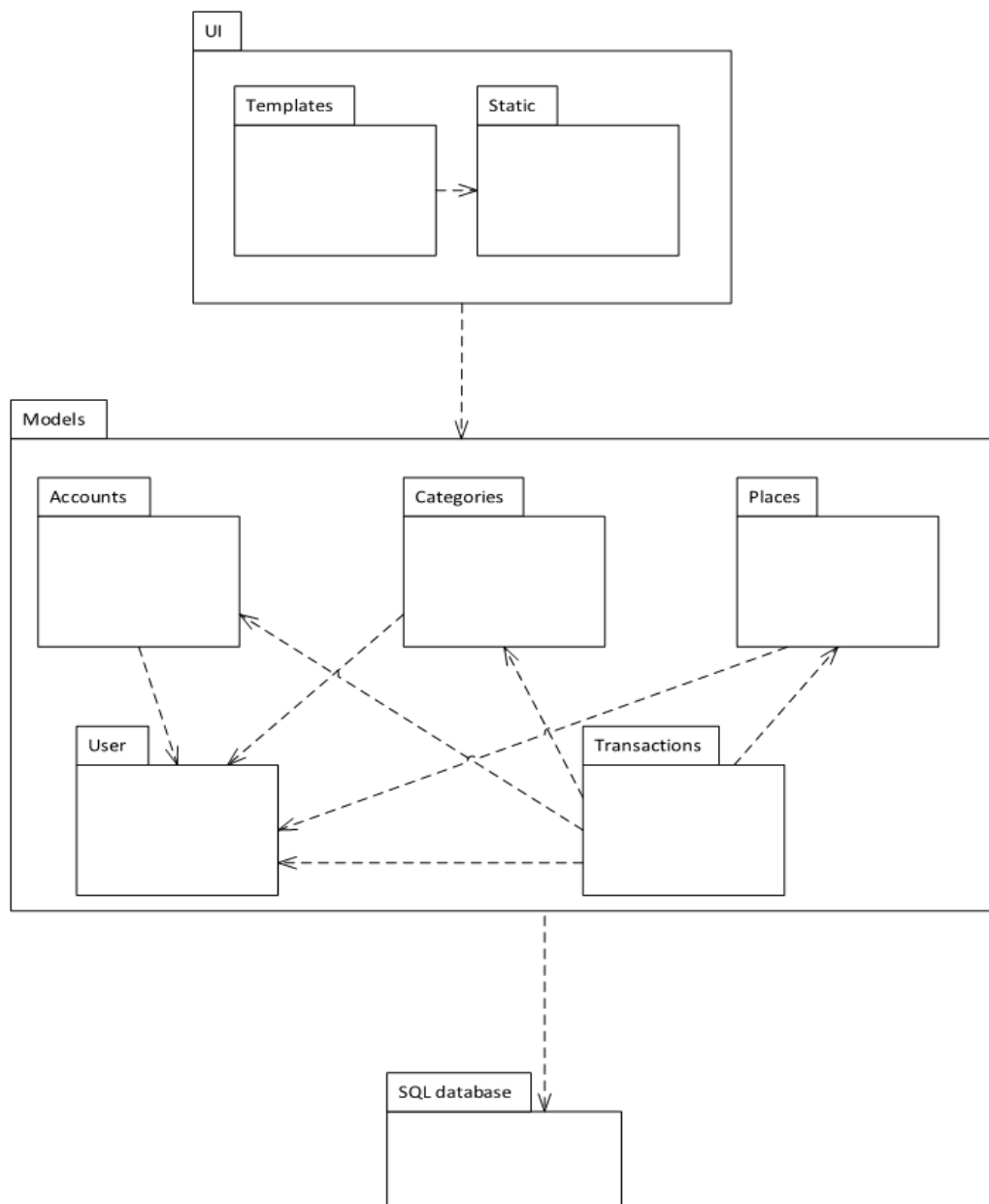


Рисунок 4 – Диаграмма пакетов

Пакет SQL database является базой данных, используемой для хранения информации, необходимой для работы системы.

### 3.3 Диаграммы последовательностей для операций проектных классов

Диаграмма последовательности – это диаграмма, описывающая один сценарий приложения. На диаграмме изображаются экземпляры объектов и сообщения, которыми они обмениваются в рамках одного прецедента (use case).

Основными элементами диаграммы последовательности являются обозначения объектов, вертикальные линии жизни, отображающие течение

времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции, и стрелки, показывающие обмен сигналами или сообщениями между объектами. В ходе выполнения курсового проекта были спроектированы диаграммы последовательностей для нескольких функций. На рисунке 5 изображена диаграмма последовательности для функции `full_clean` в классе `BaseForm`.

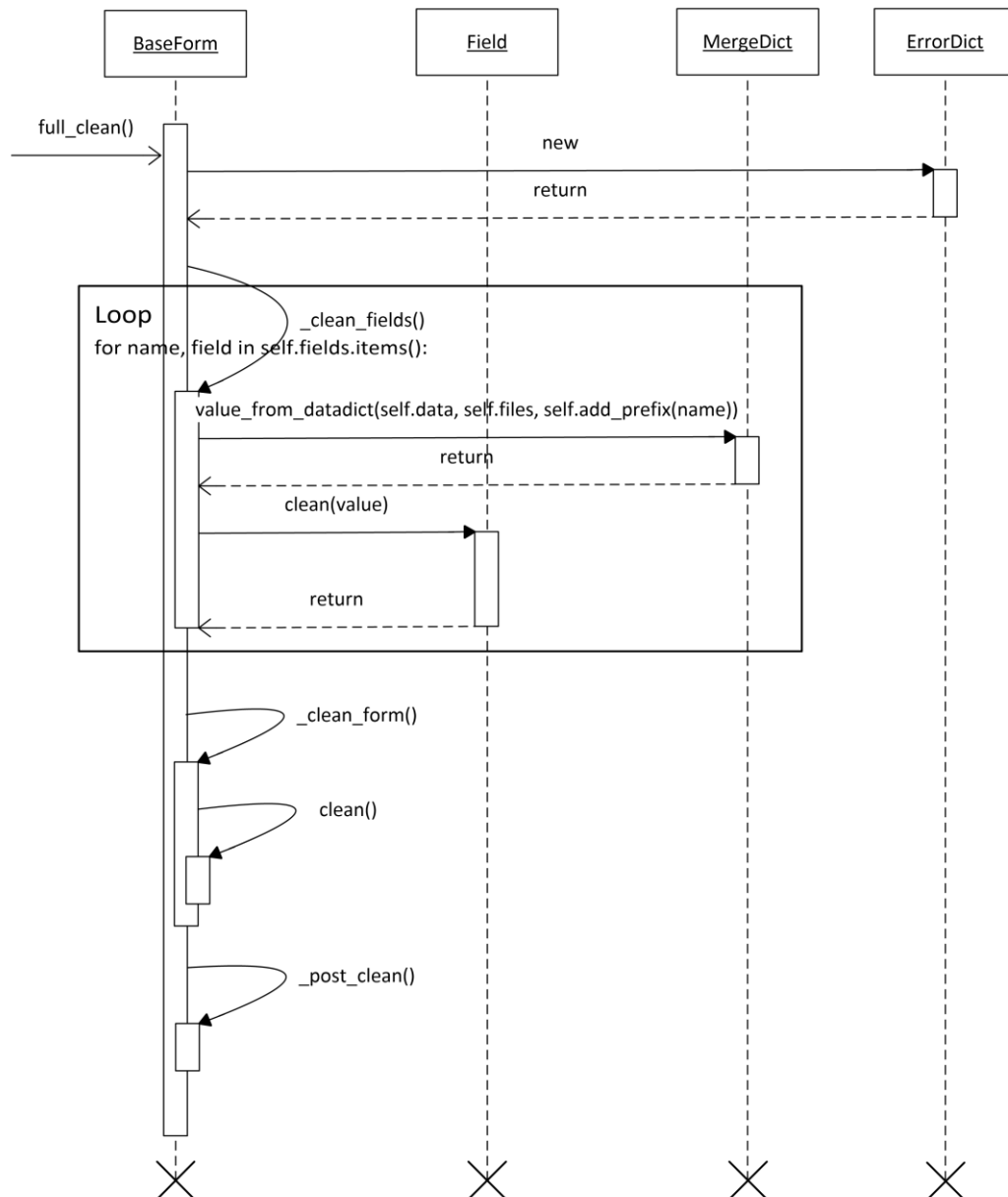


Рисунок 5 – Диаграмма последовательности `full_clean` в `BaseForm`

Описание диаграммы последовательностей для метода `full_clean()` класса `BaseForm`.



