

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
BACK-END ДЛЯ СЕРВИСА ПО АРЕНДЕ ЖИВОТНЫХ
БГУИР КП 1-40 04 01 324 ПЗ

Студент
Руководитель

Е. В. Чижевская
И. А. Удовин

Минск 2020

Оглавление

Вступление.....	3
Анализ литературы по теме дипломного проекта.....	4
Анализ прототипов.....	4
Требования на основе анализа прототипа.....	6
Моделирование предметной области.....	9
Терминология.....	9
Разработка логической модели данных.....	10
Разработка функциональной модели.....	14
Диаграмма состояний.....	16
Диаграмма активности.....	18
Диаграмма потоков данных.....	19
Функциональные требования.....	25
Проектирование программного средства.....	31
API.....	31
Основные компоненты.....	32
Пакеты и классы.....	34
Тестирование.....	36
Методика использования разработанного программного средства.....	39
Заключение.....	41
Список использованных источников.....	42
Приложения.....	43
Приложение 1.....	43
Приложение 2.....	44
Приложение 3.....	45
Приложение 4.....	46
Приложение 5.....	47

Вступление

В качестве темы своего дипломного проекта я ввела реализацию back-end для сервиса по аренде домашних животных. В Беларуси и России этот сегмент предпринимательства почти не освоен, поэтому конкуренция невысокая. Однако население пока не привыкло к такой услуге, поэтому затраты на рекламу будут довольно высокими.

Подавляющее большинство людей любят домашних животных, но не все имеют возможность содержать их у себя дома. Особенно это касается крупных городов, поскольку квартиры в основном не слишком большие, а выгуливать питомца не всегда есть время.

Причин, по которым аренда животных будет отличным бизнесом много. В первую очередь это спрос на услугу:

1. Кошек берут очень часто на новоселье. Согласно примете, чтобы жизнь в новом доме или квартире была счастливой, нужно сперва запустить туда кошку.
2. Чтобы проверить, есть ли у кого-то в семье аллергия на то или иное животное, берут питомца напрокат. Особенно это касается семей, где есть маленькие дети.
3. Опробовать свои силы в уходе за дорогим, породистым псом или котом, а также за экзотическим домашним любимцем.
4. Эмоциональная разрядка. Людям часто нужно отдохнуть, расслабиться, снять стресс. В этом отлично помогут коты и кошки.
5. В хозяйстве могут пригодиться коты, отлавливающие грызунов. Собак могут брать для охраны дачного участка на сезон.
6. Животные для фотосессии. Зачастую для красивого образа на фото нужен питомец, разумеется, покупать его нет смысла. Особенно часто такой услугой пользуются частные фотографы или фотосалоны.

Короче говоря, для многих людей на самом деле гораздо проще арендовать домашних питомцев, а не приобретать на всю жизнь. Именно поэтому такой бизнес нередко встречается в странах Европы, США и Японии.

Анализ литературы по теме дипломного проекта

Анализ прототипов

Так как в России и Беларуси данный бизнес еще очень плохо развит, то не существует специальных сайтов для сдачи в аренду домашних питомцев. В данное время такая услуга осуществляется только с помощью объявлений в интернете или газете и стала особенно популярна во время пандемии. Поэтому в качестве прототипа для анализа я решила выбрать сайт для аренды животных в Японии, городе Токио фирмы Wanpaku land, расположенные по адресу <http://www.wanpakuland.com/>.

Основные положительные черты сайта:

1. Наличие индивидуальной страницы для каждого животного
2. Проверка животного администраторами сайта перед его добавлением
3. Подробное описание не только размеров животного, но и его повадок и черт характера
4. Возможность связаться с хозяином животного как по email, так и по телефону

Основные отрицательные черты сайта

1. Сайт на японском языке из-за чего его достаточно сложно использовать например туристам или людям, недавно переехавшим в страну
2. Нет отзывов прошлых арендаторов
3. Невозможно оставить жалобу и так же нет возможности оценить животное, чтобы упростить дальнейшим клиентам их выборочный
4. Невозможность владельцу животного самому выставить цену за аренду своего питомца, так как для всех пользователей сайта цена аренды одинаковая

5. Одинаковая информация о животном и цена, как для физических, так и для юридических лиц

6. Невозможность указать награды животного

7. Невозможность указать, как можно забрать животное (необходимо вывести его самому или существует доставка)

Требования на основе анализа прототипа

Назначение разработки

Необходимо создать back-end для сервиса по аренде домашних животных

Целевая аудитория:

- Владельцы животных
- Люди которые хотят арендовать домашних животных

Риски:

- Риск произвести неправильную оценку сложности проекта и не уложиться в поставленные сроки.
- Риск наличия неквалифицированных сотрудников.
- Риск не учесть некоторые аспекты предметной области.
- Риск создать не конкурентоспособный продукт.
- Риск потерять ценного сотрудника

Перечень основных выполняемых функций

Приложение должно давать возможность зарегистрироваться, взять в аренду животное или добавить свое для аренды, написать отзыв о животном или жалобу, возможность оценить животное, а также изменить пароль и/или телефон. Для администратора так же должна быть возможность одобрить животное и заблокировать или разблокировать пользователя.

Входные и выходные данные

Для различных операций входные и выходные данные будут различными, но они все будет в формате JSON.

Среда эксплуатации

Наличие Python версии 3.6 и выше, PostgreSQL версии 9.5.24 и выше и pip версии 19.0.1 и выше.

Обоснование выбора языка и сред разработки

Разработка будет вестись на языке python с использованием базы данных PostgreSQL и фреймворка Django.

Одним из преимуществ языка Python является кроссплатформенность. Python – это интерпретируемый язык, его интерпретаторы существуют для многих платформ. Поэтому с запуском его на любой ОС не должно возникнуть проблем. С Python доступно огромное количество сервисов, сред разработки, и фреймворков. Легко можно найти подходящий продукт для работы. Возможность подключить библиотеки, написанные на C. Это позволяет повысить эффективность, улучшить быстродействие. Python отличается строгим требованием к написанию кода (требует отступы), что является преимуществом. Изначально язык способствует писать код организованно и красиво. Для большинства задач: для веб-разработки, для скриптов, прототипирования, машинного обучения и работы с большими данными, — один из лучших языков. Все эти преимущества способствуют тому, что в качестве основного языка для написания back-end для сервиса будет выбран именно Python.

Фреймворк Django был выбран по следующим причинам:

1. Django содержит огромное количество функциональности для решения большинства задач веб-разработки. Например, ORM, миграции базы данных, аутентификация пользователя, панель администратора, формы и т.д.

2. Django как фреймворк задаёт структуру проекта. Она помогает разработчикам понимать, где и как добавлять новую функциональность. Благодаря одинаковой для всех проектов структуре гораздо проще найти уже готовые решения или получить помощь от сообщества. Огромное количество увлеченных разработчиков поможет справиться с любой задачей гораздо быстрее.

3. Приложения в Django позволяют разделить проект на несколько частей. Приложения устанавливаются путём добавления в `settings.INSTALLED_APPS`. Этот подход позволяет легко интегрировать готовые решения. Сотни универсальных модулей и приложений очень сильно ускоряют разработку.

4. Django безопасен из коробки и включает механизмы предотвращения распространенных атак вроде SQL-инъекций (XSS) и подделки межсайтовых запросов (CSRF).

5. Django REST Framework, который часто сокращают до «DRF», является библиотекой для построения API. Он имеет модульную и настраиваемую архитектуру, которая хорошо работает для создания как простых, так и сложных API. В DRF политики аутентификации и разрешений доступны из коробки. Он поставляется с базовыми классами для CRUD операций и встроенной утилитой для тестирования разрабатываемого API.

Основными причинами выбрать PostgreSQL были широкая поддержка различных типов данных, которыми не обладают другие реляционные СУБД, а так же хорошая поддержка Django именно PostgreSQL, т.к. фреймворк изначально писался для использования именно этой СУБД и оптимизирует многие запросы, а так же существует отдельная библиотека для работы с PostgreSQL, которая позволяет полностью использовать функционал, специфичный для этой СУБД.

Моделирование предметной области

Терминология

Гость – это не авторизованный пользователь сервиса.

Зарегистрированный пользователь – это пользователь имеющий учётную запись в сервисе.

Администратор – это зарегистрированный пользователь, который должен рассматривать заявки на добавление животных для сдачи в аренду; может управлять пользователями, отзывами.

Бизнес пользователь – зарегистрированный пользователь, являющийся индивидуальным предпринимателем или юридическим лицом, использующий сервис в коммерческих целях.

Пользователь-клиент – зарегистрированный пользователь, являющийся физическим лицом, использующий сервис в личных целях.

Разработка логической модели данных

Проведённый анализ предметной области позволил выявить следующий набор сущностей:

- Пользователи (rent_users)
- Животные (animals)
- Заявка на аренду животного (rent_request)
- Заявка на добавление животного для сдачи в аренду (add_animal_for_rent_requests)
- Портфолио животного (portfolios)
- Отзывы (reviews)
- Жалобы (Complaint)

Схема данных в нотации IDEF1.X – логическая ER-модель, приведенная на рисунке 1, показывает сущности, их атрибуты, связи между сущностями, первичные и внешние ключи.

