РЕФЕРАТ

Выпускная квалификационная работа содержит 56 страниц, 1 рисунок, и 7 источников.

ПРОГРАММНО-ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ, ЛОГИСТИКА, ВЕБ-ПРОГРАММИРОВАНИЕ

В выпускной квалификационной работе разработана и введена в эксплуатацию система онлайн-бронирования чартерных рейсов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время на рынке гражданской чартерной авиации широко используются системы автоматизированного бронирования билетов и оптимизации регулярных рейсов. В то же самое время в индустрии чартерных авиаперевозок до сих пор основными средствами организации рейсов остаются телефонные звонки и личные встречи.

Широкое использование систем онлайн-бронирования и автоматического планирования рейсов, с которой могли бы сотрудничать авиакомпании, приведёт к увеличению прозрачности схем ценообразования, повышению конкуренции между авиакомпаниями и, в конечном итоге, развитию рынка.

Целью этой выпускной квалификационной работы является задача разработки и введения в эксплуатацию продукта – системы онлайн-бронирования чартерных рейсов на основе системы, предложенной в работе Григорьева И.Н. [1]

ОСНОВНАЯ ЧАСТЬ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Архитектура серверного приложения

Проект написан на языке Python 3.6 [2] с использованием Django 1.10 [3] – фреймворка для веб-приложений. В основе этого фреймворка лежит шаблон проектирования MTV – Model-Template-View (Модель-Шаблон-Представление), а потому в разработке проекта были широко использованы все три аспекта этого шаблона проектирования. В этом разделе будет подробно описана схема использования моделей, шаблонов и представлений, а также диспетчера URL адресов и менеджера очереди заданий.

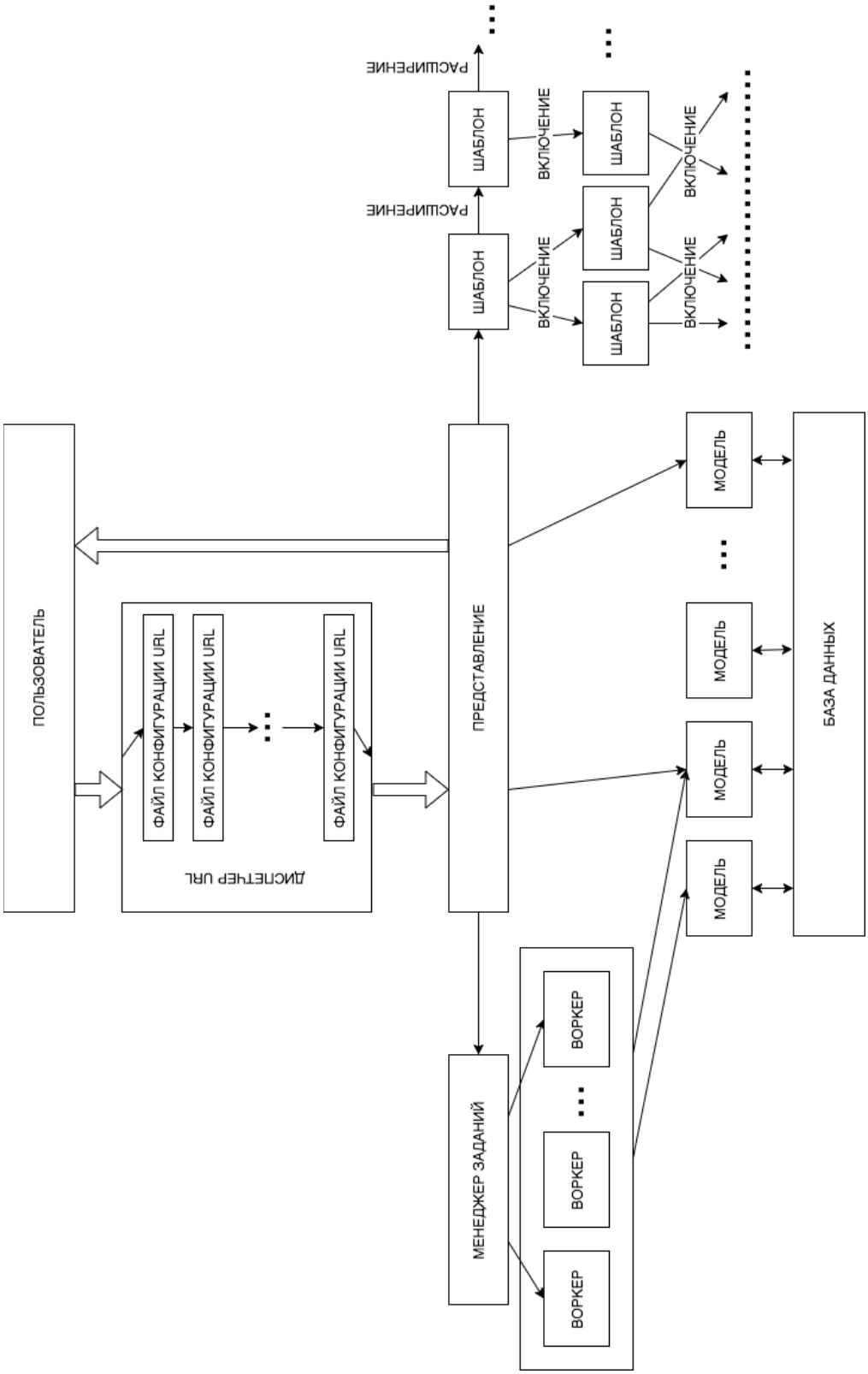

Рис. 1.1 Общая схема архитектуры серверного приложения

1.1.1 Модели

Модель (Model) – это компонент, отражающий актуальное состояние записей в соответствующей ему таблице базы данных [8] и предоставляющий интерфейс для изменения этих записей. Всего в рамках проекта была реализована 21 модель.

Рассмотрим разработку моделей на примере реализации модели `fleet.Aircraft`, предназначенной для работы с зарегистрированными в проекте самолётами. Исходный код этой модели представлен в приложении 1.

В Django используется технология ORM – Object Related Model, которая автоматически создаёт схему базы данных по описанной в классе модели. Так, в строках 2-60 описываются поля модели, по которым и будет сгенерирована схема таблицы для хранения самолётов, которая также описана в Приложении 1.