На диаграмме, для того, чтобы не вызвать затруднения при понимании выбран один из вариантов работы методов.

При вызове `full_clean()` происходит создание нового `ErrorDict`, который хранит ошибки, возникающие при работе метода. После этого происходит несколько разветвлений, не указанных на диаграмме, которые могут привести к концу функции. Затем происходит последовательный вызов трёх функций `clean_fields()`, `clean_form()` и `post_clean()`. Первая функция содержит цикл, в котором происходит выполнение `clean()` для каждого `field` по отдельности. В функции `clean_form()` происходит вызов `clean()` ещё раз, но уже для `self`. Последняя функция `_post_clean()` служит для возможной дополнительной проверки и в самом классе не обладает функционалом.

Код метода `full_clean`, используемого в `BaseForm` представлен ниже:

```
def full_clean(self):
    """
    Cleans all of self.data and populates self._errors and
    self.cleaned_data.
    """
    self._errors = ErrorDict()
    if not self.is_bound: # Stop further processing.
        return
    self.cleaned_data = {}
    # If the form is permitted to be empty, and none of the
    form data has
    # changed from the initial data, short circuit any
    validation.
    if self.empty_permitted and not self.has_changed():
        return

    self._clean_fields()
    self._clean_form()
    self._post_clean()

    def _clean_fields(self):
        for name, field in self.fields.items():
            # value_from_datadict() gets the data from the data
            dictionaries.
            # Each widget type knows how to retrieve its own data,
            because some
            # widgets split data over several HTML fields.
            value = field.widget.value_from_datadict(self.data,
            self.files, self.add_prefix(name))
```

```

        try:
            if isinstance(field, FileField):
                initial = self.initial.get(name, field.initial)
value = field.clean(value, initial)
            else:
                value = field.clean(value)
                self.cleaned_data[name] = value
                if hasattr(self, 'clean_%s' % name):
                    value = getattr(self, 'clean_%s' % name)()
                    self.cleaned_data[name] = value
        except ValidationError as e:
            self.add_error(name, e)

def _clean_form(self):
    try:
        cleaned_data = self.clean()
    except ValidationError as e:
        self.add_error(None, e)
    else:
        if cleaned_data is not None:
            self.cleaned_data = cleaned_data

def _post_clean(self):
    """
    An internal hook for performing additional cleaning
    after form cleaning
    is complete. Used for model validation in model forms.
    """
    pass

```

Диаграмма последовательности для метода `full_clean()` в классе `Model` представлена на рисунке 6.

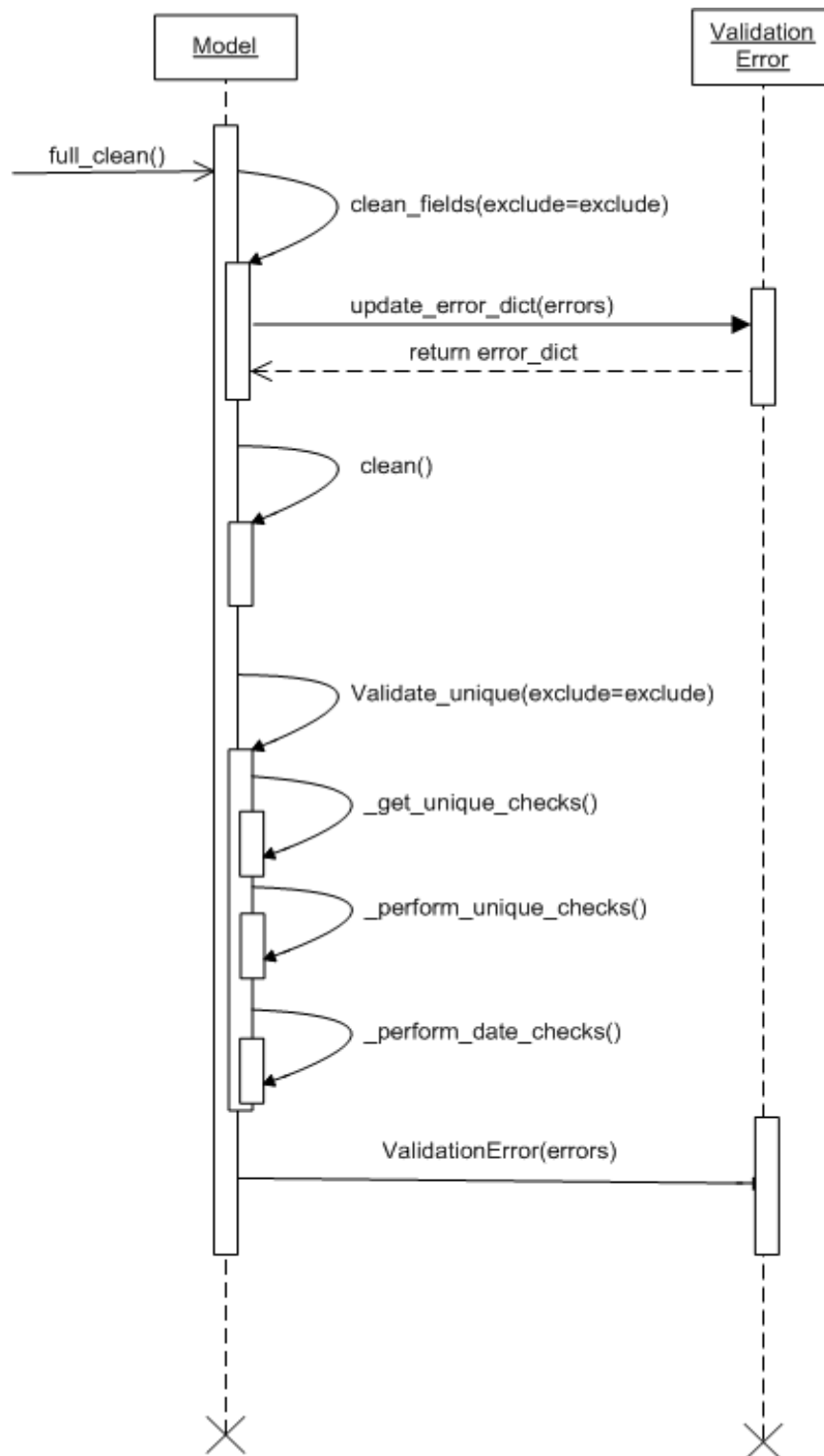


Рисунок 6 – Диаграмма последовательности `full_clean` в `Model`

Описание диаграммы последовательностей для метода `full_clean()` класса `Model`.

На диаграмме, для того, чтобы не вызвать затруднения при понимании выбран один из вариантов работы методов.

