МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

**«Южно-Уральский государственный университет»**

**(национальный исследовательский университет)**

Факультет Вычислительной математики и информатики

Кафедра системного программирования

|  |  |
| --- | --- |
| РАБОТА ПРОВЕРЕНА  Рецензент  Директор ООО «Грид-Инжиниринг»  \_\_\_\_\_\_\_\_\_\_В.А. Дорохов  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2014 г. | ДОПУСТИТЬ К ЗАЩИТЕ  Заведующий кафедрой, д.ф.-м.н., профессор  \_\_\_\_\_\_\_\_\_\_Л.Б. Соколинский  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2014 г. |

**Разработка сервера и веб-сайта для сетевой игры "12 подвигов" на платформе Django**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЮУрГУ– 010400.62.2014.10-012-1887.КР

|  |  |
| --- | --- |
|  | Научный руководитель  к. ф.-м. н., доцент кафедры СП  \_\_\_\_\_\_\_\_\_\_Г.И. Радченко  Автор работы, студент группы ВМИ-456  \_\_\_\_\_\_\_\_\_\_Н.И. Барабанщикова  Ученый секретарь  (нормоконтролер)  \_\_\_\_\_\_\_\_\_\_\_\_\_ О.Н. Иванова  “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2014 г. |

Челябинск-2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования

**«Южно-Уральский государственный университет»**

**(национальный исследовательский университет)**

Факультет Вычислительной математики и информатики

Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

08.02.2014

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы бакалавра**

студенту группыВМИ-456 Барабанщиковой Наталье Игоревне,

обучающемуся по направлению 010400.62 «Информационные технологии»

1. **Тема работы**(утверждена приказом ректора от 12.03.2014 № 368)

Разработка сервера и веб-сайта для сетевой игры "12 подвигов" на платформе Django.

1. **Срок сдачи студентом законченной работы:**04.06.2014.
2. **Исходные данные к работе**
   1. Головатый А., Каплан-Мосс Дж. Django. Подробное руководство, 2-е издание. Пер. с англ. СПб.: Символ-Плюс, 2013. 560 с..
   2. Официальный веб-сайт платформы Django Rest [Электронный ресурс]. URL: <http://www.django-rest-framework.org>
   3. Чан У., Биссекс П., Форсье Д. Django. Разработка веб-приложений на Python*.* СПб: Символ-плюс, 2009. 456 с.
3. **Перечень подлежащих разработке вопросов**
   1. Выполнить анализ требований к системе;
   2. Спроектировать архитектуру сервера, включая диаграммы прецедентов, компонентов и деятельностей;
   3. Реализовать сервер игры, включая REST-сервис;
   4. Протестировать систему модульными и функциональными тестами
4. **Дата выдачи задания:** 08.02.2014.

**Научный руководитель**

Доцент кафедры СП,

кандидат физ.-мат. наук Г.И. Радченко

**Задание принял к исполнению** Н.И. Барабанщикова

Оглавление

[Введение 5](#_Toc389114904)

[Актуальность темы 5](#_Toc389114905)

[Цели и задачи 6](#_Toc389114906)

[Структура и объем работ 6](#_Toc389114907)

[1. Обзор литературы 8](#_Toc389114908)

[1.1 Обзор альтернативных облачных серверов 8](#_Toc389114909)

[1.1.1 Windows Azure 8](#_Toc389114910)

[1.1.2 Google Cloud 8](#_Toc389114911)

[1.2 Выбор средства разработки 9](#_Toc389114912)

[1.3 Архитектура REST 10](#_Toc389114913)

[1.4 Выбор СУБД 12](#_Toc389114914)

[1.5 Вывод 13](#_Toc389114915)

[2. Требования к серверу игры «12 Подвигов» 14](#_Toc389114916)

[2.1 Функциональные требования 14](#_Toc389114917)

[2.2 Нефункциональные требования 15](#_Toc389114918)

[2.3 Описание прецедентов 16](#_Toc389114919)

[2.4 Вывод 18](#_Toc389114920)