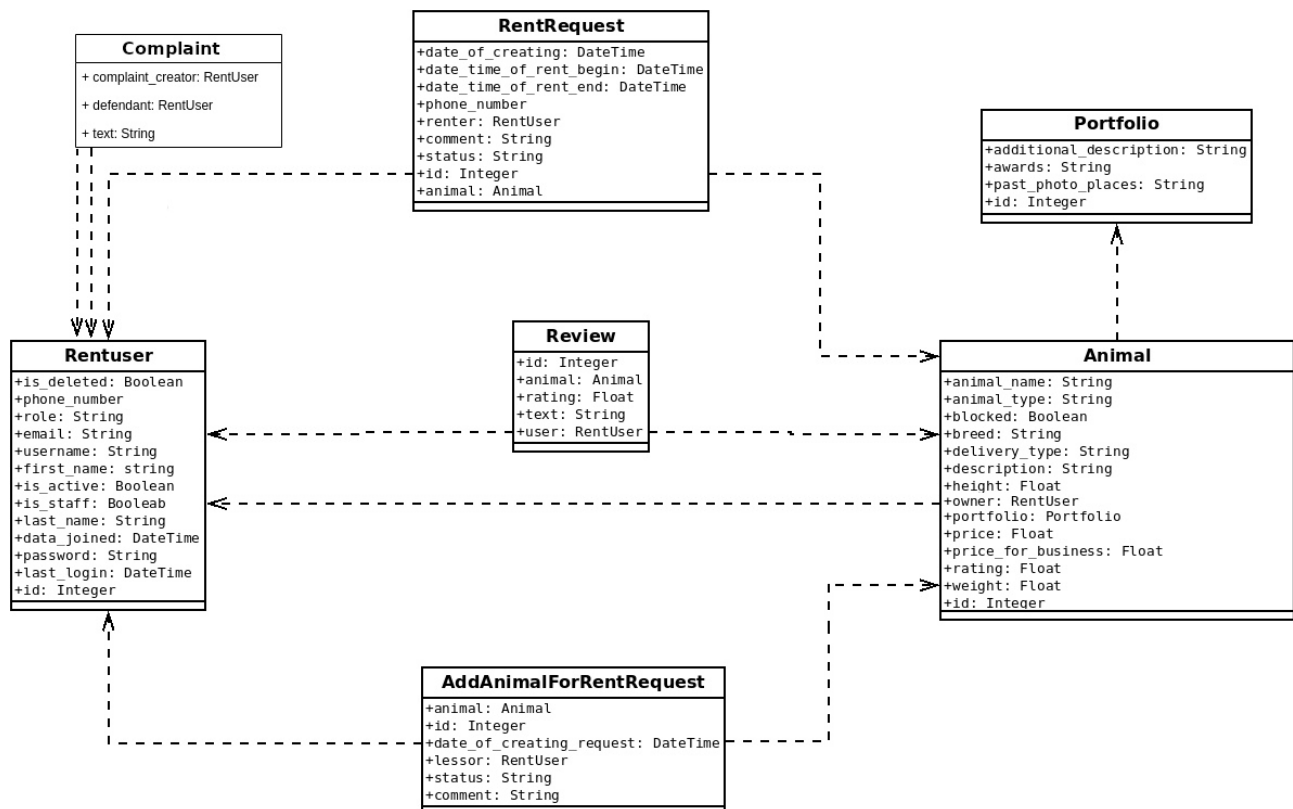


Рисунок 1- Логическая ER-модель

Между сущностями установлены следующие отношения:

- отзыв относится к одному животному, у животного может быть много отзывов (1:M);

- отзыв относится к одному пользователю, у пользователя может быть много отзывов (1:M);

-отзыв относится к одному пользователю, у пользователя может быть много отзывов (1:M);

- заявка на аренду относится к одному пользователю, у пользователя может быть много запросов(1:M);

- заявка на аренду относится к одному животному, у животного может быть много запросов(1:M);

- заявка на добавление животного для сдачи в аренду относится к одному пользователю, у пользователя может быть много заявок (1:M);

- заявка на добавление животного для сдачи в аренду относится к одному животному, у животного может быть много заявок (1:M);

- животное относится к портфолио, у животного может быть одно портфолио (1:1);

-животное относится к пользователю, у пользователя может быть много животных (1:M);

Проверка перечисленных объектов на соответствие различным нормальным формам позволила сделать вывод о нахождении отношений в третьей нормальной форме.

Атрибуты сущностей.

Сущность "Пользователь" имеет следующие атрибуты:

- Id – идентификатор
- password – пароль пользователя для входа в сервис
- last_login – дата последнего логина
- username – имя пользователя(никнейм)
- first_name – имя
- last_name – фамилия
- email – электронная почта
- is_active – является ли пользователь активным

- is_staff – является ли пользователь администратором
- date_joined – дата регистрации
- is_deleted – удалён ли пользователь
- phone_number – номер телефона
- role – роль

Сущность “Животное” имеет следующие атрибуты:

- id – идентификатор
- animal_type – вид животного
- breed – порода
- height – рост
- weight – вес
- delivery_type – способ доставки
- rating – рейтинг
- description – описание
- price – стоимость для физ. лиц
- price_for_business – стоимость для юр. лиц и ИП
- is_blocked – заблокирован ли пользователь
- animal_name – кличка животного
- portfolio_id – идентификатор портфолио
- owner_id – идентификатор владельца

Сущность “Портфолио” имеет следующие атрибуты:

- id – идентификатор
- additional_description – дополнительно описание
- past_photoplaces – предыдущие места съёмок
- awards – награды

Сущность “Жалоба” имеет следующие атрибуты:

- id – идентификатор
- complaint_creator – создатель жалобы

- defendant – тот, на кого была создана жалоба
- text – текст жалобы

Сущность “Заявка на аренду” имеет следующие атрибуты:

- id – идентификатор
- date_of_creating – дата создания
- date_of_rent_begin – дата начала аренды
- date_of_rent_end – дата окончания аренды
- phone_num – номер телефона арендатора
- status – статус заявки
- comment – комментарий к заявке
- animal_id – идентификатор животного
- renter_id – идентификатор арендатора

Сущность “Отзыв” имеет следующие атрибуты:

- id – идентификатор
- rating – рейтинг
- rent_user_id – идентификатор пользователя
- animal_id – идентификатор животного

Сущность “Заявка на добавление животного для сдачи в аренду” имеет следующие атрибуты:

- id - идентификатор
- status – статус заявки
- comment – комментарий
- animal_id – идентификатор животного
- lessor_id – идентификатор пользователя

Разработка функциональной модели

Функциональную модель предметной области представим в виде диаграммы вариантов использования в нотации UML, представляющей систему в виде набора варианта использования и актеров, взаимодействующих с ними.

В рамках предметной области можно выделить двух актеров:

- администратор;
- пользователь.

Диаграмма вариантов использования системы приведена на рисунке 2

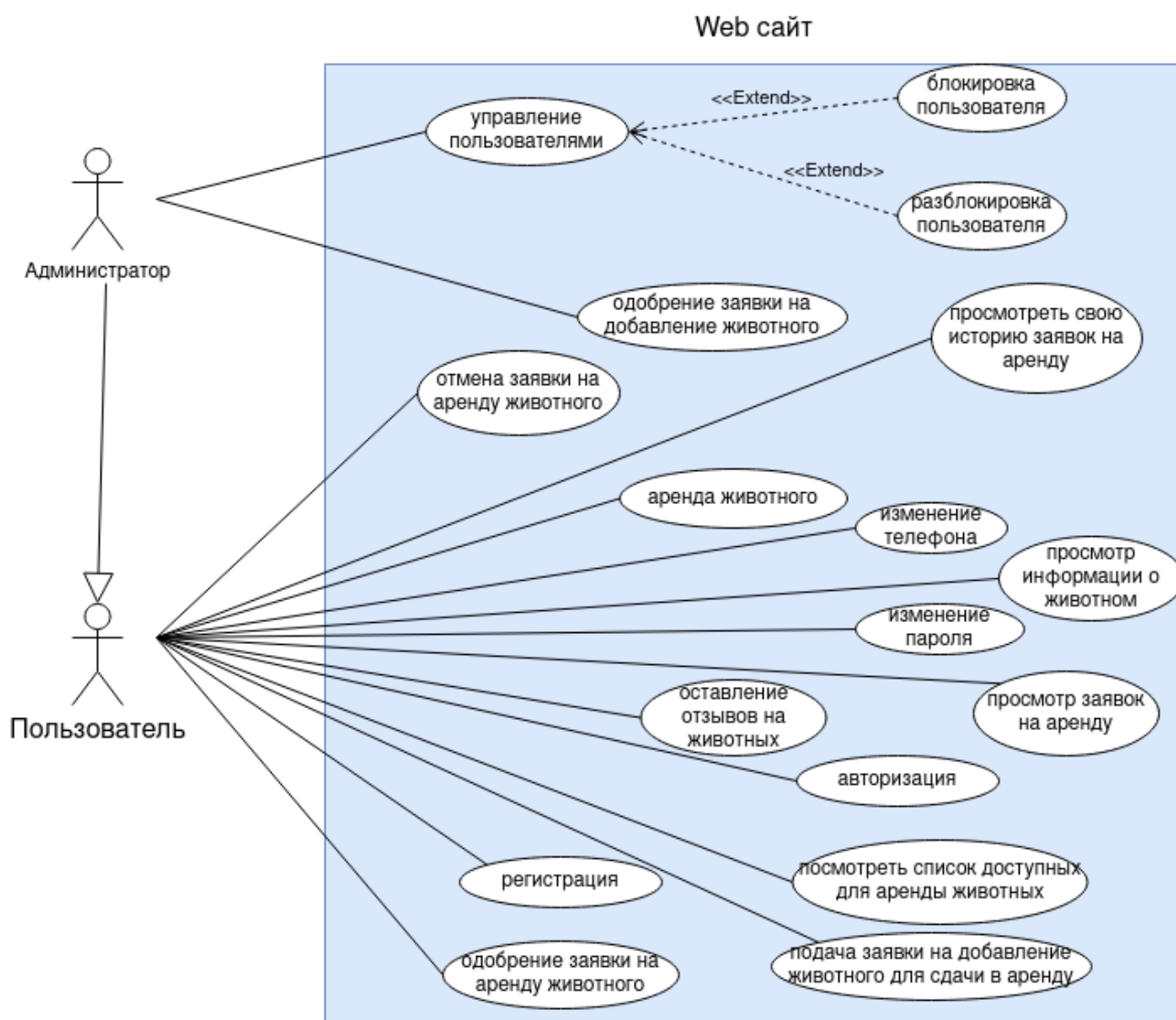


Рисунок 2 - Диаграмма вариантов использования системы

Согласно приведенной диаграмме пользователь применяет следующие варианты использования:

- регистрация
- авторизация
- просмотр доступных для аренды животных
- просмотр информации о животном
- аренда животного
- отмена заявки на аренду животного
- просмотр своей истории заявок на аренду животного
- просмотр заявок, поданных на аренду животного
- одобрение заявки на аренду животного
- оставление отзыва на животного
- оставление жалобы
- подача заявки на добавление животного для сдачи в аренду
- изменение пароля, телефона

Администратор применяет варианты использования, доступные пользователю, а также дополнительные варианты использования – управление пользователями (блокировка пользователя, разблокировка пользователя), одобрение заявки на добавление животного.