Например, в строках 2-8 описывается поле `reg`, в котором хранится регистрация (международный идентификатор) самолёта:

```
reg = models.CharField(
    "Registration",
    max_length=64,
    validators=[
      validators.RegexValidator(
        '^[a-zA-Z0-9-]+$'
      ),
    ]
)
```

Здесь указано, что поле `reg` – это строковое поле, ограниченное длиной в 64 символа, имеющее название `Registration` и удовлетворяющее регулярному выражению `^[a-zA-Z0-9-]+$`, то есть состоящему из символов латинского алфавита, цифр и знака дефиса. В сгенерированном SQL коде этому полю соответствует строка 7:

```
'reg' varchar(64) NOT NULL,
```

Помимо простых типов полей, вроде строк, чисел и меток времени, в моделях проекта широко используются поля, указывающие на объекты других моделей. Например, в строках 26-30

```
baseport = models.ForeignKey(
    'geography.Airport',
    blank=True, null=True,
    on_delete=models.PROTECT
)
```

описывается поле `baseport` (аэропорт базирования), значениями которого являются объекты модели `geography.Airport` (аэропорты). Соответствующий SQL код описан в строках 62-65:

```
CREATE INDEX 'fleet_aircraft_6240b757' ON
'fleet_aircraft' ('baseport_id');
ALTER TABLE 'fleet_aircraft'
  ADD CONSTRAINT
'fleet_aircraft_baseport_id_db50190c_fk_geography_airpo
rt_id'
  FOREIGN KEY ('baseport_id') REFERENCES
'geography_airport' ('id');
```

Кроме полей, в моделях также описываются методы, связанные с дополнительной обработкой данных при изменении записей (такие, как методы `save` и `clean` в строках 96-110 и 121-136 Приложения 1 соответственно), методы упрощающие обработку записей (например, методы `position` и `cover_image` в строках 112-119 и 154-162 соответственно) и методы, предназначенные для специфичного изменения записей (в данной модели не представлены).

1.1.2 Шаблоны

Шаблон (Template) – это компонент, представляющий способ генерации искомого ресурса (обычно HTML страницы). Всего в рамках проекта было реализовано 65 шаблонов.

В фреймворке Django шаблоны представляют собой текстовые файлы со специальными управляющими конструкциями. В качестве примера рассмотрим шаблон `flight_card.html`, который отвечает за генерацию блока с информацией о бронировании самолёта. Исходный код этого шаблона приведён в Приложении 2.

Исходный код этого шаблона представляет собой обычный код на языке разметки веб страниц HTML с дополнительными управляющими конструкциями, заключёнными в `{{…}}` и `{%…%}`.

Конструкции вида `{{…}}` указывают на то, что вместо них следует подставить значение вычисленного выражения.

Так, в строке 68

```
<span>{{ res.ac.type.name }}</span>
```

подстрока `{{ res.ac.type.name }}` будет заменена на значение `res.ac.type.name`. В данном контексте `res` – это объект поискового результата, который хранит в себе предложенный пользователю самолёт (поле `ac`), просчитанный маршрут (поле `route`) и дополнительную служебную информацию, то есть `res.ac` – это объект модели `Aircraft`, которая была рассмотрена выше. У этой модели есть поле `type` – указатель на объект модели `AircraftType`, представляющей из себя описания типов воздушных судов с сопутствующей информацией (например, Boeing 737-300 или Airbus A340-500). В этой модели среди прочих описано поле `name`, в котором содержится название типа самолёта. Таким образом, `{{ res.ac.type.name }}` будет заменено на название типа самолёта,

предложенного пользователю в поисковом результате (на скриншоте в Приложении 2 этот результат можно наблюдать в секции Aircraft).

Второй тип используемых управляющих конструкций – это конструкции вида `{%...%}`. Рассмотрим, например, строки 14-47:

```
{% for seg in res.route.segments %}
    ...
{% endfor %}
```

Здесь `res.route` – это просчитанный маршрут, а `res.route.segments` – список запланированных перелётов (сегментов). Весь код, расположенный между этими строками, будет добавлен к искомому ресурсу для всех элементов `res.route.segments`, то есть для каждого сегмента просчитанного маршрута (на скриншоте в Приложении 2 этот результат можно наблюдать в секции `Itinerary`).

Или, например, рассмотрим строки 198-200:

```
{% if res.ac.seats_img %}
   <img src="{{ res.ac.seats_img.url }}"/>
{% endif %}
```

Здесь `res.ac.seats_img` – это файл-изображение со схемой рассадки в самолёте, который может быть загружен, а может отсутствовать. В зависимости от того, загружен файл или отсутствует, res.ac.seats_img будет приведён к типу `Bool` (логический тип) как `True` (истина) или `False` (ложь) соответственно, и строка 199 будет добавлена к результату генерации только при условии, что `res.ac.seats_img` будет приведён к `True`, то есть файл-изображение `seats_img` для данного объекта модели был загружен в систему.

Также проект придерживается философии наследования шаблонов, которая выражена в использовании управляющих конструкций `{% include %}` и `{% extends %}`/`{% block %}`.

В строке 14 шаблона booking_request.html (отвечающего за предварительный просмотр бронирования), исходный код которого представлен в приложении 2, можно увидеть, как рассмотренный ранее шаблон flight_card.html включается в состав шаблона booking_request.html:

```
{% include 'newjetway/flight_card.html' with res=res %}
```

Содержимое flight_card.html было вынесено в отдельный шаблон, поскольку генерируемая им карточка с информацией о бронировании используется в нескольких шаблонах (не только на странице предварительного просмотра бронирования, но и, например, на страницах подтверждения бронирования и отслеживания статуса бронирования).

Шаблоны могут не только включать друг друга, но и расширять друг друга. Это необходимо, например, для сохранения консистентности дизайна сайта – поскольку страницы могут быть визуально похожи, предлагается сначала описать их общую часть и определить блоки, содержимое которых будет изменено при расширении.

Например, в первой строке шаблона booking_request.html

```
{% extends 'newjetway/base_normal.html' %}
```

указано, что он расширяет шаблон base_normal.html. То, каким образом он его расширяет, описано в управляющих конструкциях {% block %}/{% endblock %}. Так, например, в строках 12-18 шаблона booking_request.html указано, каким должно стать содержимое блока content:

```
{% block content %}
    <div class="section-nav"></div>
    {% include 'newjetway/quote_fsm.html' with
fsm_status="BOOKING_REQUEST" %}
```

```
    {% include 'newjetway/flight_card.html' with
res=res %}
    {% include 'newjetway/booking_footer.html' with
fixed="fixed" href=context.request_button onclick=""%}
    <div class="footer-placeholder"></div>
{% endblock %}
```

1.1.3 Представления

Представление (View) – это компонент, предназначенный для отображения данных модели пользователю. Всего в рамках проекта было реализовано 41 представление.

Рассмотрим в качестве примера разработку представления `booking_request`, отвечающего за генерацию страницы с предварительным просмотром пользовательского бронирования. Исходный код этого представления представлен в Приложении 3.