[3. Архитектура сервера игры «12 Подвигов» 19](#_Toc389114921)

[3.1 Компоненты сервера игры «12 Подвигов» 19](#_Toc389114922)

[3.2 Процессы работы сервера игры «12 Подвигов» 21](#_Toc389114923)

[3.2.1 Процесс добавления нового задания 21](#_Toc389114924)

[3.2.2 Процесс выдачи задания мобильному клиенту 22](#_Toc389114925)

[3.3 Вывод 23](#_Toc389114926)

[4. Реализациясервераигры«12 Подвигов» 24](#_Toc389114927)

[4.1 Схема базы данных 24](#_Toc389114928)

[4.2 Реализация сервера игры 26](#_Toc389114929)

[4.3 Интерфейс администратора 29](#_Toc389114930)

[4.4 Реализация REST-сервиса 34](#_Toc389114931)

[4.5 Вывод 39](#_Toc389114932)

[5. Тестирование 40](#_Toc389114933)

[5.1 Модульное тестирование 40](#_Toc389114935)

[5.2 Тестирование REST-сервиса 42](#_Toc389114936)

[5.2.1 Регистрация новых игроков 43](#_Toc389114937)

[5.2.2 Авторизация игроков 44](#_Toc389114938)

[5.2.3 Редактирование профиля 45](#_Toc389114939)

[5.2.4 Получение случайного задания 46](#_Toc389114940)

[5.2.5 Завершение задания 47](#_Toc389114941)

[5.3 Вывод 48](#_Toc389114942)

[Заключение 49](#_Toc389114943)

[Литература 50](#_Toc389114944)

# Введение

## Актуальность темы

На сегодняшний день мобильные приложения находятся на пике своей популярности. Согласно данным исследования компании J’son & Parthers Consulting, за 2012 год рынок мобильных приложений в мире составил $7,83 млрд. Эксперты прогнозируют, что в дальнейшем этот рост будет стремительно расти вверх: к 2016 году составит $65,79 млрд[15].

К 2013 году мобильные устройства становятся по-настоящему массовыми, в то время как продажи персональных компьютеров продолжают падать. Почти каждый, кто имеет на руках мобильное устройство (смартфон, коммуникатор или планшет) пользуется приложениями: браузером, клиентом электронной почты и мгновенных сообщений, играми. Число загрузок популярных приложений в Google Play и Apple App Store уже измеряется сотнями миллионов. Поэтому вопрос разработки новой мобильной игры с необычным сюжетом является актуальным.

Значительная часть современных [приложений для мобильных платформ](http://www.enterra.ru/mobile-application-development) (iOS, Android) работает в паре с сервером. Разделение мобильного приложение на клиентскую (представление данных) и серверную (прикладная логика) части обусловлено рядом следующих причин.

1. Можно создать единую систему управления контентом приложения, а для поддержки новых устройств адаптировать только фронт-энд, не затрагивая прикладную логику.
2. Сетевое приложение может обеспечить постоянное обновление данных с сервера на устройство, предоставляя актуальный контент пользователю.
3. Сетевое приложение позволяет обеспечить взаимодействие пользователей друг с другом, обмен информацией, различные виды социальной активности.

Мобильные игры не всегда носят только лишь развлекательный характер. Некоторые из них могут выступать в роли «коуча» - «тренера успеха личности». Мобильные приложения позволяют проводить тренинг игрока в интересующей его области в легкой игровой форме. Обучение и развитие при помощи мобильных игр куда более интереснее и приятнее, чем обычное чтение учебников или просмотр видеоматериалов. Поэтому дополнительным преимуществом в разработке мобильного приложения является включение в него «коучинг» составляющей.

## Цели и задачи

Основной целью данной работы является разработка сервера и веб-сайта для мобильной игры «12 Подвигов». Мобильное приложение игры взаимодействует с сервером для получения карточек с заданиями, которые ежедневно высылаются игрокам. На сервере должны быть реализованы алгоритмы выдачи заданий игрокам, процедуры автоматической генерации карточек с заданиями. Сайт мобильной игры предназначен в первую очередь для администрирования пользователей и игрового контента.