Диаграмма состояний

Диаграмма состояний приложения (рисунок 3).

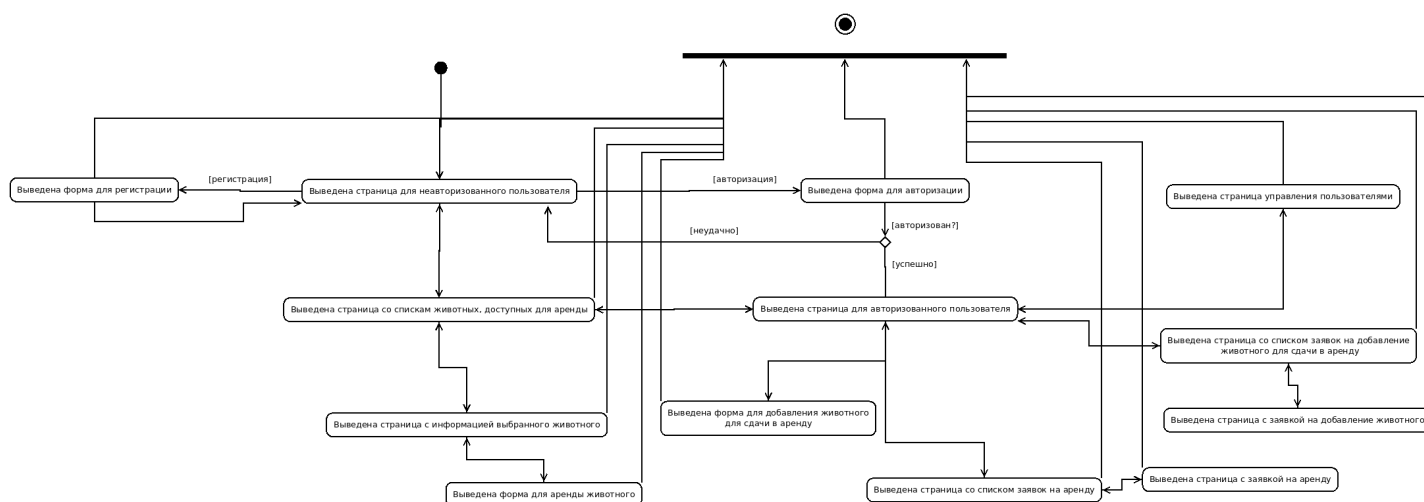


Рисунок 3 - Диаграмма состояний приложения

После запуска приложения система принимает состояние «Выведена форма для неавторизованного пользователя».

Дальнейшие состояния зависят от функции или операции, выбранной пользователем.

Если пользователь выбирает операцию авторизации – происходит переход к состоянию “Выведена форма для авторизации”. Если авторизация пройдена успешно, то приложение переходит в состояние “Выведена страница для авторизованного пользователя”.

Если пользователь выбирает операцию регистрации – происходит переход к состоянию “Выведена форма для регистрации”. Если регистрация завершена успешно, то приложение перейдёт в состояние “Выведена страница для неавторизованных пользователей”.

Если пользователь выбирает операцию отображения доступных животных – происходит переход к состоянию “Выведена страница со списком животных, доступных для аренды”.

Если пользователь выбирает операцию просмотра информации животного – происходит переход к состоянию “Выведена страница с информацией о животном”.

Если пользователь выбирает операцию арендовать животное – происходит переход к состоянию “Выведена форма для аренды животного”.

Если пользователь выбирает операцию добавить животное – происходит переход к состоянию “Выведена форма для добавления животного для сдачи в аренду”.

Если пользователь выбирает операцию просмотреть заявки на аренду – происходит переход к состоянию “Выведена страница со списком заявок на аренду”.

Если пользователь выбирает операцию просмотреть заявку – происходит переход к состоянию “Выведена страница с заявкой на аренду”.

Если пользователь выбирает операцию просмотра заявок на добавление животного – происходит переход к состоянию “Выведена страница со списком заявок на добавление животного для сдачи в аренду”.

Если пользователь выбирает операцию просмотреть заявку на добавление животного – происходит переход к состоянию “Выведена страница с заявкой на добавление животного”.

Диаграмма активности

Диаграмма активности для варианта использования «Аренда животного» (рисунок 4).

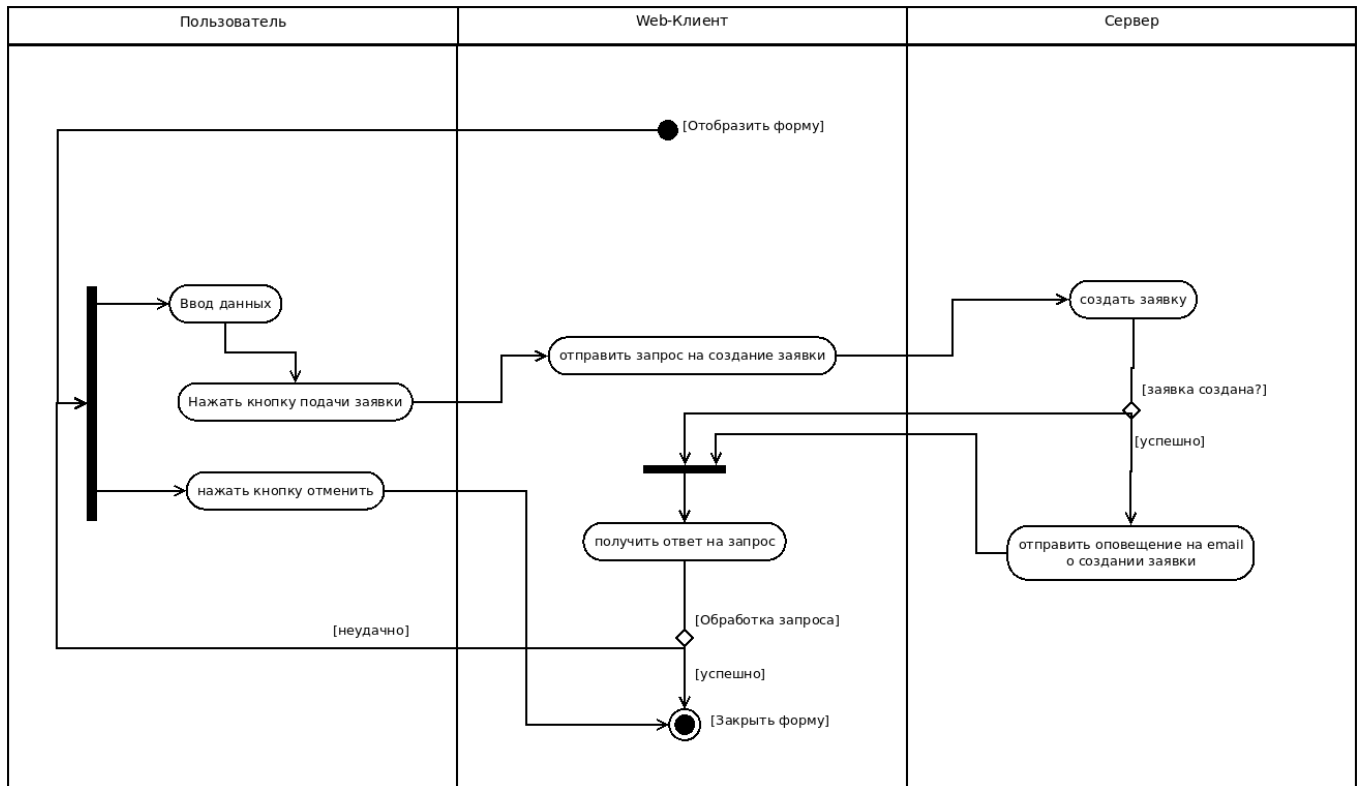


Рисунок 4 - Диаграмма активности для варианта использования «Аренда животного»

Активность предполагает состояние «Выведена форма для аренды животного» и включает следующие действия:

- пользователь вводит данные заявки на аренду животного (если пользователь отменяет операцию, активность завершается);
- приложение сохраняет данные заявки в базу данных;
- приложение отправляет оповещение на email пользователя;
- приложение закрывает форму обращения.

Диаграмма потоков данных

Для составления потоков данных могут использоваться две нотации (рисунок 5):

- Нотация Юрдана и Коада
- Нотация Гейна и Сарсона



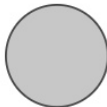





Нотация	Юрдан и Коад	Гейн и Сарсон
Внешняя сущность		
Процесс		
Хранилище данных		
Поток данных		

Рисунок 5 - Нотации для составления потоков данных

DFD диаграмма для регистрации

Пользователь вводит данные о себе, затем отправляет запрос на сервер. На сервере обрабатываются данные пользователя из запроса и возвращается ответ на запрос (рисунок 6).