Функция `booking_request` принимает два параметра – `request` и `uuid`. Параметр `request` содержит в себе информацию о пользовательском запросе – адрес запроса (URL), метод запроса (GET/POST/прочее), параметры запроса, пользовательскую сессию, куки (cookie), IP пользователя и прочую информацию. Параметр `uuid` – это уникальный идентификатор поискового результата, полученный из адреса пользовательского запроса методом, описанным далее.

В строках 17-18

```
if Quote.objects.filter(uuid=uuid).first():
    return redirect('quote:status', uuid=uuid)
```

проверяется существование в системе уже оформленного пользовательского бронирования с указанным идентификатором. В случае, если бронирование существует, пользователь будет перенаправлен на страницу, отображающую статус этого бронирования.

В строке 19

```
res = get_object_or_404(SearchResult, uuid=uuid)
```

из базы данных запрашивается объект модели `SearchResult` с идентификатором `uuid`. В случае, если такого объекта не существует, будет возбуждено исключение `Http404` (404 – код возврата протокола HTTP, указывающий, что запрашиваемая страница не была найдена) и пользователю

в качестве результата будет возвращена служебная страница, указывающая на то, что запрашиваемая страница не была обнаружена.

В строке 20

```
context = {'request_button':reverse('search:contact_info', kwargs={'uuid':uuid}),}
```

создаётся вспомогательный словарь (ассоциативный массив), в котором для ключа `request_button` будет записан адрес страницы подтверждения бронирования.

Наконец, в строках 21-24

```
return render(request, 'newjetway/search/booking_request.html', {
        'res': prepare_flight_for_template(res),
        'context': context}
)
```

генерируется ответ пользователю на основе описанного ранее шаблона `booking_request.html`, в котором в качестве переменной `res` будет использоваться значение `prepare_flight_for_template(res)`, то есть обогащенный дополнительной служебной информацией поисковый результат, а в качестве переменной `context` – созданный в строке 20 вспомогательный словарь.

1.1.4 Диспетчер URL адресов

Для сопоставления запрашиваемых интернет-адресов и представлений используется механизм URL dispatcher (диспетчер URL адресов).

Рассмотрим пример такого сопоставления на примере запроса по адресу `search/result/26534cb3-ede8-4a95-99b9-d5e50fe4de69`.

В приложении 4 представлен фрагмент корневого файла конфигурации URL `jetway/urls.py`. Этот файл главным образом представляет собой список правил – пар из регулярных выражений [4] и представлений или указателей на другие файлы конфигурации URL. Так, вышеприведённый адрес удовлетворяет правилу в строке 8:

```
url(r'^search/', include(
  'search.urls',
  namespace='search'
)),
```

Это правило указывает, что подобные адреса необходимо обрабатывать согласно файлу конфигурации URL `search/urls.py`, отбрасывая при этом совпавший префикс. Таким образом, система будет пытаться найти соответствующее представление для адреса

```
result/26534cb3-ede8-4a95-99b9-d5e50fe4de69
```

в файле конфигурации URL `search/urls.py`. Такой адрес удовлетворяет правилу в строках 18-22:

```
url(
    '^result/(?P<uuid>[\w-]+)$',
    views.booking_request,
    name='booking_request',
),
```

Здесь `(?P<uuid>[\w-]+)` – это именованная группа `uuid`, которой соответствует ненулевое количество символов из набора латинских символов,

цифр, андерскора и дефиса, то есть в рассматриваемом нами примере именованной группе `uuid` будет сопоставлена строка

`26534cb3-ede8-4a95-99b9-d5e50fe4de69`.

Правило предписывает сопоставить такому адресу представление `booking_request`, которое было описано ранее, при этом, поскольку в правиле были описаны именованные группы, в это представление будет передан параметр с именем, совпадающим с именем именованной группы, и со значением, равным сопоставленной этой группе строке, то есть с параметром `uuid=26534cb3-ede8-4a95-99b9-d5e50fe4de69`.

1.1.5 Менеджер очереди заданий

Архитектура проекта также построена на широком использовании менеджера очереди заданий Celery [5] – механизма распределения вычислений между воркерами (workers). Воркеры – это сервера, расположенные в одной сети с главным (master) сервером и выполняющие полученные от него задания.

В качестве примера использования этой технологии рассмотрим реализацию модели `RosterFile` – модели, сохраняющей и обрабатывающей загруженные ростеры авиакомпаний (ежедневные расписания рейсов самолетов, подробнее структура ростеров изложена в работе Григорьева И.Н. [1]). Исходный код этой модели приведён в приложении 5.

Обработка ростера – это тяжёлый вычислительный процесс, время исполнения которого может исчисляться минутами, а потому он не может исполняться синхронно.

В строках 31-37

```
STATUS_CHOICES = (
    ('PD', 'Pending'),
    ('RU', 'Processing'),
    ('OK', 'OK'),
    ('FL', 'Failed'),
    ('IE', 'Internal error'),
)
```

описываются возможные состояния процесса обработки ростера:

* PD – Pending – «Обработка ещё не началась»
* RU – Processing – «Обработка в процессе»
* OK – OK – «Обработка завершена успешно»
* FL – Failed – «Обработка завершена с ошибкой»
* IE – Internal error – «Произошла непредвиденная ошибка»

В строках 23-24 с помощью декоратора `@shared_task` объявляется задача `roster_file_proceed`, запускающая обработку ростера:

```
@shared_task
def roster_file_proceed(roster_file):
```

Эта задача ставится в очередь при сохранении ростера в строках 33-34, если его текущим состоянием является PD – «Обработка ещё не началась»:

```
def save(self, **kwargs):
    super().save(**kwargs)
    if self.status == 'PD':
        roster_file_proceed.delay(self)
```

После того, как Celery назначает (assign) задачу какому-либо воркеру, состояние переводится в RU – «Обработка в процессе»:

```
roster_file.status = 'RU'
```

и запускается непосредственно процесс обработки:

```
try:
    driver()(
        roster_file.airline, roster_file.roster.path
    )
```

Если процесс завершается с ожидаемой ошибкой (например, ростер составлен в неверном формате), то во время обработки будет вызвано исключение типа `RosterDriverException`, которое будет перехвачено, состояние ростера будет изменено на FL – «Обработка завершена с ошибкой», а в поле `comment` будет записано сообщение об ошибке:

```
except RosterDriverException as E:
    roster_file.status = 'FL'
    roster_file.comment = str(E)
    roster_file.save()
    return
```

Если же произошла непредвиденная ошибка, то состояние ростера будет изменено на IE – «Произошла непредвиденная ошибка», а исключение будет возбуждено заново и подробная информация о произошедшей ошибке будет передана администратору (подробнее об этом рассказано в разделе «Развёртка и сопровождение продукта»):

```
    except:
        roster_file.status = 'IE'
        roster_file.comment = "Internal error. Please
contact administrator."
        roster_file.save()
        raise
```

Наконец, если обработка ростера прошла успешно, его состояние будет изменено на OK – «Обработка завершена успешно»:

```
    roster_file.status = 'OK'
```

1.2. Реализация интернет интерфейса

Реализация интернет интерфейса заключена в решении двух задач:

* Вёрстка и дизайн веб-страниц – каскадные таблицы стилей (CSS)
* Активные скрипты – JavaScript-файлы

1.2.1 Вёрстка и дизайн веб-страниц

Реализация вёрстки и дизайна веб-страниц заключается в написании файлов каскадных таблиц стилей – CSS. Файл каскадной таблицы стилей – это набор правил (селекторов) и соответствующих им наборам значений атрибутов – свойств элементов веб-страницы, определяющих их поведение и внешний вид.