Для достижения цели надо решить следующие задачи:

1. изучить язык программирования Python, платформу Django;
2. изучить основы архитектуры REST, библиотеку django-rest-framework;
3. выполнить анализ требований к системе;
4. выполнить проектирование системы;
5. реализовать систему в соответствии с установленными требованиями;
6. выполнить модульное и функциональное тестирование системы.

## Структура и объем работ

Работа состоит из введения, пяти глав, заключения, библиографии и приложения. Объем работы составляет 51 страницу, объем библиографии –16 источников.

В первой главе дается обзор аналогов облачных серверов для построения клиент-серверных мобильных приложений. В этом разделе также будут рассмотрены технологии и платформы для построения сервера и веб-сайта игры «12 Подвигов».

Вторая глава содержит общую информацию о сетевой игре «12 Подвигов», описание и анализ функциональных и нефункциональных требований к системе и вариантов использования системы.

В третьей главе представлена архитектура системы. В этом разделе подробно рассмотрено статическое и динамическое представление архитектуры разрабатываемой системы.

Четвертая глава посвящена реализации всех компонентов системы на платформе Django с использованием технологии REST.

В пятой главе приводятся результаты модульного тестирования системы всех компонентов разработанной системы.

В заключении сделаны выводы о проделанной работе.

# Обзор литературы

## Обзор альтернативных облачных серверов

В связи с развитием технологий облачных вычислений у разработчиков мобильных приложений появляется возможность выбора облачной платформы для разработки сервера приложения. Рассмотрим особенности работы и возможности наиболее популярных из них.

### Windows Azure

[*Windows*](http://msdn.microsoft.com/ru-ru/jj712214) [*Azure*](http://msdn.microsoft.com/ru-ru/jj712214) [*Mobile*](http://msdn.microsoft.com/ru-ru/jj712214) [*Services*](http://msdn.microsoft.com/ru-ru/jj712214)*—* это облачный сервис, который предназначен для обеспечения мобильных приложений готовым серверным бэкендом. WindowsAzureMobileServices предлагают поддержку всех современных популярных мобильных платформ, включая[Windows](http://msdn.microsoft.com/ru-ru/jj712659), [iOS](http://msdn.microsoft.com/ru-ru/dn133811), [Windows Phone](http://msdn.microsoft.com/ru-ru/dn133820), [Android](http://msdn.microsoft.com/ru-ru/dn133819) [7].

Разработчики, пользующиеся сервисом Windows Azure Mobile Services, получают единую, готовую инфраструктуру для всех своих приложений, включая реляционное хранилище данных, возможность аутентификации пользователей через популярные социальные сети (Facebook, Twitter, Microsoft Account, Google) и настроенный функционал для отправки push-уведомления на все платформы с помощью одного API [8].

Мобильные сервисы Windows Azure Mobile Services предлагают разработчикамбыстрый старт с бесплатного тарифаи гибкое масштабирование по ресурсам по требованию.

### Google Cloud

Компания Google предлагает сразу две облачные платформы *Google Cloud Endpoints* и [*Mobile*](http://apps4all.ru/news/apple/google-mobile-backend-starter-ios.html) [*Backend*](http://apps4all.ru/news/apple/google-mobile-backend-starter-ios.html) [*Starter*](http://apps4all.ru/news/apple/google-mobile-backend-starter-ios.html), которые позволяют «в один клик» развернуть облачный бекенд для мобильных приложений, позволяя разработчикам сконцентрироваться на более инновационных аспектах своей работы [10].

*Google Cloud Endpoints*– это набор инструментов и библиотек, упрощающих создание мобильных облачных приложений. Связь с внешними сервисами осуществляется при помощи REST-запросов к API, которые предоставляет Google для доступа к сервисам. Мобильное приложение может получить упрощенный доступ ко многим сервисам Google App Engine, включая Datastore, Google Cloud Storage, Mail, Url Fetch и Task Queues. Endpoints также автоматизирует системные задачи, например, балансировку нагрузки, масштабирование ресурсов и сопровождение сервера [11].