При вызове `full_clean()` происходит вызов функции `clean_fields()` с параметром `exclude=exclude`, после этого происходит вызов `clean()`. Затем происходит вызов функции `validate_unique`, также с параметрами `exclude=exclude`, которая в свою очередь вызывает `get_unique_checks` с параметром `exclude=exclude`, `perform_unique_checks` с параметром `unique_checks`, после функцию `_perform_date_checks` с параметром `date_checks`. После этого, при наличии ошибок происходит вызов `ValidationError` с параметром `errors`.

Код метода:

```
def full_clean(self, exclude=None, validate_unique=True):
    """
    Calls clean_fields, clean, and validate_unique, on the
    model,
    and raises a ``ValidationError`` for any errors that
    occurred.
    """
    errors = {}
    if exclude is None:
        exclude = []
    else:
        exclude = list(exclude)

    try:
        self.clean_fields(exclude=exclude)
    except ValidationError as e:
        errors = e.update_error_dict(errors)

    # Form.clean() is run even if other validation fails, so
    do the
    # same with Model.clean() for consistency.
    try:
        self.clean()
    except ValidationError as e:
        errors = e.update_error_dict(errors)

def validate_unique(self, exclude=None):
    """
    Checks unique constraints on the model and raises
    ``ValidationError``
    if any failed.
    """
    unique_checks, date_checks =
self._get_unique_checks(exclude=exclude)

    errors = self._perform_unique_checks(unique_checks)
```

```

        date_errors = self._perform_date_checks(date_checks)

        for k, v in date_errors.items():
            errors.setdefault(k, []).extend(envy)

        if errors:
            raise ValidationError(errors)
        # Run unique checks, but only for fields that passed
        validation.
        if validate_unique:
            for name in errors.keys():
                if name != NON_FIELD_ERRORS and name not in exclude:
                    exclude.append(name)
            try:
                self.validate_unique(exclude=exclude)
            except ValidationError as e:
                errors = e.update_error_dict(errors)

        if errors:
            raise ValidationError(errors)

```

На рисунке 7 представлена диаграмма последовательности вызова конструктора класса `BaseModelForm`.

В конструкторе есть различные ветвления, в данной диаграмме был рассмотрен конкретный вариант исполнения:

```

def __init__(self, data=None, files=None, auto_id='id_%s',
             prefix=None, initial=None, error_class=ErrorList,
             label_suffix=None, empty_permitted=False, instance=None):
    opts = self._meta
    self.instance = instance
    object_data = model_to_dict(instance, opts.fields,
                                opts.exclude)
    object_data.update(initial)
    super(BaseModelForm, self).__init__(data, files,
                                         auto_id, prefix, object_data, error_class, label_suffix,
                                         empty_permitted)
    for field_name in self.fields:
        formfield = self.fields[field_name]
        limit_choices_to = formfield.limit_choices_to
        formfield.queryset =
        formfield.queryset.complex_filter(limit_choices_to)

```

После вызова конструктора класса `BaseModelForm` происходит вызов функции `model_to_dict()` с аргументами `instance`, `opts.fields` и `opts.exclude`, которая возвращает объект словаря, на котором далее

вызывается метод `update()` с аргументом `initial`, после чего вызывается конструктор родительского класса `BaseForm`. В конечном итоге выполняется цикл по `self.fields`, вызывающий метод `complex_filter()` с аргументом `limit_choices_to` класса `QuerySet`.

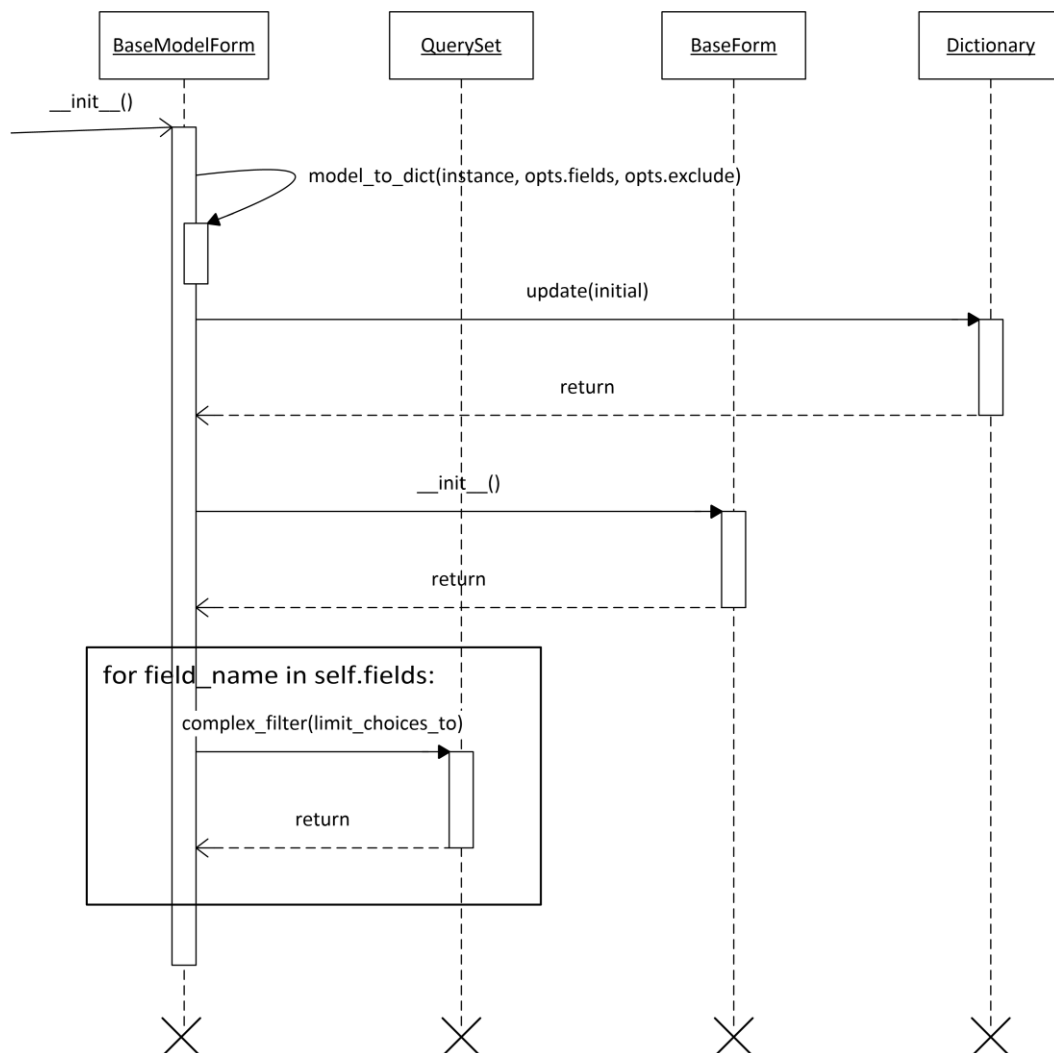


Рисунок 7 – Диаграмма последовательности вызова конструктора класса `BaseModelForm`

Диаграмма последовательности для метода `balance()` в классе `Account` показана на рисунке 8, код метода приведён в разделе 3.1.