Практика написания CSS-файлов вручную была признана порочной в силу следующих причин:

* Отсутствие мультиплексирования в протоколе HTTP 1.1 предполагает открытие нового соединения с сервером для скачивания каждого файла, поэтому суммарное время загрузки страницы сильно увеличивается вместе с ростом количества необходимых CSS-файлов. Альтернатива – хранение CSS-правил в малом количестве файлов – ведёт к уменьшению модульности проекта и усложнению его поддержки

* Отсутствие встроенной системы наследования и поддержки констант. Например, руководство по стилю проекта предполагает использование цвета `#b3ffff` (оттенок цвета «светлый циан») в качестве акцента на активных элементах управления страницей. В связи с этим цвет должен присутствовать в большом количестве CSS-правил, и его изменение в руководстве по стилю предполагало бы изменение в разных местах разных CSS-файлов, причём проверка консистентности этих изменений была бы затруднительна.

* Человекочитаемый CSS-код значительно больше по объёму, чем сжатый – без лишних пробелов, переводов строк и прочего.

В связи с вышеуказанными проблемами было принято решение для генерации CSS-файлов использовать препроцессор SASS.

SASS [6] – это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS кода и упрощения файлов каскадных таблиц стилей.

Например, следующий SASS код:

```
 =clearfix
   &:after
     content: ""
     display: table
     clear: both
 .clr
   +clearfix
 .section-partners ul
   margin-top: 20px
   +clearfix
   display: table
   li
     float: left
     height: 125px
```

будет скомпилирован в следующий CSS-код (в человекочитаемом виде):

```
.clr:after {
    content: "";
    display: table;
    clear: both;
}
.section-partners ul {
    margin-top: 20px;
    display: table;
}
.section-partners ul:after {
    content: "";
    display: table;
    clear: both;
```

```
}
.section-partners ul li {
    float: left;
    height: 125px;
}
```

Вышеприведённый пример показывает, как с помощью SASS можно избавиться от избыточного повторения одинаковых атрибутов (`content`, `display` и `clear`).

1.2.2 Активные скрипты

Для реализации логики на веб-страницах (реагирование на нажатия активных элементов, отрисовка полученных результатов поиска и прочее) используются скрипты на языке JavaScript.

В проекте активно используются следующие JavaScript библиотеки:

* jQuery – библиотека, фокусирующаяся на удобном взаимодействии JavaScript и HTML

* jQuery UI – библиотека для упрощения разработки пользовательского интерфейса

* notifyjs – библиотека для показа пользователю всплывающих уведомлений

* jssor-slider – библиотека для генерации «карусели» изображений – активного элемента, позволяющего пролистывать изображения в пределах одного участка веб-страницы

* jquery-form – библиотека для конструирования асинхронных запросов на основе заполненных форм

* fontfaceobserver – библиотека для загрузки шрифтов

* handlebars – библиотека для генерации HTML кода на основе шаблонов

Для управления всеми подключаемыми JavaScript библиотеками в проекте используется менеджер пакетов Bower [7]. Список всех зависимостей хранится в настройках проекта и выглядит следующим образом:

```
...
BOWER_INSTALLED_APPS = [
    'handlebars#4.0.5',
    'jquery#3.1.0',
...
```

При инициализации Bower скачает и установит все перечисленные библиотеки с указанными версиями, а также все их зависимости.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Описание серверного приложения

Серверное приложение разбито на 13 смысловых частей:

* `account` – средства для управления аккаунтом пользователя: регистрация, изменение пароля, просмотр истории заказов и прочее. Не включает в себя модели.
* `common_pages` – статические страницы (например, заглавная страница, страница о проекте, служебные страницы 404 – «Документ не найден» и 403 – «Доступ запрещён». Не включает в себя модели.
* `fake_email_backend` – симулятор почтового сервера. Избавляет от необходимости посылать настоящие электронные сообщения при разработке и тестировании приложения. Включает в себя две модели:
    - `Email` – электронное письмо
    - `Attachment` – вложение в электронное письмо
* `fleet` – управление зарегистрированными в системе самолётами. Включает в себя 5 моделей:
    - `Airline` – авиакомпания
    - `Aircraft` – самолёт
    - `AircraftImage` – фотография самолёта
    - `AircraftType` – тип самолёта
    - `AircraftTypeImage` – общие фотографии самолётов данного типа
* `geography` – база географических объектов. Включает в себя три модели:
    - `Country` – страна
    - `City` – город
    - `Airport` – аэропорт
* `quotes` – управление пользовательскими бронированиями. Включает в себя одну модель:

    - `Quote` – пользовательское бронирование

* `schedule` – управление базой расписаний и обработка ростеров. Включает в себя две модели:
    - `ScheduleAtom` – единица расписания. Информация о доступности конкретного самолёта или множества самолётов в данном аэропорту или множестве аэропортов в конкретный промежуток времени. Подробнее об атомарной единице расписания рассказано в работе Григорьева И.Н. [1])
    - `RosterFile` – загруженный ростер
* `search` – обработка пользовательских поисковых запросов, просчёт маршрутов. Включает в себя три модели:
    - `SearchRequest` – пользовательский поисковый запрос
    - `SearchResult` – ответы на поисковые запросы
    - `MagicConstant` – аппроксимационные константы, необходимые для расчёта продолжительности и стоимости полёта
* `sms_confirmation` – реализация проверки указанного пользователем телефонного номера. Включает в себя одну модель:
    - `SMSAuthPair` – информация о высланном пользователю коде подтверждения
* `staffinfo` – информация об участниках проекта. Включает в себя 4 модели:
    - `Group` – структурное подразделение (например, «команда разработки», «внешние консультанты»)
    - `Member` – участник проекта
    - `ExternalSite` – виды ссылок на внешние ресурсы, принадлежащие участнику проекта (например, страницы участника в социальных сетях «ВКонтакте», «LinkedIn»)

    o `ExternalLink` – ссылка на внешний ресурс, принадлежащий участнику проекта

* `tools` – набор вспомогательных инструментов, которые используются в остальных частях проекта. Не включает в себя модели.
* `jetway_style` – генерация таблиц каскадных стилей (CSS) с помощью препроцессора SASS. Подробнее об этом написано в разделе «Вёрстка и дизайн веб-страниц».
* `common-static` – набор статических файлов для показа в браузере (например: логотип проекта, гарнитуры, javascript-файлы).