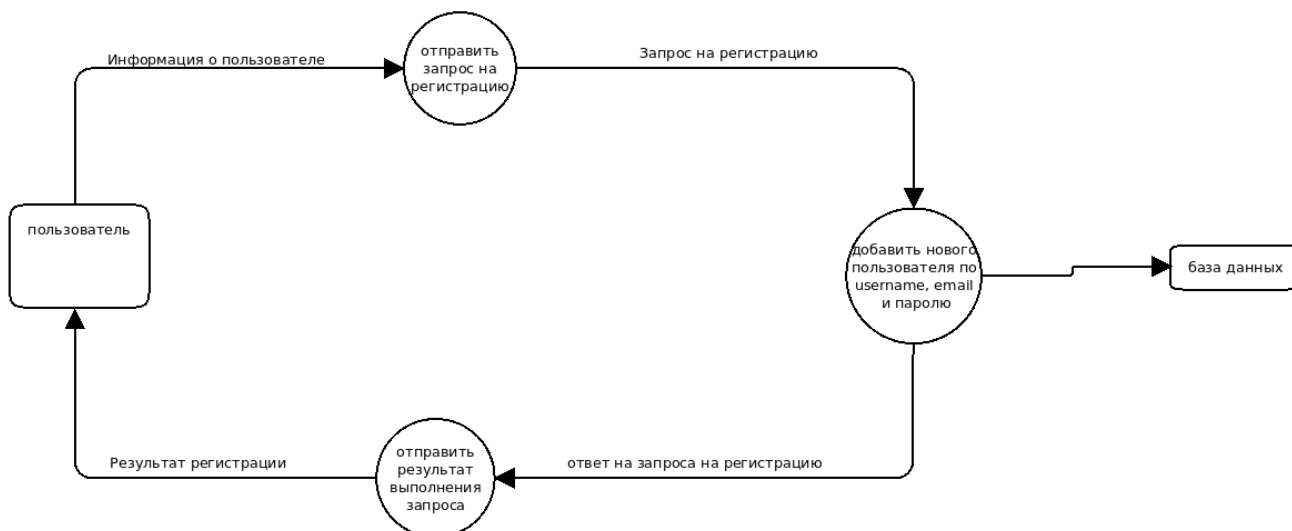


Рисунок 6 - DFD диаграмма для регистрации

DFD диаграмма для авторизации

Пользователь вводит email и пароль и отправляет на сервер запрос. На сервере введённые данные обрабатываются, затем сервер отправляет ответ на запрос. Если все данные введены верно, то в ответе будет информация о пользователе, необходимая для дальнейшей работы авторизованного пользователя (рисунок 7).

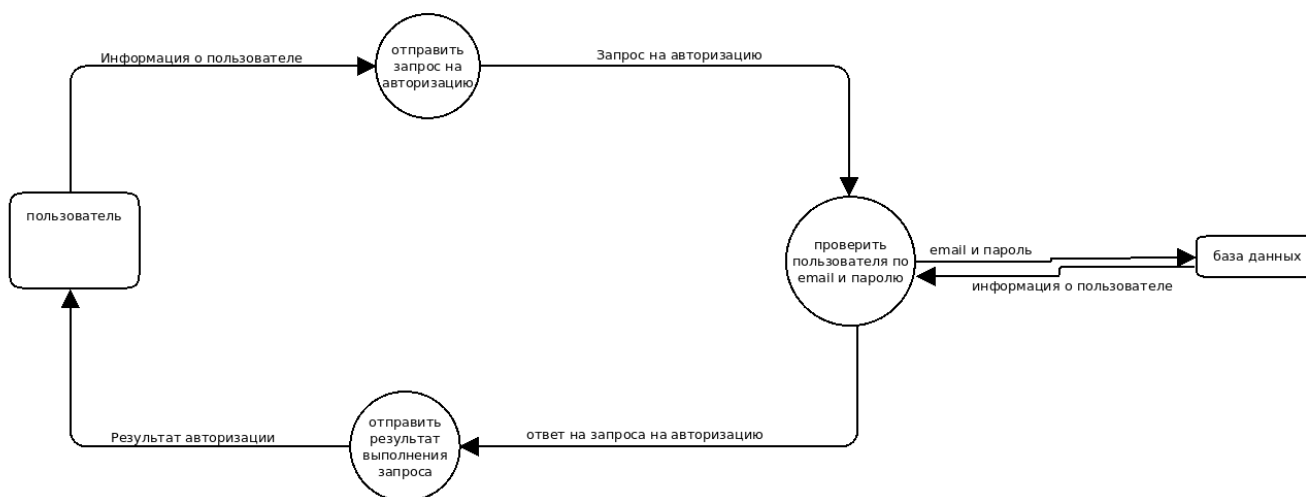


Рисунок 7 - DFD диаграмма для авторизации

DFD диаграмма для добавления животного

Пользователь вводит данные о животном, которое хочет добавить, и отправляет запрос на сервер на добавление животного. На сервере проверяется, что животное не является опасным. Затем по результатам проверки животное добавляется в базу данных. После этого сервер отправляет ответ на запрос (рисунок 8).

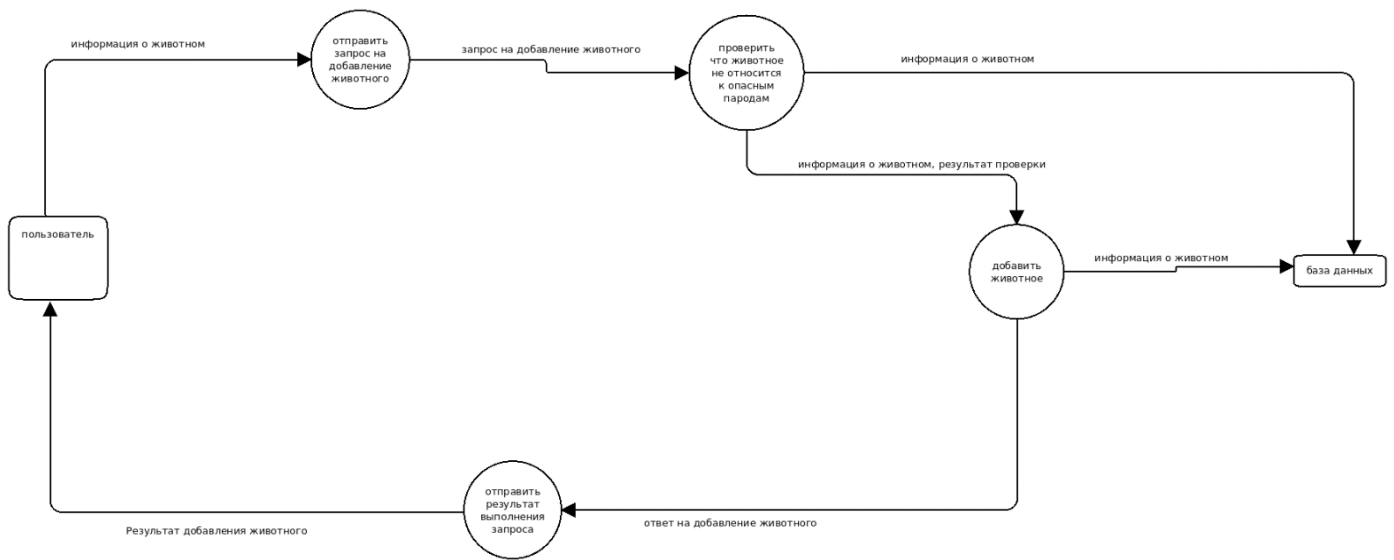


Рисунок 8 - DFD диаграмма для добавления животного

DFD диаграмма для одобрения животного

Пользователь, являющийся администратором, отправляет запрос на сервер для получения списка заявок на добавление. Затем пользователь открывает страницу с заявкой. После отправляет запрос на одобрение заявки на добавление животного. Сервер обрабатывает запрос и возвращает ответ (рисунок 9).

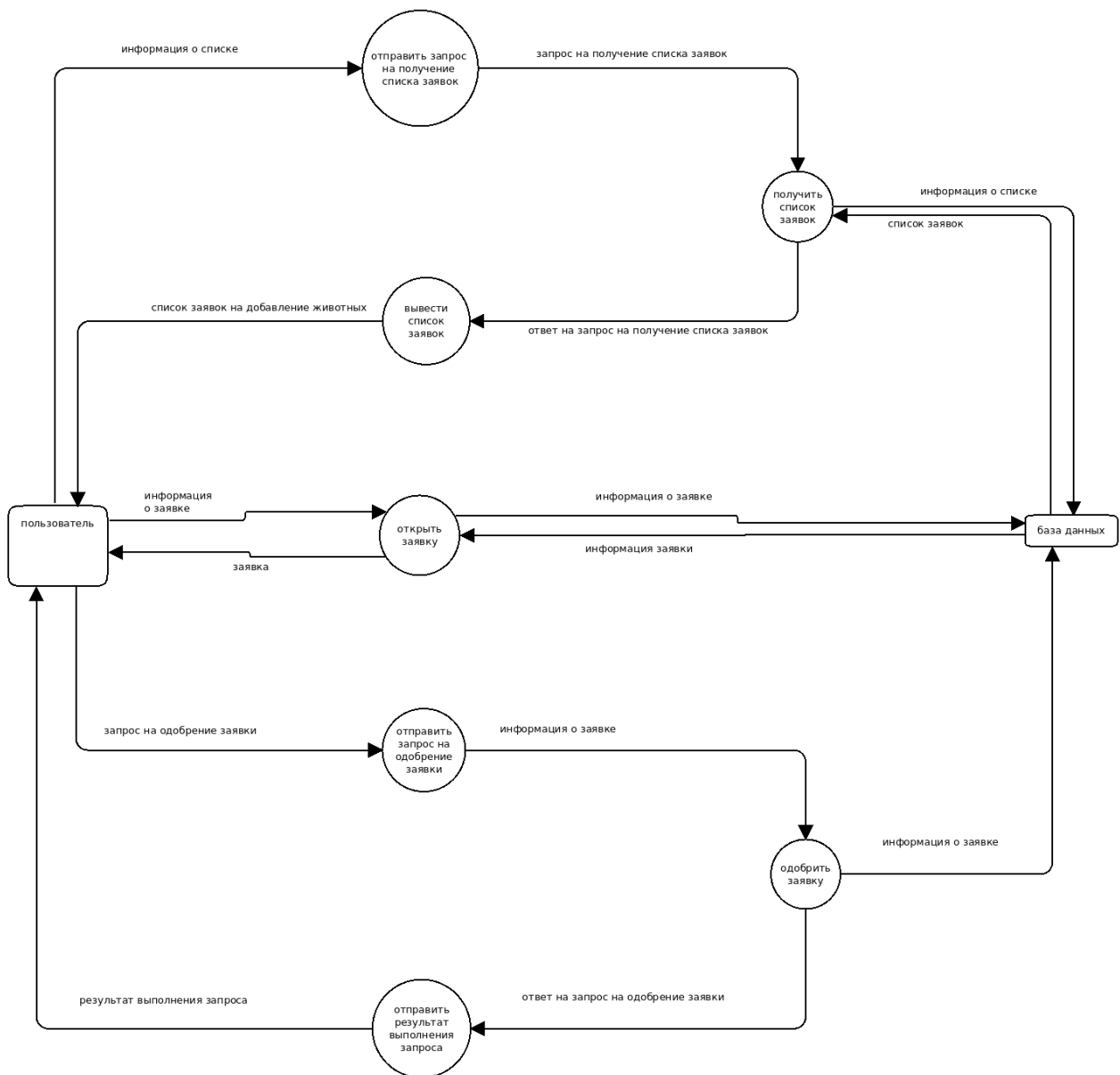


Рисунок 9 - DFD диаграмма для добавления животного

DFD диаграмма для аренды животного

Пользователь открывает список животных. Затем открывает страницу животного. После отправляет запрос на аренду животного на сервер. Сервер обрабатывает запрос и возвращает ответ (рисунок 10).

