1. **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 [1].

* 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 1 представлена диаграмма проектных классов, реализуемых в системе.

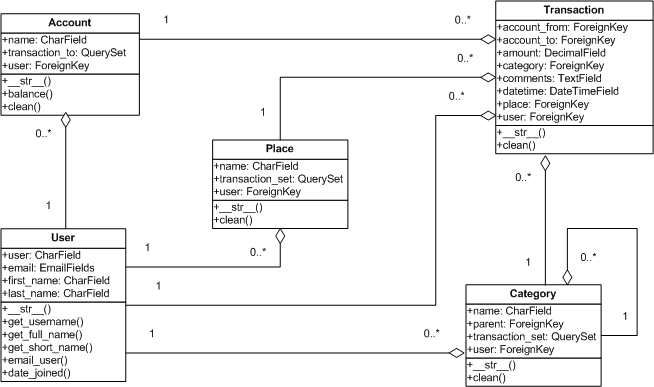


Рисунок 1 ­– Диаграмма проектных классов

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.

Cтоит выделить 3 метода, используемых в данной диаграмме, двое из которых – clean() и \_\_str\_\_(), перезаписанные функции по умолчанию, а balance() является самописной:

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

1. 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

1. 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

* 1. **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 [2].

На рисунке 2 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.

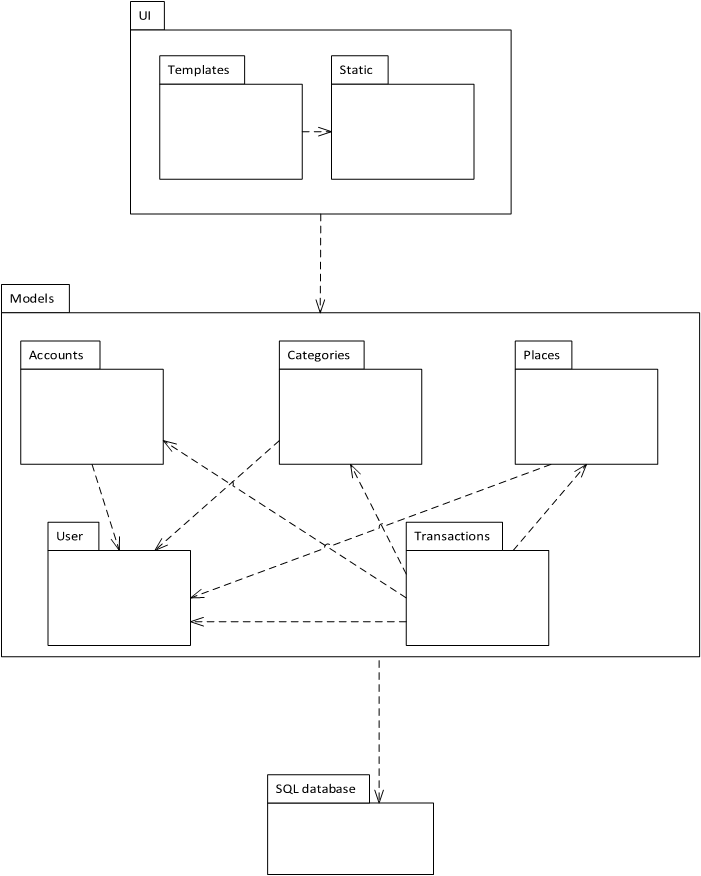


Рисунок 2 – Диаграмма пакетов

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

# 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.

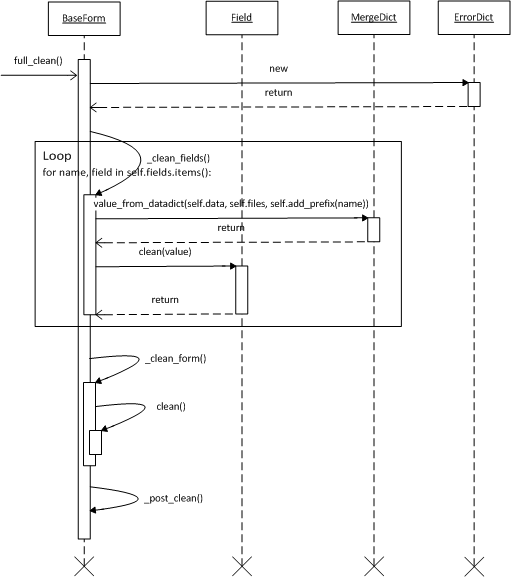


Рисунок 3 – Диаграмма последовательности full\_clean в BaseForm

В ходе выполнения курсового проекта были спроектированы диаграммы последовательностей для нескольких функций. На рисунке 3 изображена диаграмма последовательности для функции full\_clean в классе BaseForm.

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 функции full\_clean в классе Model:

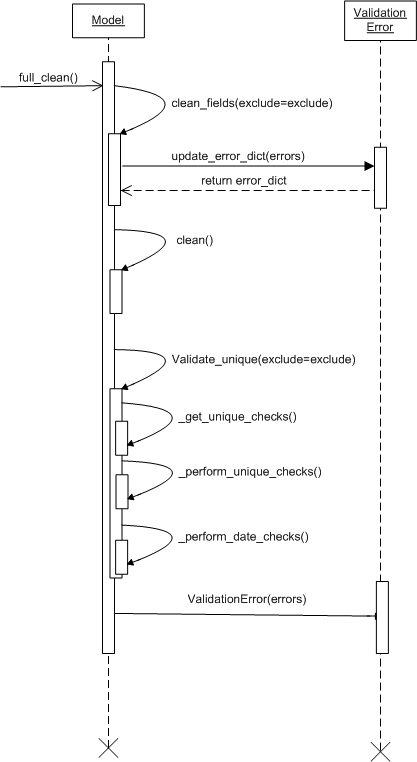


Рисунок 4 – Диаграмма последовательности full\_clean в Model

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)

# 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)

Диаграмма последовательности для функции balance в классе Account показана на рисунке 5, код метода приеден в разделе 3.1.

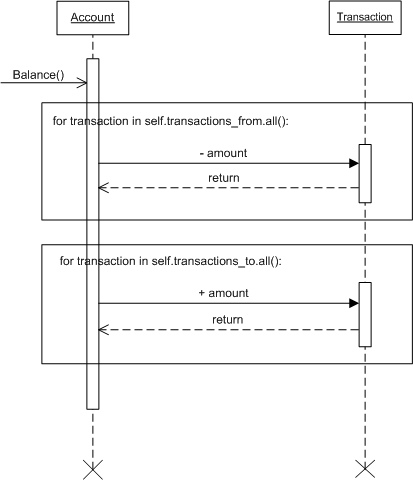


Рисунок 5 – Диаграмма последовательности balance в Account

# Список литературы

# Арлоу Д., Нейштадт А. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание / пер. с англ. – СПб.: Символ-Плюс, 2007. – 624 с.

1. [ru.wikipedia.org/wiki/Диаграмма\_последовательности](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8)