2.2 Развёртка и сопровождение продукта

2.2.1 Виртуальное окружение

Виртуальное окружение - это изолированное окружение среды, которое позволяет использовать определенные версии приложений и библиотек.

Использование виртуального окружения при разработке и на эксплуатационных серверах преследует следующие цели:

* Изоляция. Проект защищён от конфликтов библиотек и их версий с системным окружением.

* Безопасность. Реализованное на серверах разграничение прав не позволяет пользователю, от имени которого запущен веб-сервер, устанавливать глобально какие-либо пакеты или как-то иначе влиять на системное окружение

* Удобство. При работе над несколькими проектами возникает необходимость в инструменте, способном быстро переключать установленные библиотеки и версии интерпретатора Python

Все используемые в проекте библиотеки описаны в специальном текстовом файле и могут быть установлены автоматически с помощью утилиты pip.

2.2.2 Система миграций

Миграции – это инструмент переноса изменений в моделях (добавление поля, удаление модели и т.д.) на схему базы данных.

Файлы миграций представляют собой описания изменений в моделях, а также перечисление зависимостей – миграций, которые должны быть удовлетворены ранее.

Рассмотрим пример миграционного файла:

```
class Migration(migrations.Migration):
    dependencies = [
        ('fleet', '0008_airline_roster_driver'),
    ]
    operations = [
        migrations.AddField(
          model_name='aircraft',
          name='virtual',
          field=models.BooleanField(default=False),
        ),
    ]
```

Выше описана миграция, которая зависит от миграции с именем `0008_airline_roster_driver` и описывает добавление к модели `Aircraft` логического поля `virtual` со значением по умолчанию `False`.

При развёртке новой версии приложения на сервере все неудовлетворённые миграции будут выполнены и схема базы данных будет в консистентном состоянии по отношению к описаниям моделей.

2.2.3 Конфигурация эксплуатационного сервера

Проект на сервере запускается в виртуальном окружении под управлением веб-сервера Gunicorn с нативной поддержкой приложений на Django. Этот веб-сервер проксируется веб-сервером nginx, который добавляет поддержку защищенных соединений (HTTPS), сжатие ответов сервера и быстрое обслуживание неизменяемых запросов (запросы статических файлов).

Контроль над работой сервера Gunicorn выполняет система контроля процессов supervisorctl.

Также на сервере проект устанавливает подключение с программным комплексом Sentry – системой логирования и учёта необработанных исключений, которая в электронном письме высылает администратору подробную информацию о произошедшем программном сбое.

Проект предполагает следующий автоматический алгоритм развёртки:

* Скачивание актуальной версии проекта
* Установка неудовлетворённых зависимостей в виртуальное окружение
* Инициализация Bower – установка требуемых JavaScript библиотек
* Сбор всех статических файлов проекта в одну директорию для их быстрого обслуживания веб-сервером nginx
* Удовлетворение новых миграций
* Перезапуск веб-сервера Gunicorn

2.3 Результаты внедрения

Разработка программного комплекса велась командой из четырех программистов, студентов МАИ, МГУ и НИУ ВШЭ. Моей задачей была поставлена реализация и поддержка серверного приложения на основе структуры, предложенной в работе Григорьева И.Н. [1], реализация базы для разработки вёрстки и дизайна клиентского интерфейса моими коллегами и администрирование проекта и смежных с ним программных продуктов.

На основе предложенной системы был запущен веб-сервис flyJetway (https://flyjetway.com), позволяющий пользователям бронировать самолеты более чем 20 авиакомпаний с суммарным флотом свыше 100 бортов в странах Европейского Союза, за 6 месяцев с момента запуска было обработано более сотни заявок на чартерные рейсы.

Проект был отмечен несколькими премиями и грантами международных организаций, в частности разработка велась при поддержке Государственного агентства по туризму Республики Португалия и мэрии г. Лиссабона.

В рамках гранта Европейской Комиссии проект также был представлен на крупнейшей международной выставке в сфере корпоративного туризма IMEX-2017 в г. Франкфурт-на-Майне, Германия.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы было разработано, реализовано и введено в эксплуатацию программно-информационное обеспечение системы организации чартерных рейсов.

Общий объём кода проекта составил 12924 строки на языке программирования Python, 8570 строк кода на языке описания шаблонов, 6469 строк кода на мета-языке SASS и 610 строк кода на языке программирования JavaScript.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Григорьев И.Н. Архитектура и алгоритмы системы организации чартерных рейсов // ВКР Бакалавра – М.: МАИ, 2017.
2. https://docs.python.org/3.6/
3. https://docs.djangoproject.com/en/1.10/
4. https://docs.python.org/3.6/library/re.html
5. http://docs.celeryproject.org/en/latest/index.html
6. http://sass-lang.com/documentation/file.SASS_REFERENCE.html
7. https://bower.io/
8. К. Дж. Дейт. Введение в системы баз данных – 7-е изд. – М.: Вильямс, 2001.