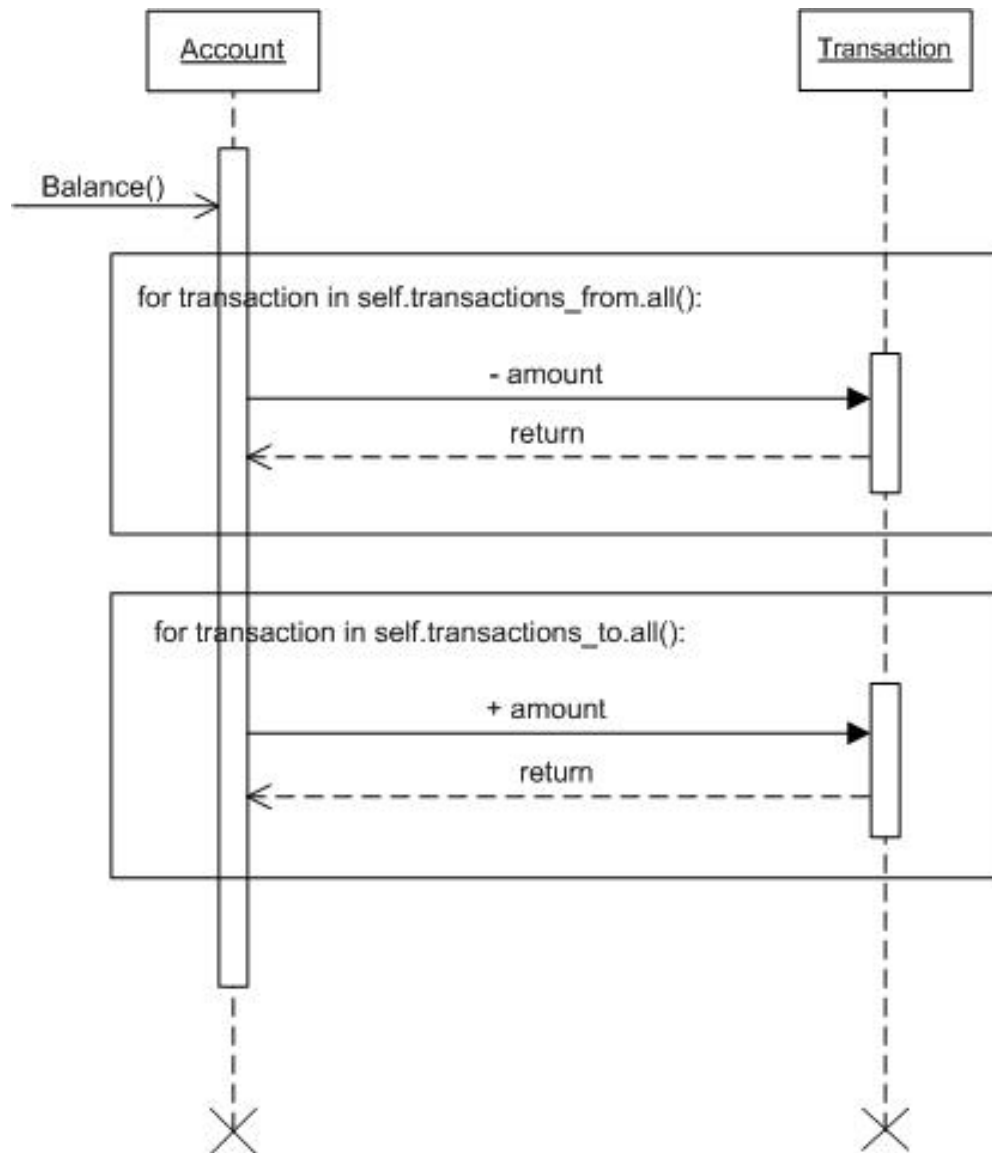


Рисунок 8 – Диаграмма последовательности для метода `balance` в `Account`

Описание диаграммы последовательностей для метода `balance()` класса `Account`.

При вызове метода `balance()` осуществляется подсчёт количества средств на счёте. В зависимости от вида осуществлённой транзакции, то есть перевода средств на счёт `transaction in self.transactions_to.all` или со счёта `transaction in self.transactions_from.all`, происходит изменение баланса в положительную сторону `+amount`, и, соответственно, вычитание количества средств `-amount`.

На рисунке 9 изображена диаграмма последовательности для метода `clean` класса `Category.Model`.

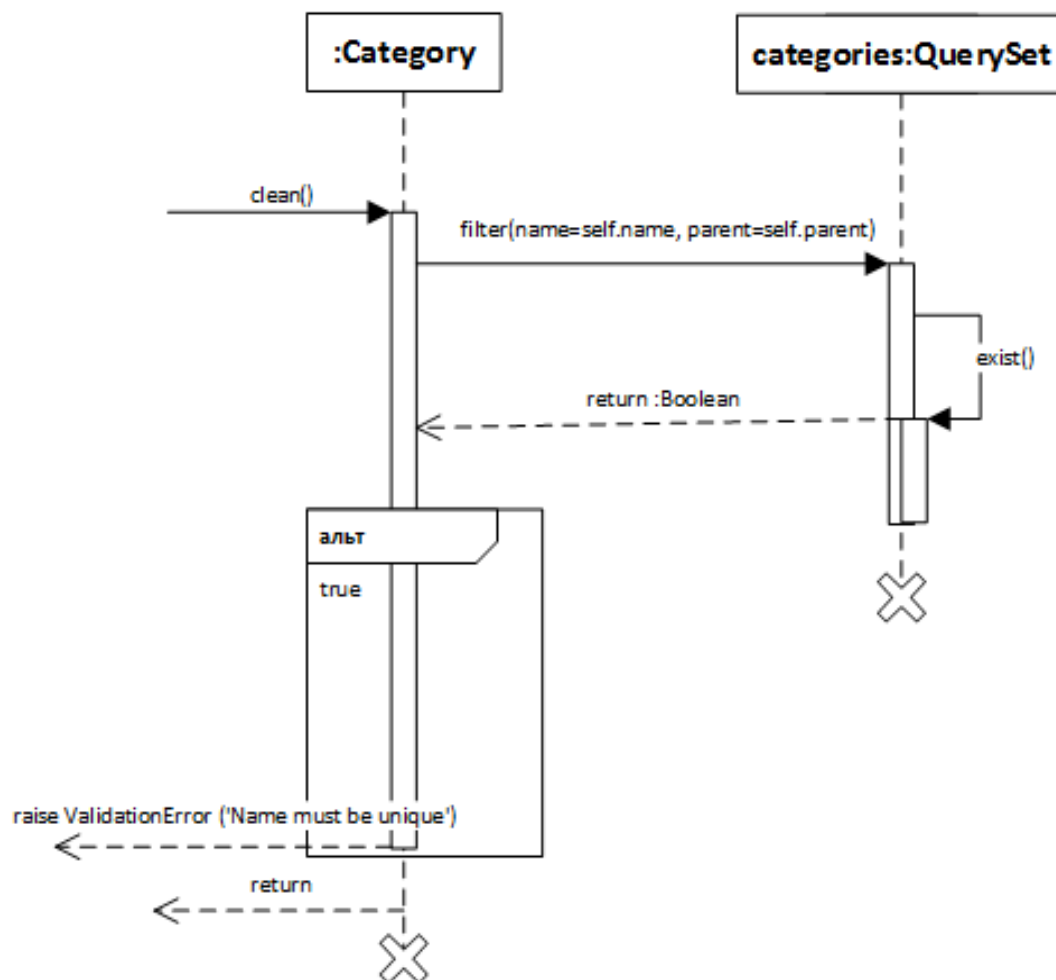


Рисунок 9 – Диаграмма последовательности для метода `clean()` в `Category.Model`

Описание диаграммы последовательностей для метода `clean()` класса `Category.Model`.

При вызове метода `clean()` происходит получение набора категорий по заданным параметрам (метод `filter()`). Далее следует проверка, существует ли в полученном наборе категория с таким же именем, как и текущая, если существует, то возникает исключение `ValidationError`. Если нет, то происходит возврат в вызывающую часть программы.