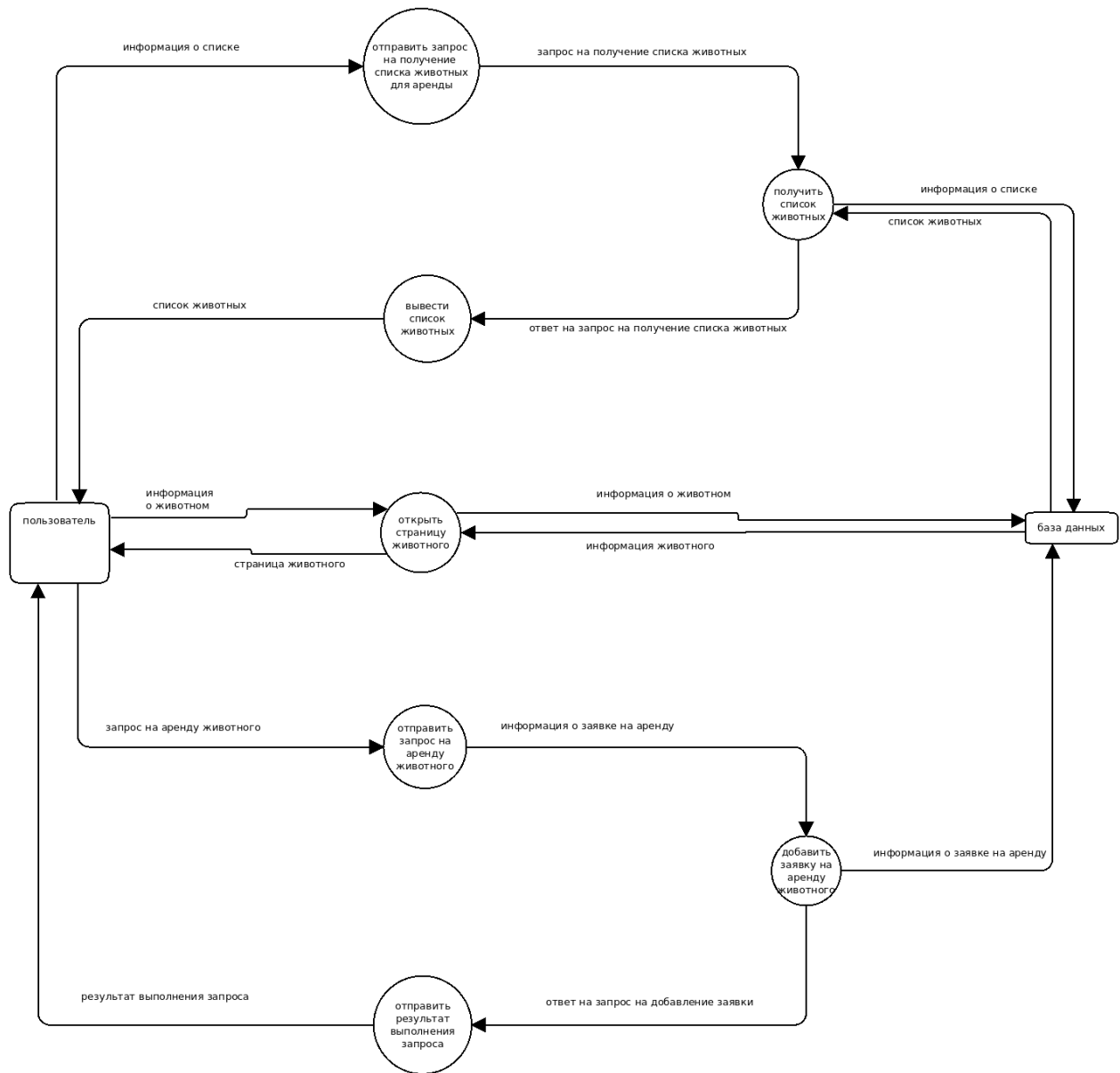


Рисунок 10 - DFD диаграмма для аренды животного

DFD диаграмма для одобрения заявки на аренду

Пользователь, являющийся арендодателем, открывает список заявок на аренду, затем открывает страницу с заявкой. После отправляет запрос на сервер на одобрение заявки. Сервер обрабатывает запрос и возвращает ответ (рисунок 11).

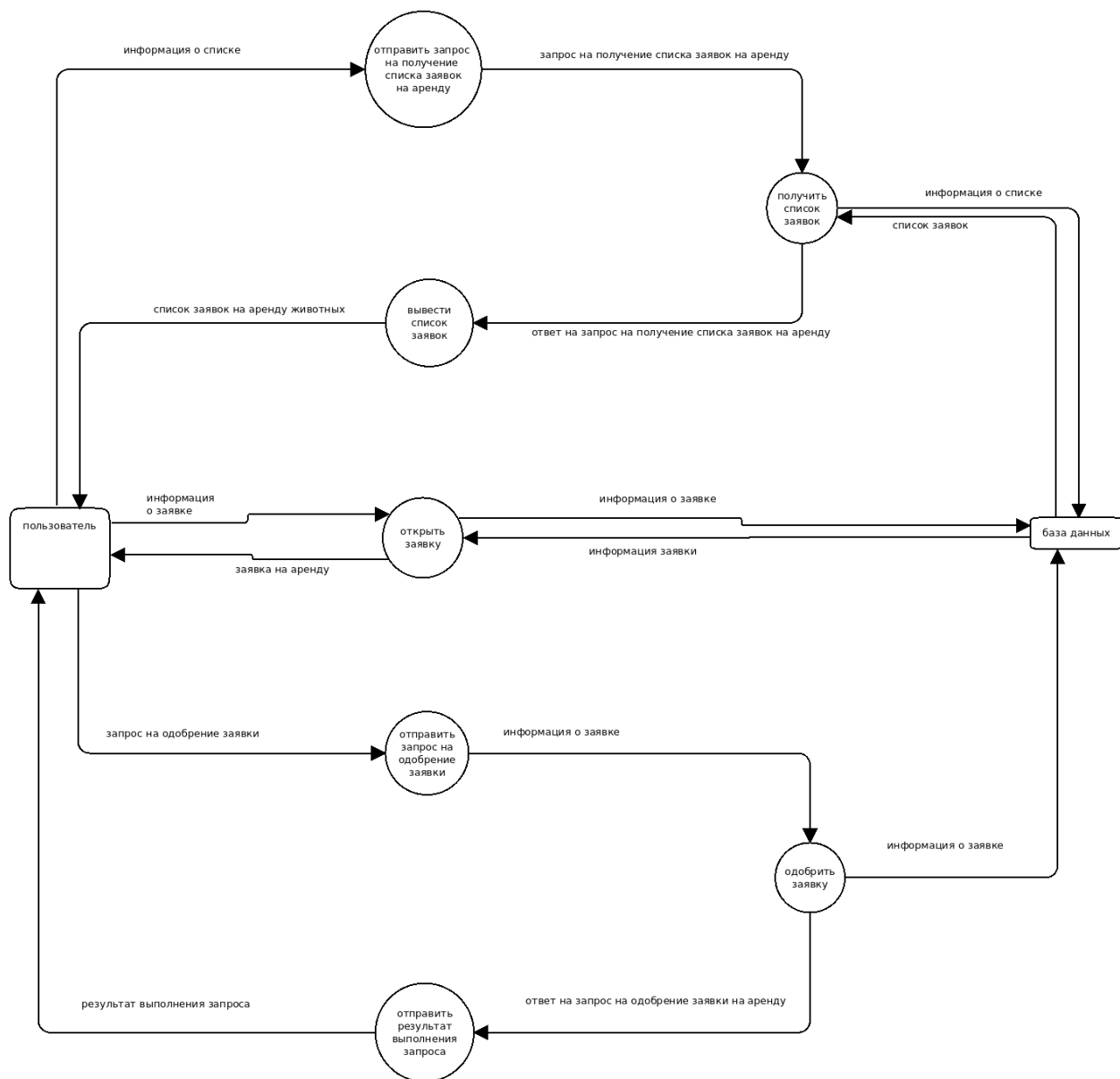


Рисунок 11 - DFD диаграмма для одобрения заявки на аренду

Функциональные требования

Общие требования: сервис является веб-приложением.

1. Любой пользователь должен иметь возможность просмотреть информацию о животных, сдаваемых в аренду.

Страница со списком предложений должна содержать таблицу со следующими полями:

- кличка
- описание

На странице животного для пользователя, являющегося гостем или клиентом, должны быть отображены следующие поля:

- Кличка
- Рост
- Масса
- Описание
- Стоимость аренды для физических лиц

На странице животного для бизнес-пользователя должны быть отображены следующие поля:

- Кличка
- Рост
- Масса
- Описание
- Стоимость аренды для ИП и юридических лиц
- Список наград
- Список предыдущих съёмок

2. Пользователь должен иметь возможность зарегистрироваться при помощи формы.