ПРИЛОЖЕНИЯ

Приложение 1

Исходный код модели `fleet.Aircraft`

```
class Aircraft(models.Model):
    reg = models.CharField(
        "Registration",
        max_length=64,
        validators=[
          validators.RegexValidator('^[a-zA-Z0-9-]+$')
        ]
    )
    type = models.ForeignKey(
        'AircraftType',
        on_delete=models.PROTECT,
        blank=True, null=True
    )
    airline = models.ForeignKey('Airline',
on_delete=models.PROTECT)
    etops = models.IntegerField('ETOPS', default=0)
    capacity = models.IntegerField(
        'Capacity',
        blank=True, null=True,
        validators=[validators.MinValueValidator(0)]
    )
    range = models.IntegerField('Range', blank=True, null=True)
    baseport = models.ForeignKey(
        'geography.Airport',
        blank=True, null=True,
        on_delete=models.PROTECT
    )
    comment = models.TextField(blank=True, null=True)
    cph = models.FloatField("Cost per hour", blank=True,
null=True)
    rwy = models.FloatField("Runway", blank=True, null=True)
    speed = models.FloatField("Speed", blank=True, null=True)
    seats_f = models.IntegerField(
        'F Seats', default=0,
validators=[validators.MinValueValidator(0)]
    )
```

```
34.     seats_c = models.IntegerField(
35.         'C Seats', default=0,
    validators=[validators.MinValueValidator(0)]
36.     )
37.     seats_y = models.IntegerField(
38.         'Y Seats', default=0,
    validators=[validators.MinValueValidator(0)]
39.     )
40.     seats_img = models.ImageField(
41.         'Seats image', upload_to="aircrafts_seats", blank=True,
    null=True
42.     )
43.     anet_link = models.CharField(
44.         'Airliners.net link', max_length=200, blank=True,
    null=True
45.     )
46.     airfleets_link = models.CharField(
47.         'Airfleets link', max_length=200, blank=True, null=True
48.     )
49.     mfd = models.DateField("Manufacture date", null=True,
    blank=True)
50.     repr = models.CharField(max_length=100, editable=False)
51.     use_pps = models.BooleanField(default=False)
52.     pps_name = models.CharField(
53.         default=None, blank=True, null=True, max_length=100
54.     )
55.     pps_engine = models.CharField(
56.         default='', blank=True, null=True, max_length=100
57.     )
58.     suspended = models.BooleanField(default=False)
59.     virtual = models.BooleanField(default=False)
60.     rfh = models.FloatField('Reposition flight hour', blank=True,
    null=True)
61.     floating_fleet = models.BooleanField('Floating fleet',
    default=False)
62.     executive = models.BooleanField('Executive', default=False)
63.
64.     class Meta:
```

```
65.          ordering = ['reg']
66.          verbose_name_plural = 'Aircraft'
67.
68.      def save(self, **kwargs):
69.          self.repr = self.repr_func()
70.          if self.capacity is None:
71.              self.capacity = self.type.capacity
72.          if self.range is None:
73.              self.range = self.type.range
74.          if self.baseport is None:
75.              self.baseport = self.airline.baseport
76.          if self.cph is None:
77.              self.cph = self.type.cph
78.          if self.rwy is None:
79.              self.rwy = self.type.rwy
80.          if self.speed is None:
81.              self.speed = self.type.speed
82.          super().save(**kwargs)
83.
84.  def position(self, date):
85.      if not self.suspended:
86.          return self.scheduleatom_set.filter(
87.              date=date,
88.              state=0,
89.          ).first()
90.      else:
91.          return None
92.
93.  def clean(self):
94.      errors = {}
95.      if self.seats_f or self.seats_c or self.seats_y:
96.          if self.capacity != self.seats_f + self.seats_c +
    self.seats_y:
97.              errors['seats_f'] = 'F+C+Y ≠ PAX'
98.              errors['seats_c'] = 'F+C+Y ≠ PAX'
99.              errors['seats_y'] = 'F+C+Y ≠ PAX'
100.     if self.use_pps:
101.         if not self.pps_name:
```

```
102.             errors['pps_name'] = 'You should specify this field '\
103.                                  'or disable pps'
104.         if not self.pps_engine:
105.             errors['pps_engine'] = 'You should specify this field
    '\
106.                                    ' or disable pps'
107.     if errors:
108.         raise ValidationError(errors)
109.
110. def __str__(self):
111.     return self.repr
112.
113. def repr_func(self):
114.     if self.type is not None:
115.         ret = "%s/%s -- %s" % (self.reg, self.type.code,
    self.airline)
116.     else:
117.         ret = "%s/ -- %s" % (self.reg, self.airline)
118.     if self.virtual:
119.         ret = "[V] " + ret
120.     return ret
121.
122. @classmethod
123. def get(cls, reg):
124.     return cls.objects.get(reg=reg)
125.
126. def cover_image(self):
127.     ret = self.type.cover_image()
128.     if ret:
129.         return ret
130.     ret =
    self.images_set.filter(public=True).order_by('order').first()
131.     if ret:
132.         return ret.img
133.     else:
134.         return None
135.
136. def specific_images(self):
```

```
137.     return self.images_set.order_by('order')
138.
139. def images(self):
140.     return list(self.type.images()) +
141.            list(self.specific_images().filter(public=True))
142.
143. @property
144. def capacity_extra(self):
145.     if (self.seats_f+self.seats_c+self.seats_y):
146.         res = []
147.         if self.seats_f:
148.             res.append(str(self.seats_f)+' First')
149.         if self.seats_c:
150.             res.append(str(self.seats_c)+' Business')
151.         if self.seats_y:
152.             res.append(str(self.seats_y)+' Economy')
153.         seats = ", ".join(res)
154.         if len(res) > 1:
155.             seats = str(self.capacity)+': '+seats
156.         return seats
157.     return str(self.capacity)
```

Сгенерированный по ней SQL код

```
1.  BEGIN;
2.  --
3.  -- Create model Aircraft
4.  --
5.  CREATE TABLE `fleet_aircraft` (
6.    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
7.    `reg` varchar(64) NOT NULL,
8.    `etops` integer NOT NULL,
9.    `capacity` integer NULL,
10.   `range` integer NULL,
11.   `comment` longtext NULL,
12.   `cph` double precision NULL,
13.   `rwy` double precision NULL,
14.   `speed` double precision NULL,
15.   `seats_f` integer NOT NULL,
```

```
    `seats_c` integer NOT NULL,
    `seats_y` integer NOT NULL,
    `seats_img` varchar(100) NULL,
    `anet_link` varchar(200) NULL,
    `airfleets_link` varchar(200) NULL,
    `mfd` date NULL,
    `repr` varchar(100) NOT NULL,
    `use_pps` bool NOT NULL,
    `pps_name` varchar(100) NULL,
    `pps_engine` varchar(100) NULL,
    `suspended` bool NOT NULL,
    `virtual` bool NOT NULL,
    `rfh` double precision NULL,
    `floating_fleet` bool NOT NULL,
    `executive` bool NOT NULL
);
--
-- Add field airline to aircraft
--
ALTER TABLE `fleet_aircraft` ADD COLUMN `airline_id` integer NOT
NULL;
ALTER TABLE `fleet_aircraft` ALTER COLUMN `airline_id` DROP
DEFAULT;
--
-- Add field baseport to aircraft
--
ALTER TABLE `fleet_aircraft` ADD COLUMN `baseport_id` integer
NULL;
ALTER TABLE `fleet_aircraft` ALTER COLUMN `baseport_id` DROP
DEFAULT;
--
-- Add field type to aircraft
--
ALTER TABLE `fleet_aircraft` ADD COLUMN `type_id` integer NULL;
ALTER TABLE `fleet_aircraft` ALTER COLUMN `type_id` DROP DEFAULT;

ALTER TABLE `fleet_aircraftimage`
```

```
    ADD CONSTRAINT
`fleet_aircraftimage_ac_id_a98556d0_fk_fleet_aircraft_id`
    FOREIGN KEY (`ac_id`) REFERENCES `fleet_aircraft` (`id`);
CREATE INDEX `fleet_aircraft_6d121bba` ON `fleet_aircraft`
(`airline_id`);
ALTER TABLE `fleet_aircraft`
    ADD CONSTRAINT
`fleet_aircraft_airline_id_d9cf4e59_fk_fleet_airline_id`
    FOREIGN KEY (`airline_id`) REFERENCES `fleet_airline` (`id`);
CREATE INDEX `fleet_aircraft_6240b757` ON `fleet_aircraft`
(`baseport_id`);
ALTER TABLE `fleet_aircraft`
    ADD CONSTRAINT
`fleet_aircraft_baseport_id_db50190c_fk_geography_airport_id`
    FOREIGN KEY (`baseport_id`) REFERENCES `geography_airport`
(`id`);
CREATE INDEX `fleet_aircraft_94757cae` ON `fleet_aircraft`
(`type_id`);
ALTER TABLE `fleet_aircraft`
    ADD CONSTRAINT
`fleet_aircraft_type_id_d0dc71d6_fk_fleet_aircrafttype_id`
    FOREIGN KEY (`type_id`) REFERENCES `fleet_aircrafttype` (`id`);
COMMIT;
```