## 4 РЕАЛИЗАЦИЯ

В данном курсовом проекте кроме реализации самого проекта выполнена реализация ряда дополнительных, вспомогательных процессов, таких как тестирование и непрерывная интеграция.

### 4.1 Тестирование

Тестирование в проекте представлено двумя видами, описанными далее.

#### 4.1.1 Модульное тестирование

Модульное тестирование проводилось для ряда приложений, представленных в проекте, но так как не все из этих приложений имеют функции, пригодные для проверки, то проверка осуществлялась не только для уникальных функций, но и для встроенных.

Для написания тестов класс, осуществляющий проверку, должен наследоваться от `TestCase`. `TestCase` – это тестовый сценарий (набор проверяемых условий, переменных, состояний системы или режимов). Обычно является логически неделимым и может содержать одну или более проверок (`asserts`) [9]. Также названия всех методов классов-тестов должны начинаться с `test_`.

Далее представлен исходный код методов и описано, какую проверку они осуществляют.

##### 4.1.1.1 Тесты *Place*

Для приложения *Place* представлены простейшие проверки модели – недопустимость создания сущности без обязательных полей. Эти проверки разбиты на два логических модуля – один проверяет недопустимость создания сущности вообще без атрибутов, другой без атрибута пользователя.

Исходный код этих тестов представлен далее.

```
class PlaceTest(TestCase):
    def test_null_place(self):
        with self.assertRaises(IntegrityError):
```

```

        p = Place()
        p.save()

    def test_place_without_user(self):
        with self.assertRaises(IntegrityError):
            p = Place.objects.create(
                name='test',
            )

```

#### 4.1.1.2 Тесты Category

Для приложения Category представлено три теста, один из которых проверяет, что атрибут name созданной сущности возвращается как название сущности. Второй и третий тесты проверяют валидацию модели, то есть недопустимость создания категорий с одинаковыми именами и одинаковыми родительскими полями (один из тестов корневой элемент, второй с явно назначенным родителем).

Исходный код этих тестов представлен далее.

```

class CategoryTest(TestCase):
    def test_trycategoryname(self):
        user = get_user_model().objects.create(
            username='test',
        )
        cat = Category(name='TestCategory', parent=None,
            user=user)
        self.assertEqual(cat.__str__(), 'TestCategory')

    def test_cleancategory(self):
        user = get_user_model().objects.create(
            username='test',
        )
        Category.objects.create(name='test', user=user)
        cat2 = Category(name='test', user=user)
        self.assertRaisesMessage(ValidationError, 'Name must be
            unique', cat2.clean)

    def test_trycategorynamethesameparent(self):
        user = get_user_model().objects.create(
            username='test',
        )
        parent = Category.objects.create(name='parent',
            user=user, parent=None)
        Category.objects.create(name='child', user=user,
            parent=parent)
        child = Category(name='child', user=user,
            parent=parent)

```

```
self.assertRaisesMessage(ValidationError, 'Name must  
be unique', child.clean)
```

#### 4.1.1.3 Тесты Transaction

Для приложения Transaction представлен тест, проверяющий валидацию модели, которая запрещает создавать транзакцию с одинаковыми значениями полей `account_from` и `account_to`, то есть «счёт откуда» и «счёт куда».

Исходный код этого теста представлен далее.

```
class TransactionTest(TestCase):  
    def test_double_amount(self):  
        user = get_user_model().objects.create(  
            username='test',  
        )  
        # Transaction.objects.create(account_from='test',  
account_to='test', user=user)  
        acc = Account(name='qwe', user=user)  
        acc.save()  
        trans2 = Transaction(account_from=acc, account_to=acc,  
user=user)  
        self.assertRaisesMessage(ValidationError, 'Счета должны  
быть разные', trans2.clean)
```

#### 4.1.1.4 Тесты Account

Для приложения Account тест представляет проверку собственного метода `balance`, который подсчитывает текущий баланс для счёта.

Исходный код этого теста представлен далее.

```
class AccountTest(TestCase):  
    def test_balance(self):  
        amount = Decimal(1.23)  
        user = get_user_model().objects.create(  
            username='test',  
        )  
        account = Account.objects.create(  
            name='test',  
            user=user,  
        )  
        Transaction.objects.create(  
            account_to=account,  
            amount=amount,  
        )  
        self.assertAlmostEqual(account.balance(), amount, 2)
```

Все описанные тесты проходят проверку, что подтверждает правильность как написания тестов, так и ожидаемую реакцию функций на производимые в тестах проверки.

Правильность подтверждается прохождением тестов в среде PyCharm. Результат запуска тестов представлен на рисунке 10.

Test ▲	Time elapsed	Results
accounts.tests.TransactionTest	4 ms	P:1
categories.tests.CategoryTest	5 ms	P:2
places.tests.PlaceTest	1 ms	P:2
transactions.tests.TransactionTest	1 ms	P:1

Рисунок 10 – Результат тестов

## 4.2 Покрытие кода

Покрытие кода показывает, насколько исходный код программы был протестирован. Получить этот показатель можно несколькими способами, но в рамках курсового проекта был использован способ, предоставляемый PyCharm. Для этого необходимо создать конфигурацию, запускающую все тесты и выбрать вариант Run with Coverage (рисунок 11).



Рисунок 11 – Запуск с покрытием

Результат запуска представлен на рисунке 12.

Coverage Summary: 59% files, 92% lines covered	
Element	Statistics, %
.idea	
accounts	62% files, 87% lines covered
api	0% files, not covered
categories	62% files, 98% lines covered
moneytalks	20% files, 100% lines covered
places	62% files, 87% lines covered
static	
templates	
transactions	75% files, 97% lines covered
Документы	
.gitignore	
.travis.yml	
db.sqlite3	
manage.py	42% lines covered
README.md	
requirements.txt	
tox.ini	

Рисунок 12 – Результат покрытия кода

Итоговое покрытие получилось 59% файлов и 92% строк. Из статистики выбивается приложение `api`, которое создано для возможного дальнейшего развития и не используется на текущий момент.

### 4.3 Непрерывная интеграция

Для проекта также настроена непрерывная интеграция. Непрерывная интеграция (англ. Continuous Integration) – это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий [10].

Для данного курсового проекта используется Travis CI. Travis CI – распределённый веб-сервис для сборки и тестирования программного обеспечения, использующего GitHub в качестве хостинга исходного кода.

Программная составляющая сервиса также располагается на GitHub'е, однако разработчики не рекомендуют использовать её в закрытых проектах [11].

Процесс настройки довольно прост, и представляет собой последовательность четырёх шагов, которые описаны далее.

Для использования сервиса необходимо зайти через свою учётную запись GitHub и дать разрешение на доступ к ряду параметров учётной записи, основным разрешением является доступ к webhook [12]. Для входа достаточно быть авторизованным на хостинге GitHub и нажать кнопку Sign in with GitHub, представленную на рисунке 13.

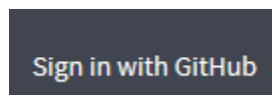


Рисунок 13 – Вход в Travis CI

После этого необходимо перейти в свою учётную запись и активировать построение репозитория нажатием переключателя. Результат этого процесса представлен на рисунке 14.