Форма регистрации пользователя-клиента должна содержать следующие поля:

- Электронная почта

- Пароль
- Подтверждение пароля
- Номер телефона
- Имя
- Отчество
- Является ли пользователь ИП или юридическим лицом (бизнес пользователь)

3. Пользователь должен иметь возможность авторизоваться при помощи формы.

Форма должна содержать следующие поля:

- Электронная почта
- Пароль

Все поля обязательны для заполнения.

При нажатии на кнопку регистрации приложение должно проверить правильность введённых данных. В результате проверки введённых данных пользователь либо входит в учётную запись, либо выводится сообщение о неправильности введённых данных.

Если пользователь заблокирован, то ему выводится сообщение о блокировке.

4. Зарегистрированному пользователю должна быть предоставлена возможность сменить пароль, номер телефона.

Форма для изменения данных должна иметь следующие поля:

- Новый пароль
- Подтверждение нового пароля
- Текущий пароль
- Новый номер телефона

Приложение должно произвести проверку корректности введённых данных (соответствие нового пароля требованиям, новый пароль не должен совпадать со старым). При успешной смене пароля пользователю должно прийти уведомление о смене пароля на электронную почту. Если введённые данные некорректны, то пользователю должно быть выведено сообщение об этом.

5. Зарегистрированный пользователь, должен иметь возможность убрать своё предложение о сдаче в аренду животного.

Пользователю должна быть доступна страница, где будет выведен список его предложений о сдаче животных в аренду.

Список предложения должен быть со следующими полями:

- вид животного
- порода
- кличка

6.Зарегистрированный пользователь должен иметь возможность просмотреть:

- историю заявок на аренду животного
- историю заявок на добавление животного для сдачи
- Список животных, которых пользователь сдаёт в аренду

История заявок на аренду животного должна содержать:

- Дата подачи заявки
- Дата и время начала аренды
- Дата и время окончания аренды
- Стоимость аренды
- Вид животного
- Кличка животного
- Статус заявки

Истории заявок на добавление животного для сдачи должны содержать:

- Дата подачи заявки
- Вид животного
- Кличка животного
- Статус заявки

7. Зарегистрированный пользователь, должен иметь возможность подать заявку на добавление животного для сдачи в аренду, для этого необходимо предоставить следующие данные.

- Кличка животного

- Вид
- Порода
- Масса
- Рост
- Описание
- Доставка

Все поля, кроме поля подвид, обязательны для заполнения. Пользователю приходит письмо с уведомлением о заявке и текущим статусом. При изменении статуса заявки, пользователю должно приходиться уведомление на электронную почту.

Доставка может быть двух видов: самовывоз, доставка.

Приложение должно проверять, что указанное в заявке животное не относится к потенциально опасным видам. В случае если животное является потенциально опасным, пользователю должно быть выведено сообщение об этом.

8. Зарегистрированный пользователь, являющийся администратором, должен иметь возможность просмотреть поданные пользователями заявки на добавление животных для сдачи в аренду

Страница с заявками на добавление животного для сдачи в аренду должна содержать:

- Дата подачи
- Адрес электронной почты пользователя, который подал заявку
- Вид животного
- Порода животного
- Статус заявки

Для каждой заявки должно быть более подробное описание, которое должно содержать:

- Дату подачи заявки
- Адрес электронной почты пользователя, подавшего заявку
- Кличка животного
- Вид

- Порода
- Масса
- Рост
- Описание
- Текущий статус заявки

У заявки может быть два статуса: не рассмотрена, отклонена, принята.

При изменении статуса заявки пользователь должен получить уведомление на электронную почту.

9. Зарегистрированный пользователь, являющийся администратором, должен иметь возможность блокировать и разблокировать пользователей.

Страница для этого должна содержать следующие поля:

- Адрес электронной почты пользователя
- Текущий статус

Если пользователь блокируется, приложение должно заблокировать все его предложения о сдаче в аренду животных.

10. Зарегистрированный пользователь должен иметь возможность арендовать животное.

Форма аренды животного должна содержать следующие поля:

- дата и время начала аренды
- дата и время окончания аренды
- комментарий
- номер телефона

Все поля, кроме комментария, обязательны для заполнения. При нажатии на кнопку подачи заявки, пользователь должен получить уведомление на электронную почту.

11. Зарегистрированный пользователь должен иметь возможность просмотреть заявки на аренду животного, которое этот пользователь сдаёт в аренду. Для этого пользователю должна быть доступна страница с заявками на аренду со следующими полями:

- дата подачи заявки
- имя, подавшего заявку

- дата начала аренды
- дата окончания аренды

Для каждой заявки должна быть страница с подробным описанием.

Страница заявки должна содержать:

- дата подачи
- имя, пользователя, подавшего заявку
- номер телефона
- комментарий
- вид животного
- порода
- кличка
- дата и время начала аренды
- дата и время окончания аренды
- кнопка для изменения статуса заявки

У заявки может быть три статуса: не рассмотрена, одобрена, отклонена.

При изменении статуса заявки, пользователь, подававший заявку, должен получить уведомление на электронную почту.

12. Зарегистрированный пользователь, должен иметь возможность оставить отзыв, и оценку, и жалобу.

Оценка даётся по шкале от 1 до 5.

Проектирование программного средства

API

API (Application Programming Interface) — это интерфейс программирования, интерфейс создания приложений. RESTful API - это архитектурный стиль для интерфейса прикладных программ (API), который использует HTTP-запросы для доступа и использования данных. Эти данные могут использоваться для типов данных GET, PUT, POST и DELETE, которые относятся к чтению, обновлению, созданию и удалению операций, касающихся ресурсов.

REST, используемый браузерами, можно рассматривать как язык Интернета. С ростом использования облака API используются потребителями облака для предоставления и организации доступа к веб-службам. REST - это логичный выбор для создания API-интерфейсов, которые позволяют пользователям гибко подключаться к облачным службам, управлять ими и взаимодействовать с ними в распределенной среде. API RESTful используются такими сайтами, как Amazon, Google, LinkedIn и Twitter. RESTful API использует команды для получения ресурсов. Состояние ресурса в любой момент времени называется представлением ресурса. RESTful API использует существующие HTTP-методологии, определенные протоколом RFC 2616, например:

- GET для получения ресурса;
- PUT, PATCH для изменения состояния или обновления ресурса, который может быть объектом, файлом или блоком;
- POST для создания этого ресурса; а также
- DELETE, чтобы удалить его.

В REST сетевые компоненты - это ресурс, к которому пользователь запрашивает доступ, например, черный ящик, детали реализации которого неясны. Все звонки не имеют гражданства; Служба RESTful ничего не может сохранить между выполнениями.

Форматы данных, которые поддерживает REST API, включают:

- json
- xml
- x-wbe + xml
- x-www-form-urlencoded
- multipart

Основные компоненты

Основная архитектура

API DRF состоит из 3-х слоев: сериализатора, вида и маршрутизатора.

Сериализатор: преобразует информацию, хранящуюся в базе данных и определенную с помощью моделей Django, в формат, который легко и эффективно передается через API.

Вид (ViewSet): определяет функции (чтение, создание, обновление, удаление), которые будут доступны через API.

Маршрутизатор: определяет URL-адреса, которые будут предоставлять доступ к каждому виду.

Сериализаторы(Serializers)

Модели Django интуитивно представляют данные, хранящиеся в базе, но API должен передавать информацию в менее сложной структуре. Хотя данные будут представлены как экземпляры классов Model, их необходимо перевести в формат JSON для передачи через API.

Сериализатор DRF производит это преобразование. Когда пользователь передает информацию (например, создание нового экземпляра) через API, сериализатор берет данные, проверяет их и преобразует в нечто, что Django может сложить в экземпляр модели. Аналогичным образом, когда пользователь обращается к информации через API, соответствующие экземпляры передаются в сериализатор, который преобразовывает их в формат, который может быть легко передан пользователю как JSON.

Наиболее распространенной формой, которую принимает сериализатор DRF, является тот, который привязан непосредственно к модели Django. Сериализаторы — это невероятно гибкий и мощный компонент DRF. Хотя подключение сериализатора к модели является наиболее распространенным, сериализаторы могут использоваться для создания любой структуры данных Python через API в соответствии с определенными параметрами.

Пример одного из сериалайзеров, использовавшегося при написании сервиса приведен в приложении 1.

Виды (Views)

Сериализатор анализирует информацию в обоих направлениях (чтение и запись), тогда как ViewSet - это тот код, в котором определены доступные операции. Наиболее распространенным ViewSet является ModelViewSet, который имеет следующие встроенные операции:

- Создание экземпляра: create ()

- Получение / чтение экземпляра: `retrieve ()`
- Обновление экземпляра (все или только выбранные поля): `update ()` или `partial_update ()`
- Уничтожение / Удаление экземпляра: `destroy ()`
- Список экземпляров (с разбивкой по страницам по умолчанию): `list ()`

Каждая из этих функций может быть переопределена, если требуется другое поведение, но стандартная функциональность работает с минимальным кодом. Если необходимы дополнительные настройки, можно использовать общие представления, вместо `ModelViewSet` или даже отдельные пользовательские представления.

Пример одного из видов, использовавшегося при написании сервиса приведен в приложении 2.

Маршрутизаторы (Routs)

И наконец, маршрутизаторы: они предоставляют верхний уровень API. Чтобы избежать создания бесконечных URL-адресов вида: «списки», «детали» и «изменить», маршрутизаторы DRF объединяют все URL-адреса, необходимые для данного вида в одну строку для каждого View.

Пример одного из видов, использовавшегося при написании сервиса приведен в приложении 3.