Приложение 2

Исходный код шаблона `flight_card.html`

```
<div class="section booking itinerary">
     <div class="content"><!--
     --><h2>Itinerary</h2>
         <div class="legs">
             <table>
                 <tr class="labels">
                     <td> </td>
                     <td> From </td>
                     <td> To </td>
                     <td> Date </td>
                     <td> Depart </td>
                     <td> Duration </td>
                 </tr>
                 {% for seg in res.route.segments %}
                 {% if not seg.is_reposition or
res.show_repositions %}
                 <tr>
                     <td class="labels leg-label"> Leg
{{ seg.leg_number }} </td>
                     <td>
                         {% with seg.orig as airport %}
                             <span>
                                 {{ airport.iata }}/
{{ airport.icao }}:
                                 {{ airport.name }}<br/>
                                 <small>{{ airport.city }}</small>
                             </span>
                         {% endwith %}
                     </td>
                     <td>
                         {% with seg.dest as airport %}
                             <span>
                                 {{ airport.iata }}/
{{ airport.icao }}:
                                 {{ airport.name }}<br/>
                                 <small>{{ airport.city }}</small>
```

```
33.                                 </span>
34.                             {% endwith %}
35.                         </td>
36.                         <td>
37.                             <span>{{ seg.flight_date|date:"M d, Y" }}
    </span>
38.                         </td>
39.                         <td>
40.                             <span>{{ seg.flight_time|time:"H:i" }}</
    span>
41.                         </td>
42.                         <td>
43.                             <span>{{ seg.time_duration }}</span>
44.                         </td>
45.                     </tr>
46.                     {% endif %}
47.                     {% endfor %}
48.                 </table>
49.             </div>
50.         </div>
51.     </div>
52.
53.     <div class="section booking aircraft">
54.         <div class="content ac_info">
55.             <h2>Aircraft</h2>
56.             <div class="blob">
57.                 <div class="logo">
58.                     <!--suppress HtmlUnknownTarget -->
59.                     <div class="viewbox">
60.                         <img class="coverimage"
    src="{{ res.ac.cover_image }}">
61.                         <i> </i>
62.                         <div> Images may not correspond with reality
    </div>
63.                     </div>
64.                 </div>
65.                 <div class="info">
66.                     <div class="strow">
```

```
                        <div class="actypecell cell">
                            <span>{{ res.ac.type.name }}</span>
                        </div>
                    </div>
                    <div class="strow">
                        <div class="paxcell cell">
                            <label>Seats/Class</label>
                            <span>{{ res.ac.capacity_extra }}</span>
                        </div>
                    </div>
                    <div class="strow">
                        <div class="manufacturedceil cell">
                            <label>Manufactured</label>
                            <span>{{ res.ac.mfd|date:"Y" }} </
    span>
                        </div>
                    </div>
                    <div class="strow">
                        <div class="speedcell cell">
                            <label>Speed</label>
                            <span>
                                {{ res.ac.speed|floatformat:"0" }}
    km/h 
                            </span>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="section booking price">
        <div class="content"><!--
       --><h2>Price</h2>
            <div class="price-info">
                {% load humanize %}
                <div class="label"> Preliminary Price: </div>
                <div class="price-val">{{ res.cost|intcomma }}
    <i>EUR</i> </div>
```

```
102.          </div>
103.      </div>
104.  </div>
105.  <div class="section booking images">
106.      <div class="content">
107.          <h2>Images</h2>
108.          <div class="pages slide">
109.            <div
110.                class="pagehead selected"
111.                id="side-page-photos" page="page-photos"
112.            >Photos</div><!--
113.           --><div
114.                  class="pagehead"
115.                  id="side-page-seatmap"
116.                  page="page-seatmap"
117.           >Seatmap</div><!--
118.           -->{% if res.route.get_google_maps_link %}<!--
119.           --><div
120.                  class="pagehead"
121.                  id="side-hide"
122.                  page="page-map"
123.          >Map</div>
124.          {% endif %}
125.          </div>
126.          <div class="page" id="page-photos">
127.              <div class="gallerycell cell">
128.                  <div id="sidepanel-gallery" class="gallery-
    spinner">
129.                      <div u="slides">
130.                          {% for img in res.ac.images %}
131.                              <div>
132.                                  <img u="image"
    src="{{ img.img.url }}"/>
133.                              </div>
134.                          {% endfor %}
135.                      </div>
136.                      <div
137.                          u="navigator"
```

```
                            class="jssorb09"
                            style="top: 16px; right: 10px;"
                        >
                            <div u="prototype"></div>
                        </div>
                        <span u="arrowleft" class="jssora14l">
                          <i></i>
                        </span>
                        <span u="arrowright" class="jssora14r">
                          <i></i>
                        </span>
                    </div>
                </div>
            </div>
            <div class="page" id="page-seatmap" style="display:
    none">
                {% if res.ac.seats_img %}
                    <img src="{{ res.ac.seats_img.url }}"/>
                {% endif %}
            </div>
            {% if res.route.get_google_maps_link %}
            <div class="page" id="page-map" style="display: none">
                <img class="map"
    src="{{ res.route.get_google_maps_link }}"/>
            </div>
            {% endif %}
        </div>
    </div>
```