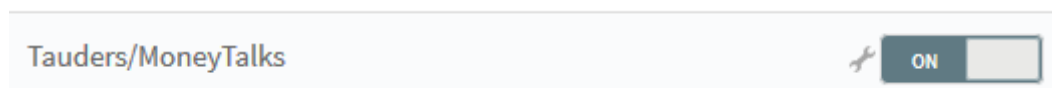


Рисунок 14 – Включение построения репозитория

Далее следует создать файл `.travis.yml`, который содержит описание репозитория – на каком языке написан исходный код, на какой версии, и дополнительные параметры, такие как список необходимых пакетов для построения проекта. Файл для данного курсового проекта представлен далее.

```
language: python
python:
  - "3.2"
  - "3.3"
# command to install dependencies
install: "pip install -r requirements.txt"
script:
  - "flake8 ."
  - "python manage.py test"
```



## **5 ДОКУМЕНТАЦИЯ**

### **5.1 Назначение программы**

Разработанный проект «MoneyTalks» предназначена для комплексного информационно-аналитического обеспечения контроля за личными финансами пользователей. При авторизации каждый пользователь получает возможность запомнить, где, сколько и при каких обстоятельствах он потратил или заработал деньги.

В частности данный сайт позволяет:

- зарегистрироваться и авторизоваться для получения доступа к другим функциям сайта;
- добавление, редактирование и удаление счетов, мест, категорий и транзакций;
- отображение счетов, мест, категорий и транзакций в соответствующих разделах;
- при отображении транзакций существует возможность фильтрации на переводы со счёта на счёт и отображения только доходов или расходов.

Сайт «MoneyTalks» рассчитан на людей, имеющих постоянный доступ в интернет и не так много свободного времени, но которые хотят следить за своими расходами.

### **5.2 Условия выполнения программы**

Для работы на сайте пользователю необходимо иметь персональный компьютер, смартфон или любое другое техническое устройство с возможностью выхода в интернет со скоростью не менее 512 КБ/с. Также необходим установленный браузер и возможность ввода.

В ходе самой эксплуатации есть ряд ограничений, такие как:

- нельзя вводить отрицательную сумму;



- при добавлении перевода со счёта на счёт необходимо указать два различных счёта и сумму;
- при добавлении расхода, надо указать, на какой счёт будут переведены книги и сумму;
- при добавлении дохода указывается счёт, с которого получены деньги и сумму;
- при добавлении места, категории и счёта нужно указать название места, категории и счёта соответственно;
- нельзя создавать места и счёт с одинаковыми именами у одного пользователя;
- нельзя создавать категории и подкатегории с одинаковыми именами у одного пользователя, но подкатегории могут быть с одинаковыми именами, если у них разные родители.

### 5.3 Выполнение программы

Сервис выполняет следующие функции:

#### 5.3.1 Регистрация уникального пользователя

Для прохождения регистрации необходимо пройти по кнопке «Регистрация» в правом верхнем углу страницы (Рисунок 16).

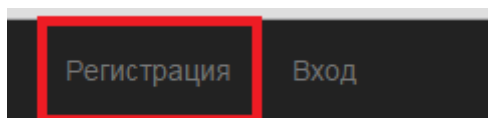


Рисунок 16 – Кнопка «Регистрация»

Затем открывается новая страница с полями: Имя пользователя, E-mail, Пароль и Пароль (еще раз). Эта страница показана на рисунке 17.

Имя пользователя\* Tervin

E-mail\* Tervin@scalpnet.ru

Пароль\* .....

Пароль (еще раз)\* .....

Зарегистрироваться

Рисунок 17 – Страница регистрации

Каждое поле является обязательным. После заполнения каждого поля, Пользователь нажимает кнопку «Зарегистрироваться», после чего пользователь может авторизоваться.

### 5.3.2 Авторизация пользователя

Для прохождения авторизации необходимо пройти по кнопке «Вход» в правом верхнем углу страницы (рисунок 18).

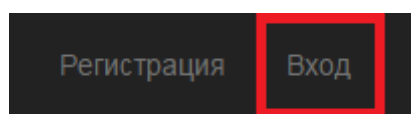


Рисунок 18 – Кнопка «Вход»

Затем открывается новая страница с полями: Имя пользователя и Пароль. Эта страница показана на рисунке 19.

Имя пользователя\* Tervin

Пароль\* ...

Войти

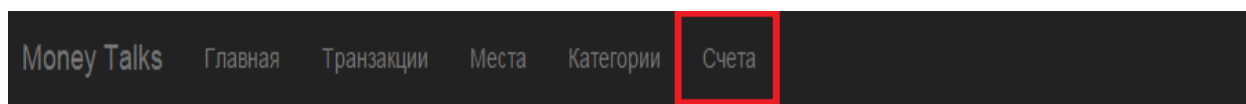
Не зарегистрированы?  
[Регистрация.](#)

Рисунок 19 – Страница авторизации

Каждое поле является обязательным. После заполнения каждого поля, Пользователь нажимает кнопку «Войти», после чего пользователь получает доступ к другим функциям сайта.

### 5.3.3 Создание нового уникального счета

Для создания нового счёта необходимо пройти по кнопке «Счета» в центреверху страницы (рисунок 20).



- 1. [qwe Удалить](#)
- 2. [asd Удалить](#)
- [Создать](#)

Рисунок 20 – Кнопка «Счета»

Затем открывается новая страница с несколькими кнопками. Если счета до этого не были созданы, будет отображаться только кнопка «Создать». При наличии счетов появляется список с возможностью изменять и удалять счета с помощью соответствующих кнопок. При нажатии кнопки «Создать» появляется новая страница, показанная на рисунке 21.

A form for creating a new account. It consists of a label 'Name:' followed by a white rectangular input field. Below the input field is a grey button with the text 'Далее' in white.

Рисунок 21 – Создание нового счёта

Здесь одно обязательное для заполнения поле: название счёта. При нажатии кнопки «Создать» счёт проходит проверку на уникальность, и при успешном прохождении проверки создаётся новый счёт и открывается страница со списком счетов.

### 5.3.4 Создание нового уникального места

Для создания нового места необходимо пройти по кнопке «Места» в центреверху страницы (рисунок 22).

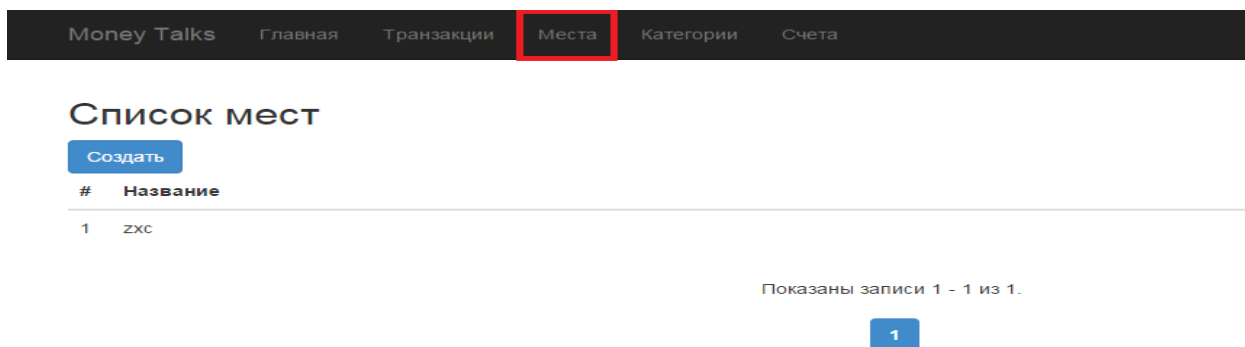


Рисунок 22 – Кнопка «Места»

Затем открывается новая страница с несколькими кнопками. Если места до этого не были созданы, будет отображаться только кнопка «Создать». При наличии мест появляется список с возможностью изменять и удалять места с помощью соответствующих кнопок. При нажатии кнопки «Создать» появляется новая страница, показанная на рисунке 23.