Пакеты и классы

Таким образом в результате анализа предметной области все файлы сервиса решено было разбит на пакеты, представленные на рисунке 12 пакеты

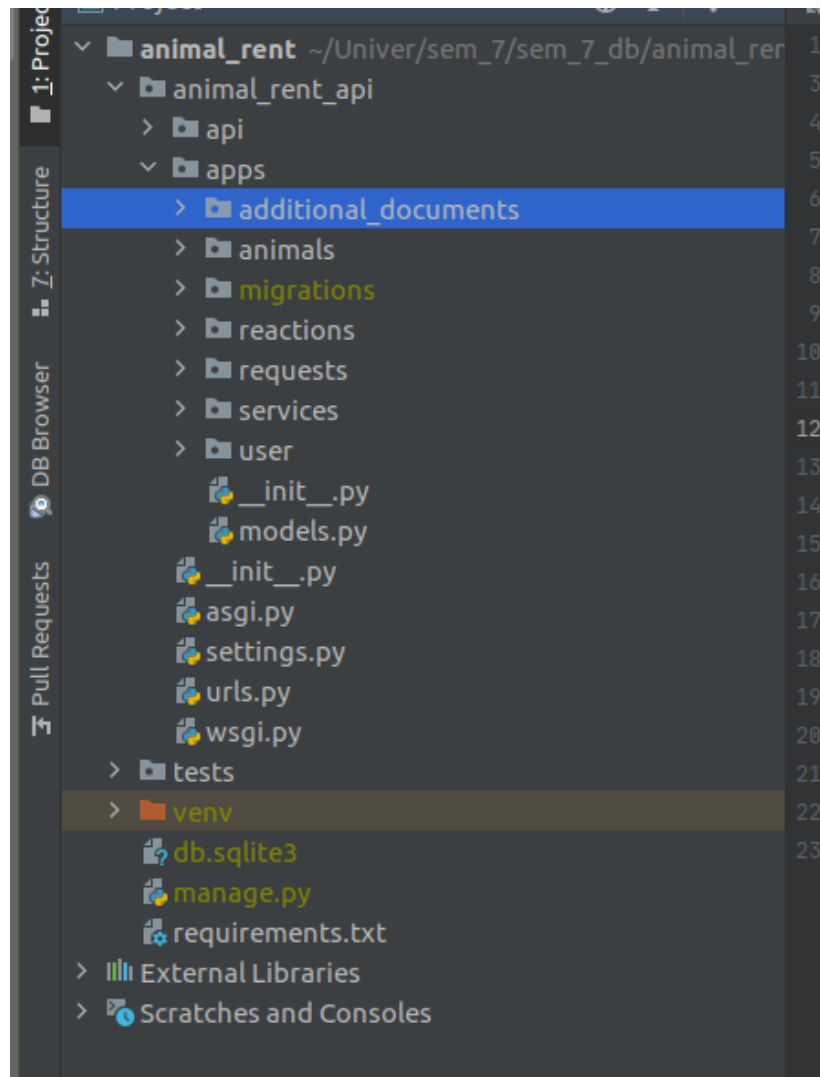


Рисунок 12 – Пакеты сервиса

Каждая таблица в базе данных представлена отдельным классом модели, для всех запросов существуют соответствующие классы представлений и сериализаторы. Для организации рассылки сообщений использовался smtp-сервер от gmail, который позволяет бесплатно отправлять до 1000 сообщений в месяц, а так же библиотека django.core.mail. В приложении 4 приведен код класса, с помощью которого в сервисе организована рассылка сообщений.

Таким образом классы сервиса отображены на схеме ниже (смотреть рисунок 13)

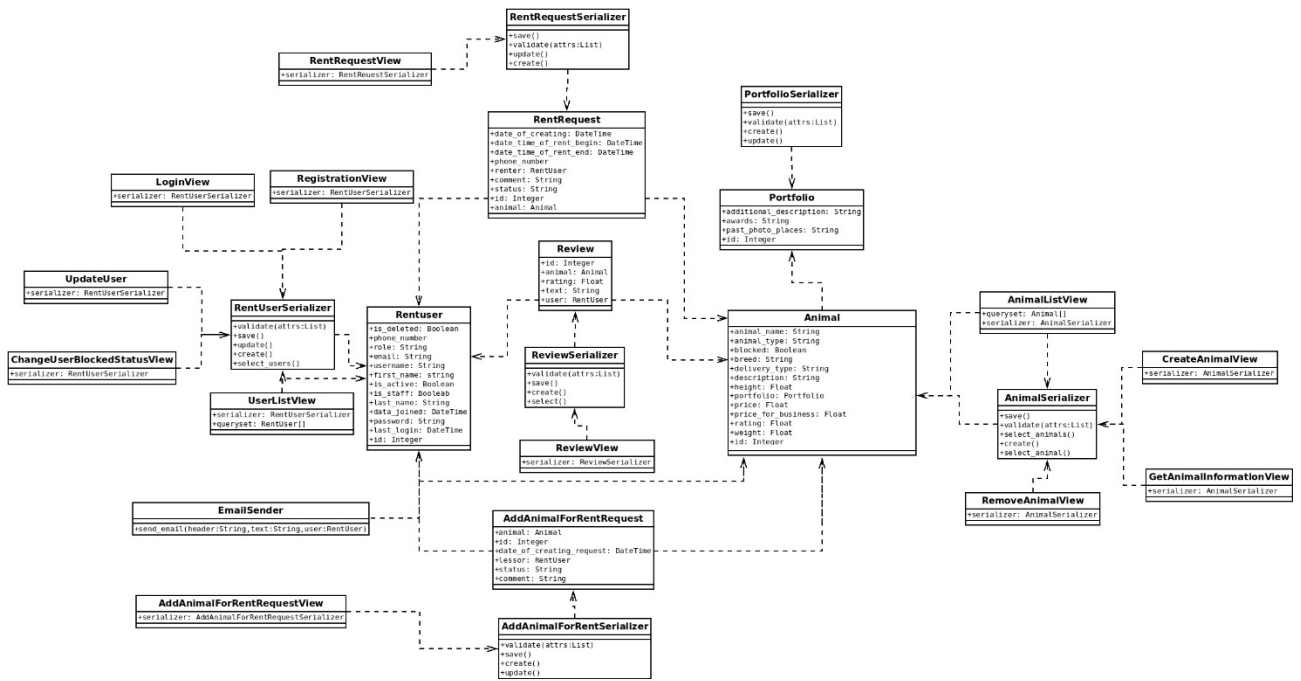


Рисунок 13 – классы сервиса

Тестирование

Модульное тестирование, иногда блочное тестирование или юнит-тестирование — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Преимущества:

- Поощрение изменений (Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений)

- Упрощение интеграции (Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом)

- Документирование кода (Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера)

- Отделение интерфейса от реализации (Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный мок-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе)

Поэтому для тестирования сервиса по аренде домашних животных широко использовались юнит тесты. Они были написаны для каждого модуля и

для каждого представления и проверяли как позитивные, так и негативные сценарии его использования. (Позитивное тестирование – это тестирование с применением сценариев, которые соответствуют нормальному (штатному, ожидаемому) поведению системы. Негативным называют тестирование, в рамках которого применяются сценарии, которые соответствуют внештатному поведению тестируемой системы. Это могут быть, например, исключительные ситуации или неверные данные.). Таким образом получившуюся схему пакетов и классов можно увидеть на рисунке 14

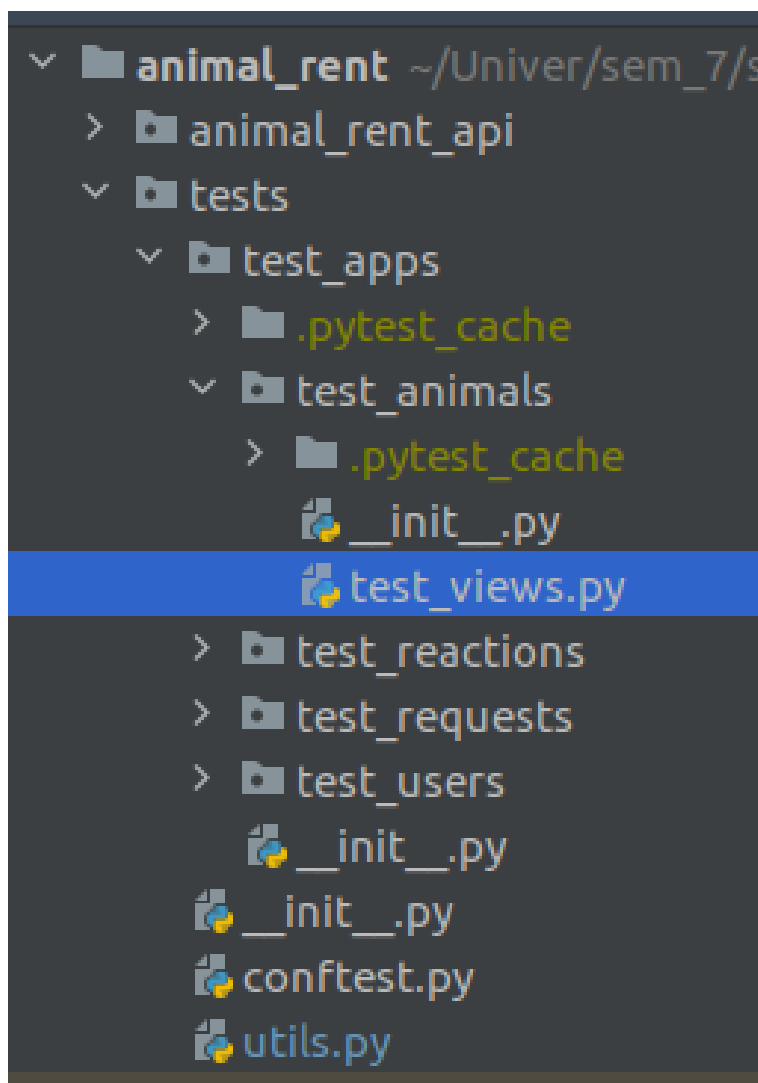


Рисунок 14 – Схема пакетов тестов сервиса

Так же была созданная отдельная база данных, которая использовалась в тестах и для каждого класса написаны специальные фабрики, позволяющие создавать объекты для тестирования с необходимыми в каждом конкретном случае параметрами. Пример такого класса можно увидеть в приложении 5.

Для написания фабрик была использована библиотека django-factory-boy,

которая позволяет не только создавать фабрики с минимальным количеством написанного кода, но также достаточно просто создавать подфабрики для вложенных классов.