Исходный код шаблона `booking_request.html`

```
{% extends 'newjetway/base_normal.html' %}

{% block title %}Aircraft booking - Jetway Online Charter
Marketplace{% endblock %}

{% load staticfiles %}
{% block extrastyle %}
    {{ block.super }}
```

```
8.      <link rel="stylesheet" href="{% static 'css/
    booking_request.css' %}"/>
9.  {% endblock %}
10.
11. {% block content %}
12.     <div class="section-nav"></div>
13.     {% include 'newjetway/quote_fsm.html' with
    fsm_status="BOOKING_REQUEST" %}
14.     {% include 'newjetway/flight_card.html' with res=res %}
15.     {% include 'newjetway/booking_footer.html' with fixed="fixed"
    href=context.request_button onclick=""%}
16.     <div class="footer-placeholder"></div>
17. {% endblock %}
18.
19. {% block endscripts %}
20.     {{ block.super }}
21.     <script src="{% static 'js/init_pages.js' %}"></script>
22.     <script src="{% static 'jssor-slider/js/jssor.slider.mini.js'
    %}"></script>
23.     <script src="{% static 'notifyjs/dist/notify.js' %}"></script>
24.     <script src="{% static 'js/booking_request.js' %}"></script>
25. {% endblock %}
26. Исходный код шаблона base_normal.html
27. {% extends "newjetway/base.html" %}
28.
29. {% load menu %}
30.
31. {% block content_leading %}
32.     {% if DEBUG_ENV %}
33.         <div id="debug_alert">
34.                 This is <b>not</b> a production Jetway instance.
    Current enviroment: <div class="env_name">{{ DEBUG_ENV }}</div>
35.         </div>
36.     {% endif %}
37.     {% generate_menu %}
38.     {% include "newjetway/widgets/topnavbar.html" with
    navbar=menus.navbar account_nav=menus.account_nav %}
39. {% endblock %}
```

```
40.
41.  {% block footer %}
42.      {% generate_menu %}
43.      {% include "newjetway/widgets/footer.html" with
     center_menu=menus.center_footer right_menu=menus.right_footer %}
44.  {% endblock %}
```

Приложение 3

Исходный код представления `booking_request`

```
1.  def prepare_flight_for_template(search_result):
2.      num = 0
3.      flight = search_result.flight
4.      flight.show_repositions =
    search_result.search_request.show_repositions
5.      for i in range(len(flight.route.segments)):
6.          if flight.route.segments[i].is_reposition and not
    flight.show_repositions:
7.              continue
8.          num += 1
9.          flight.route.segments[i].leg_number = num
10.         flight.route.segments[i].time_duration = \
11.           timedelta_hm(flight.route.segments[i].time_duration)
12.     flight.number_of_visible_legs = num
13.     flight.ac.cover_image = flight.ac.cover_image().url
14.     return flight
15.
16. def booking_request(request, uuid):
17.     if Quote.objects.filter(uuid=uuid).first():
18.         return redirect('quote:status', uuid=uuid)
19.     res = get_object_or_404(SearchResult, uuid=uuid)
20.     context = {'request_button':reverse('search:contact_info',
    kwargs={'uuid':uuid}),}
21.     return render(request, 'newjetway/search/
    booking_request.html', {
22.         'res': prepare_flight_for_template(res),
23.         'context': context}
24.     )
```

Приложение 4

Фрагмент файла конфигурации URL `jetway/urls.py`

```
.....
1.  urlpatterns = [
2.      url(r'^grappelli/', include('grappelli.urls')),
3.      url(r'^admin/', admin.site.urls),
4.
5.      url(r'^geography/', include('geography.urls',
    namespace='geography')),
6.      url(r'^fleet/', include('fleet.urls', namespace='fleet')),
7.      url(r'^search/', include('search.urls', namespace='search')),
8.      url(r'^quote/', include('quotes.urls')),
9.      url(r'^sms_confirmation/', include('sms_confirmation.urls',
    namespace='sms_confirmation')),
10.     url(r'^account/', include('account.urls',
    namespace='account')),
.....
```

Фрагмент файла конфигурации URL `search/urls.py`

```
.....
1.  urlpatterns = [
2.      url(
3.          r'^$',
4.          views.index_page,
5.          name='index',
6.      ),
7.
8.      url(
9.          '^query/$',
10.         views.search_query,
11.         name='search_query',
12.     ),
13.     url(
14.         '^result/(?P<uuid>[\w-]+)/sidepanel$',
15.         views.search_result_sidepanel,
16.         name='search_result_sidepanel',
17.     ),
18.     url(
```

```
19.           '^result/(?P<uuid>[\w-]+)$',
20.           views.booking_request,
21.           name='booking_request',
22.       ),
23.       url(
24.           '^result/(?P<uuid>[\w-]+)/contact-info$',
25.           views.contact_info,
26.           name='contact_info',
27.       ),
28.       url(
29.           '^result/(?P<uuid>[\w-]+)/contact-info/expired$',
30.           views.contact_info_expired,
31.           name='contact_info_expired',
32.       ),
33.   ...
```

Приложение 5

Исходный код модели `RosterFile`

```
1.  class RosterFile(models.Model):
2.      airline = models.ForeignKey('fleet.Airline')
3.      roster = models.FileField(upload_to='rosters')
4.      timestamp = models.DateTimeField(auto_now_add=True)
5.      uploader = models.ForeignKey(User, blank=True, null=True,
    editable=False)
6.
7.      STATUS_CHOICES = (
8.          ('PD', 'Pending'),
9.          ('RU', 'Processing'),
10.         ('OK', 'OK'),
11.         ('FL', 'Failed'),
12.         ('IE', 'Internal error'),
13.     )
14.     status = models.CharField(choices=STATUS_CHOICES,
    default='PD', max_length=2, editable=False)
15.     comment = models.TextField(blank=True, editable=False)
16.
17.     def save(self, **kwargs):
18.         super().save(**kwargs)
19.         if self.status == 'PD':
20.             roster_file_proceed.delay(self)
21.
22.
23. @shared_task
24. def roster_file_proceed(roster_file):
25.     if roster_file.status != 'PD':
26.         return
27.     roster_file.comment = ''
28.     roster_file.status = 'RU'
29.     roster_file.save()
30.
31.     driver = roster_file.airline.roster_driver
32.     if driver is None:
33.         roster_file.comment = 'Roster driver is undefined for {}
    airline'.format(roster_file.airline)
```

```
34.             roster_file.status = 'FL'
35.             roster_file.save()
36.             return
37.
38.         try:
39.             driver()(roster_file.airline, roster_file.roster.path)
40.         except RosterDriverException as E:
41.             roster_file.status = 'FL'
42.             roster_file.comment = str(E)
43.             roster_file.save()
44.             return
45.         except:
46.             roster_file.status = 'IE'
47.             roster_file.comment = "Internal error. Please contact
    administrator."
48.             roster_file.save()
49.             raise
50.
51.         roster_file.status = 'OK'
52.         roster_file.save()
```