### Редактирование места

Название места\*

Рисунок 23 – Создание нового места

Здесь одно обязательное для заполнения поле: название места. При нажатии кнопки «Создать» место проходит проверку на уникальность, и при успешном прохождении проверки создаётся новое место и открывается страница со списком мест.

### 5.3.5 Создание новой уникальной категории

Для создания категории необходимо пройти по кнопке «Категории» в центреверху страницы (рисунок 24).

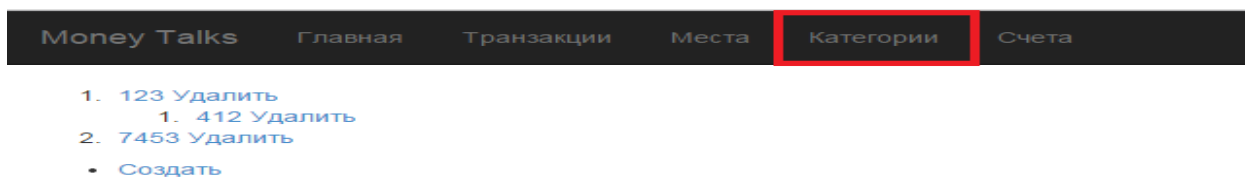


Рисунок 24 – Кнопка «Категории»

Затем открывается новая страница с несколькими кнопками. Если категории до этого не были созданы, будет отображаться только кнопка «Создать». При наличии категорий появляется список с возможностью изменять и удалять категории с помощью соответствующих кнопок. При нажатии кнопки «Создать» появляется новая страница, показанная на рисунке 25.



Рисунок 25 – Создание новой категории

Здесь одно обязательное для заполнения поле: название категории. При нажатии кнопки «Создать» категория проходит проверку на уникальность, и, при успешном прохождении проверки, создаётся новая категория и открывается страница со списком категорий.

### 5.3.6 Создание подкатегории, уникальной в рамках родительской категории

Для создания подкатегории необходимо пройти по кнопке «Категории» в центре вверху страницы (Рисунок 26).

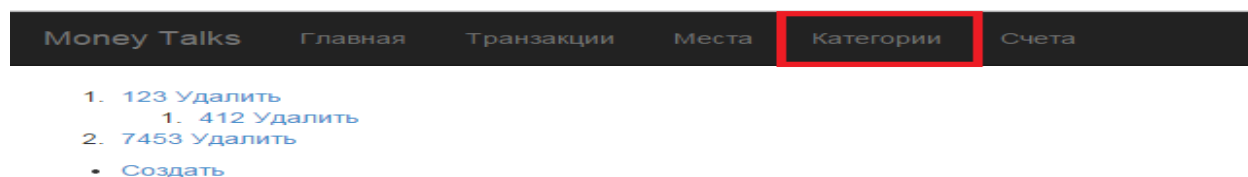


Рисунок 26 – кнопка «Категории»

Затем открывается новая страница с несколькими кнопками. Если категории до этого не были созданы, будет отображаться только кнопка «Создать». При наличии категорий появляется список с возможностью изменять и удалять категории с помощью соответствующих кнопок. При нажатии кнопки «Создать» появляется новая страница, показанная на рисунке 27.

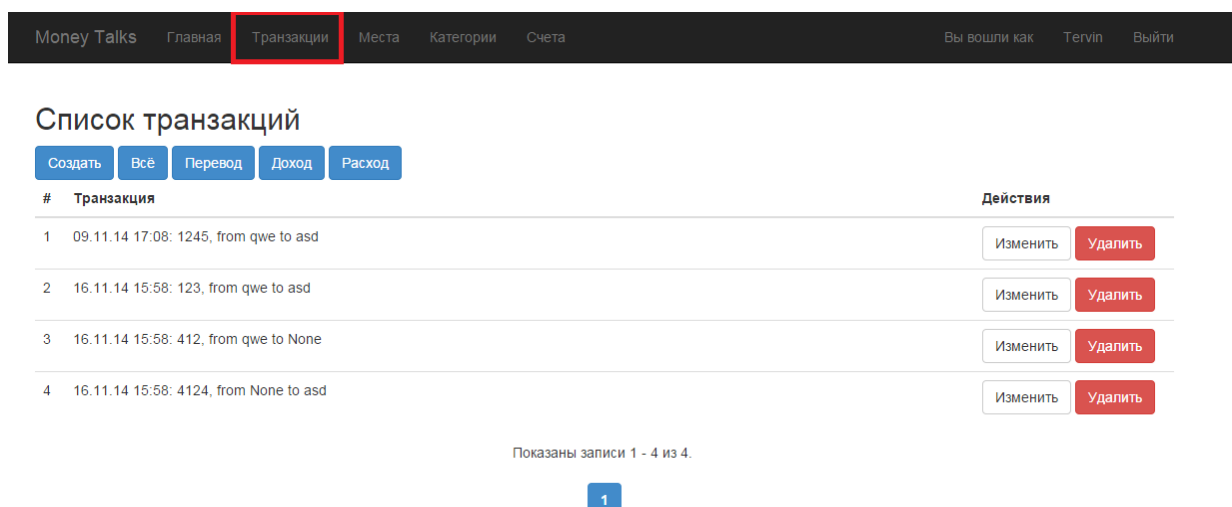


Рисунок 27 – Создание новой категории

Для создания подкатегории необходимо иметь уже созданную категорию. Пользователь должен выбрать категорию, для указания к чему принадлежит подкатегория. При нажатии кнопки «Создать» подкатегория проходит проверку на уникальность, и, при успешном прохождении проверки, создаётся новая подкатегория и открывается страница со списком категорий.

### 5.3.7 Создание записи о списании средств со счета с указанием места, категории, суммы и комментария

Для создания нового счёта необходимо пройти по кнопке «Транзакции» в центре вверху страницы (рисунок 28).



#	Транзакция	Действия
1	09.11.14 17:08: 1245, from qwe to asd	Изменить Удалить
2	16.11.14 15:58: 123, from qwe to asd	Изменить Удалить
3	16.11.14 15:58: 412, from qwe to None	Изменить Удалить
4	16.11.14 15:58: 4124, from None to asd	Изменить Удалить

Рисунок 28 – кнопка «Счета»

Затем открывается новая страница с несколькими кнопками. Если транзакции до этого не были созданы, будет отображаться только кнопка «Создать». При наличии транзакций появляется список с возможностью изменять и удалять транзакции с помощью соответствующих кнопок. При нажатии кнопки «Создать» появляется новая страница, показанная на рисунке 29.

## Редактирование транзакций

Со счёта	<input type="text" value="qwe"/>
На счёт	<input type="text" value="asd"/>
Место	<input type="text" value="zxc"/>
Категория	<input type="text" value="412"/>
Сумма*	<input type="text" value="4142"/>
Дата*	<input type="text" value="30.11.2014 14:53:49"/>
Комментарий	<div><div>bxdf</div></div>

Создать

Рисунок 29 – Создание нового счёта

Здесь несколько обязательных для заполнения полей: сумма и дата. Также необходимо указать хотя бы один из счетов. При нажатии кнопки «Создать» счёт проходит проверку на уникальность, и, при успешном прохождении проверки, создаётся новый счёт, и открывается страница с обновлённым списком счетов.

### 5.3.8 Изменение или удаление счетов, мест, категорий и транзакций

Для удаления или изменения счетов, мест, категорий и транзакций необходимо зайти в соответствующие разделы. Напротив каждой строчки с данными находится по две кнопки (рисунок 30).