Для написания самих юнит-тестов использовалась библиотека `pytest`. Данная библиотека имеет большое количество преимуществ:

- Независимость от API
- Подробный отчет. Сам вид отчета может изменяться (включая цвета) дополнительными модулями (о них будет позднее отдельно). Ну и вообще цветной отчет в консоли выглядит удобнее — красные `FAILED` видны сразу.
- Удобный `assert` (стандартный из Python). Не приходится держать в голове всю кучу различных `assert`'ов.
- Динамические фикстуры всех уровней, которые могут вызываться как автоматически, так и для конкретных тестов. Эта характеристика очень пригодилась для написания фикстур для различных типов пользователей.
- Дополнительные возможности фикстур (возвращаемое значение, финализаторы, область видимости, объект `request`, автоиспользование, вложенные фикстуры)
- Интеграция с фреймворком `django`, те многие функции, полезные в этом именно в этом фреймворке идут сразу “из коробки”
- Параметризация тестов, то есть запуск одного и того же теста с разными наборами параметров.
- Метки (`marks`), позволяющие пропустить любой тест, пометить тест, как падающий (и это его ожидаемое поведение, что полезно при разработке) или просто именовать набор тестов, чтобы можно было запускать только его по имени.

Методика использования разработанного программного средства

Данное программное средство можно использовать как отдельный продукт, так и как часть программного комплекса. Это является следствием того, что приложение является `restfull`-сервисом, таким образом к нему можно делать как отдельные запросы, так и достаточно легко интегрировать с `front` частью сервиса с помощью, например, языка программирования `javascript`.

Для того, чтобы воспользоваться сервисом необходимо:

1) Если вы собираетесь использовать этот сервис самостоятельно, рекомендую установить `Insomnia` или `Postman`, они значительно упрощают написание запросов.

2) Вам необходимо зарегистрироваться (если вы уже зарегистрированы, переходите сразу к шагу 3). Для регистрации рекомендую указывать свои настоящие телефон и `email`, т.к. телефон будет использоваться для связи с вами по вопросам аренды животного (если вы не укажете иной номер для связи), а на `email` будут приходить оповещения о изменении статуса ваших заявок. Так же учитывайте, что телефон вы сможете полностью изменить при необходимости, а вот изменить `email` пока не является возможным. Так же при отдельном использовании этого сервиса не забудьте очистить установленные им куки, если вы уже пользовались им раньше.

3) Вам необходимо ввести свой пароль и `email`, которые вы указывали при регистрации (Помните, что при необходимости вы можете сменить пароль). Так же при отдельном использовании этого сервиса не забудьте очистить установленные им куки, если вы уже пользовались им раньше. В случае, если вы получили сообщение о том, что вы заблокированы, обратитесь к вашему администратору, он сможет вам пояснить причину и разблокировать вас, если вы были заблокированы случайно.

4) Далее вы можете использовать сервис, например, для добавления для аренды своего животного, просмотра всех существующих, взятии в аренду, написания отзывов и так далее.

При использовании приложения необходимо учесть следующее:

- Добавить животное можно только существующих в приложении типов (если вы хотите добавить то, которое не существует, обратитесь к администратору с просьбой добавить новый тип животного)

- Собак, породы которых являются опасными в Республике Беларусь нельзя взять в аренду, и, как следствие сдать

- Любое животное можно арендовать не дольше, чем на 6 часов

- Вы можете оставить только отзыв на то животное, которое уже арендовали

- Блокировать и разблокировать пользователей может только администратор, при возникновении проблем обратитесь к нему.

Заключение

Из-за узкой специализации, слабой освоенности ниши и относительно небольшого количества компаний, занимающихся таким делом, а также множества других факторов, сложно спрогнозировать вероятную прибыль. Многое будет зависеть от размера города, эффективности рекламы, стоимости услуг и т. д.

Самые большие затраты при открытии дела будут на маркетинг, приобретение животных и аренду помещения. Также не стоит забывать, что потребуется человек, который будет ухаживать за животными. Для ведения деятельности нужен ветеринар, поэтому, если у вас нет такого образования, придется нанять для этого человека.

При наличии у вас диплома ветеринара и относительно небольшом количестве животных можно осуществлять деятельность самостоятельно. Это позволит значительно сэкономить на персонале. В любом случае аренда кошки или собаки – это дело затратное, но потенциально прибыльное.

Чтобы открыть подобное предприятие потребуется минимум 3500 рублей, львиная доля которых пойдет на рекламу. Остальное израсходуется на покупку животных, съем помещения, оформление документации, наем сотрудников и т. д. Если для работы приобретаются дорогие породистые животные, то стартап будет значительно больше.

Кошка напрокат в зависимости от города, региона, породы и цели приобретения обойдется приблизительно в 5-17 рублей за 6 часов. В крупном городе каждое животное можно сдавать минимум по 5-7 раз в месяц. Несложно посчитать, что доход составит ориентировочно 100-400 рублей в месяц. На содержание одного питомца уходит около 100 рублей в месяц. Таким образом, имея всего 5 кошек, ежемесячно можно получать прибыль порядка 1200 рублей.

Приблизительно такая же ситуация обстоит и с другими животными. Чаще всего приобретается собака напрокат или кошка. Экзотические животные пользуются меньшим спросом, но и цену на их аренду можно установить выше.

Таким образом реализованный сервис будет весьма востребован, хотя область его использования достаточно новая.

Список использованных источников

1. Форсье Д. Разработка веб-приложений на Python / Форсье Джефф, Биссекс Пол. - СПб: Символ-плюс, 2014-390 с.
2. Дронов, В.А. Django 3.0. Практика создания веб-сайтов на Python / В. А. Дронов - СПб: БХВ-Петербург , 2020-704 с.
3. Django [Электронный ресурс]. - Режим доступа :
<https://www.djangoproject.com/>
4. Django Rest Framework [Электронный ресурс]. - Режим доступа :
<https://www.django-rest-framework.org/>
5. Wanpaku land [Электронный ресурс]. - Режим доступа :
<https://www.wanpakuland.com/>

Приложения

Приложение 1

```
class AnswerAddAnimalForRentRequestSerializer(ModelSerializer):
    class Meta:
        model = AddAnimalForRentRequest
        fields = [
            'status',
            'text_comment'
        ]
        extra_kwargs = {
            'status': {
                'required': True,
            },
            'text_comment': {
                'required': False,
                'allow_blank': True,
                'allow_null': True
            },
        }

    def unlock_animal(self, animal):
        with connection.cursor() as cursor:
            cursor.execute(
                """UPDATE apps_animal SET blocked = %s WHERE id = %s """,
                (
                    False, animal.id
                )
            )

    def send_email(self, text, user):
        EmailSender.send_email(header=ANSWER_REQUEST_HEADER, text=text, user=user)

    def save(self):
        validated_data = self.validated_data
        comment = validated_data.get('text_comment')
        status = validated_data.get('status')
        if status in {Statuses.APPROVED, Statuses.REJECTED}:
            instance = super(AnswerAddAnimalForRentRequestSerializer, self).save()
            mail_text = comment if comment else
DEFAULT_MAIL_MESSAGES_FOR_ANSWER_REQUESTS[status]
            if status == Statuses.APPROVED:
                self.unlock_animal(instance.animal)
            self.send_email(mail_text, instance.lessor)
        else:
            raise serializers.ValidationError(
                'Can not react this way'
            )
```

Приложение 2

```
class AddAnimalForRentAnswerView(generics.UpdateAPIView):
    serializer_class = AnswerAddAnimalForRentRequestSerializer
    permission_classes = (permissions.IsAuthenticated, permissions.IsAdminUser)

    def get_object(self):
        try:
            return AddAnimalForRentRequest.objects.raw(
                f"SELECT apps_addanimalforrentrequest.id FROM
apps_addanimalforrentrequest WHERE
                (apps_addanimalforrentrequest.id = {self.request.parser_context['kwargs']
['id']})")
        except IndexError:
            raise Http404
```

Приложение 3

```
urlpatterns = [  
    path('answer/add_animal_for_rent/<int:id>', AddAnimalForRentAnswerView.as_view(),  
name='answer_add_animal_for_rent'),  
    path('get_animal_for_rent/', RentRequestCreateView.as_view(), name='animal_rent_request'),  
    path('answer/rent_request/<int:id>', RentRequestAnswerView.as_view(),  
name='answer_rent_request'),  
    path('rent_request/<int:id>', RentRequestRetrieveView.as_view(), name='retrieve_rent_request'),  
    path('rent_request/', RentRequestListView.as_view(), name='list_rent_request'),  
    path('add_animal_for_rent/<int:id>', AddAnimalForRentRetrieveView.as_view(),  
name='retrieve_add_animal_for_rent_request'),  
    path('add_animal_for_rent/', AddAnimalForRentListView.as_view(),  
name='list_add_animal_for_rent_request'),  
]
```

Приложение 4

```
class EmailSender:

    @staticmethod
    def send_email(header, text, user):
        if isinstance(user, RentUser):
            email = user.email
        else:
            email = user['email']

        send_mail(header, text, SITE_EMAIL, (email, ), fail_silently=False)

    @staticmethod
    def send_mails_to_many_users(header, text, users):
        users_mails = (user.email for user in users)

        send_mail(header, text, SITE_EMAIL, users_mails, fail_silently=False)
```

Приложение 5

```
class AnimalFactory(factory.django.DjangoModelFactory):
    animal_type = AnimalType.CAT
    breed = factory.Faker('pystr', max_chars=10)
    height = factory.Faker(
        'random_int', min=11, max=170
    )
    weight = factory.Faker(
        'random_int', min=1, max=100
    )
    delivery_type = DeliveryType.PICKUP
    rating = factory.Faker(
        'random_int', min=1, max=4
    )
    description = factory.Faker('pystr', max_chars=20)
    portfolio = factory.SubFactory(PortfolioFactory)
    price = factory.Faker(
        'random_int', min=11, max=170
    )
    price_for_business = factory.Faker(
        'random_int', min=11, max=170
    )
    blocked = False
    animal_name = factory.Faker('pystr', max_chars=10)

    class Meta:
        model = Animal
```