Действия	
Изменить	Удалить
Изменить	Удалить
Изменить	Удалить
Изменить	Удалить

Рисунок 30 – Действия над записями

При нажатии кнопки «Удалить» появляется диалоговое окно для подтверждения удаления записи. При нажатии же кнопки изменить, появится окно аналогичное созданию с данными сохранёнными ранее. Изменяя поля, сохраняем их с помощью кнопки сохранить.

## 5.4 Сообщения программы

При выполнении различных операций на сайте, могут возникать сообщения обозначающие неверные действия пользователя. Некоторые поля являются обязательными или на них стоят иные ограничения, и, при не заполнении этих полей или вводе некорректных данных, пользователь получает соответствующие уведомление.

Так, например, при редактировании места, если не ввести название, то поле для ввода подсветится красным, а под полем появится надпись, что название места является обязательным для ввода (рисунок 31)

### Редактирование места

Название места\*

Обязательное поле.

Создать

Рисунок 31 – Ошибка при редактировании места

В другом примере ошибка возникает при редактировании транзакций. Если ввести два одинаковых счёта, то сверху страницы появится сообщение о том, что счета должны быть разные (рисунок 32).

### Редактирование транзакций

• Счета должны быть разные

Со счёта

123



На счёт

123



Рисунок 32 – Ошибка при редактировании транзакции



## ЗАКЛЮЧЕНИЕ

Поставленные в рамках курсового проекта цели и задачи выполнены, все начальные требования удовлетворены. Реализован сервис учета личных финансов «MoneyTalks» на языке Python с использованием фреймворка Django в IDE PyCharm.

Основной акцент сделан на проектирование и анализ сервиса: создание схемы базы данных, диаграммы классов, описание логики взаимодействия сервиса и пользователя и компонентов сервиса между собой. Результатом является база данных, удовлетворяющая всем возможным случаям, заданным в требованиях, а также набор компонентов, построенных по схеме MVC. Создание сервиса осуществлялась в несколько этапов, результатом которых были UML диаграммы, используемые и корректируемые в дальнейшем для разработки.

Для организации коллективной работы и поддержания актуальности исходного кода проекта использовалась система управления версиями Git. В качестве системы непрерывной интеграции использовалась система Travis CI.

На данный момент проект реализовал все требования к сервису, но возможны дальнейшие пути развития. Как пример такого развития можно назвать создания API интерфейса и мобильного приложения для более удобной работы с сервисом.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ильин В.И. Общество потребления как форма капиталистического развития [Электронный ресурс] / Материалы III Всероссийского социологического конгресса. – Электрон. текст. дан. – М.: Институт социологии РАН, Российское общество социологов, 2008 – URL: [http://www.isras.ru/abstract\\_bank/1210009740.pdf/](http://www.isras.ru/abstract_bank/1210009740.pdf/), свободный. (дата обращения: 25.11.14)
2. Леденцов Н.П. Общество потребления [Электронный ресурс] / Интернет-журнал при Институте проблем глобализации. – Электрон. текст. дан. 2004. – URL: <http://www.aglob.ru/events/?id=684/>, свободный. (дата обращения: 20.11.14)
3. ezPC. Условия эксплуатации компьютера [Электронный ресурс] / ezPC – Энциклопедия необходимых компьютерных знаний. – Электрон. дан. – URL: <http://ezpc.ru/pcuslov.shtml/>, свободный. (дата обращения: 29.11.14)
4. Системные требования Windows Server 2008 [Электронный ресурс] / TechNet – ресурсы по администрированию, виртуализации, облачным вычислениям. – Электрон. дан. – URL: <http://technet.microsoft.com/ru-ru/windowsserver/bb414778.aspx/>, свободный. (дата обращения: 29.11.14)
5. Системные требования [Электронный ресурс] / Русскоязычная документация по Ubuntu. – Электрон. дан. 2012. – URL: [http://help.ubuntu.ru/wiki/минимальные\\_системные\\_требования/](http://help.ubuntu.ru/wiki/минимальные_системные_требования/), свободный. (дата обращения: 29.11.14)
6. ГОСТ19.201-78. Техническое задание. Требования к содержанию и оформлению [Текст]. Введ. с 01.01.1980. – Москва: Изд-во стандартов, 1988. – 4 с.
7. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 624 с., ил.

8. Диаграмма последовательности [Электронный ресурс] / Википедия – свободная энциклопедия – Электрон. дан. – URL: [https://ru.wikipedia.org/wiki/Диаграмма\\_последовательности/](https://ru.wikipedia.org/wiki/Диаграмма_последовательности/), свободный. (дата обращения: 30.11.14)
9. Quality Assurance in Python - работаем с unittest [Электронный ресурс] / gaHcep's programming blog – Электрон. дан. – URL: <http://gahcep.github.io/blog/2013/02/10/qa-in-python-unittest/>, свободный. (дата обращения: 29.11.14)
10. Непрерывная интеграция [Электронный ресурс] / Википедия – свободная энциклопедия – Электрон. дан. – URL: [https://ru.wikipedia.org/wiki/Непрерывная\\_интеграция/](https://ru.wikipedia.org/wiki/Непрерывная_интеграция/), свободный. (дата обращения: 29.11.14)
11. Travis CI [Электронный ресурс] / Википедия – свободная энциклопедия – Электрон. дан. – URL: [https://ru.wikipedia.org/wiki/Travis\\_CI/](https://ru.wikipedia.org/wiki/Travis_CI/), свободный. (дата обращения: 29.11.2014)
12. Webhook [Электронный ресурс] / Wikipedia, the free encyclopedia – Электрон. дан. – URL: <http://en.wikipedia.org/wiki/Webhook/>, свободный. (дата обращения: 29.11.2014)
13. Django [Электронный ресурс] / Википедия – свободная энциклопедия – Электрон. дан. – URL: <https://ru.wikipedia.org/wiki/Django/>, свободный (дата обращения: 29.11.2014)

## ПРИЛОЖЕНИЕ А

### Расчёт КТУ

Заполнил: В.В. Асмоловский

**Таблица 1**

	Асмоловский	Логинов	Мухаметшин	Пилецкая	Цыбин
Асмоловский	1	2	2	2	1
Логинов	0	1	1	1	0
Мухаметшин	0	1	1	1	0
Пилецкая	0	1	1	1	0
Цыбин	1	2	2	2	1

Заполнил: Д.А. Логинов

**Таблица 2**

	Асмоловский	Логинов	Мухаметшин	Пилецкая	Цыбин
Асмоловский	1	2	2	2	1
Логинов	0	1	1	1	0
Мухаметшин	0	1	1	1	0
Пилецкая	0	1	1	1	0
Цыбин	1	2	2	2	1

Заполнил: В.Н. Мухаметшин

**Таблица 3**

	Асмоловский	Логинов	Мухаметшин	Пилецкая	Цыбин
Асмоловский	1	2	2	2	1
Логинов	0	1	1	1	0
Мухаметшин	0	2	1	1	0
Пилецкая	0	2	1	1	0
Цыбин	1	2	2	2	1

Заполнила: А.Ю. Пилецкая

**Таблица 4**

	Асмоловский	Логинов	Мухаметшин	Пилецкая	Цыбин
Асмоловский	1	2	2	2	2
Логинов	0	1	1	1	0
Мухаметшин	1	2	1	2	1
Пилецкая	0	1	0	1	0
Цыбин	1	2	2	2	1

Заполнил: Д.В. Цыбин

**Таблица 5**

	Асмоловский	Логинов	Мухаметшин	Пилецкая	Цыбин
Асмоловский	1	2	2	2	2
Логинов	1	1	1	1	1
Мухаметшин	1	1	1	1	1
Пилецкая	1	2	2	1	1
Цыбин	1	2	2	2	1