*Mobile Backend Starter* [12] предоставляет полную бэкэнд-инфраструктуру, радикально снижающую количество усилий, требуемых для обеспечения работоспособности приложений. Она включает в себя сервисы хранения данных, аутентификации и push-уведомлений, а также средства создания событийно-зависимой логики.

## Выбор средства разработки

На основе анализа существующих платформ для разработки серверной части мобильных приложений, нами был сделан выбор в сторону самостоятельной разработки бэкенда для мобильного приложения на платформе Django, т.к. такой подход обеспечивает максимальную независимость от поставщиков облачных решений, низкую стоимость и высокую гибкость.

В качестве главного средства разработки приложения была выбрана платформа Django на языке Python [3].

Django позволяет создавать динамические веб-приложения за довольно небольшой промежуток времени. Платформа Django спроектирована таким образом, чтобы разработчики смогли сосредоточиться на решении содержательных задач, а не отвлекаться на реализацию тривиальных потребностей (аутентификации и авторизации пользователей, настройки работы с базой данных и др). Для достижения этой цели Django предоставляет общеупотребительные шаблоны веб-разработки высокого уровня абстракции, инструменты для быстрого выполнения часто встречающихся задач программирования и четкие соглашения о способах решения проблем [2].

Проект на Django строится из одного или нескольких приложений, которые рекомендуются делать отчуждаемыми и переносимыми.

Также, в отличие от других платформ разработки, обработчики URL в Django конфигурируются явно при помощи [регулярных выражений](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F), а не выводятся автоматически из структуры моделей контроллеров [6].

Для работы с базами данных Django использует собственный ORM, в котором модель данных описывается классами Python, и по ней генерируется схема базы данных.

Архитектура Django основана на классической модели проектирования MVC –Модель-Представление-Контроллер [1]. Но поскольку контролеры в Django реализованы самой платформой, Django стали называть MTV (Model-Template-View)-фреймворком, где:

* Model– уровень доступа к данным
* Template – уровень отображения (представления данных)
* View – уровень бизнес-логики.

Главное достоинство такой архитектуры – слабая связанность и строгое отделение различных компонентов приложения друг от друга.

## Архитектура REST

REST (*Representational State Transfer*, «передача состояния представления») — это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Системы, поддерживающие REST, называются RESTful-системами [4].

Ключевые принципы REST:

* **Не зависимость от состояния** (состояние клиента хранится только на клиенте; вся информация, необходимая серверу для обработки запроса должна содержаться в самом запросе);
* **Кешированая архитектура** (ответ сервера может быть закеширован на клиенте и использован повторно без новых обращений);
* **Разделение на клиента и сервера** (клиент знает только об интерфейсе сервера, детали реализации сервера скрыты от клиента).

REST обеспечивает простой интерфейс управления информацией на базе протокола HTTP без необходимости использования каких-либо дополнительных внутренних прослоек [16]. Каждая единица информации *(ресурс)*, с которой необходимо работать через REST однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат. Например, третья книга с книжной полки будет иметь вид /book/3, а 35 страница в этой книге - /book/3/page/35. Причем, сам формат данных (HTML, jpeg-файл, документ Microsoft Word), расположенных по адресу /book/3/page/35 совершенно не имеет значения.

Управление информацией сервиса основывается на протоколе передачи данных (как правило, это протокол HTTP). Для HTTP протокола действие над данными задается с помощью *методов*:

* GET- используется для получения представления ресурсов;
* PUT- используется для создания или обновления ресурса;
* POST- используется для создания зависимого ресурса;
* DELETE- используется для удаления ресурса.

Ниже приведены примеры правильного REST-API:

* GET /book/ - получить список всех книг;
* GET /book/3/ - получить книгу под номером 3;
* POST /book/ - добавить новую книгу;
* PUT /book/3/ – изменить данные о книге под номером 3;
* DELETE /book/3/ - удалить книгу под номером 3.

Для реализации REST-сервиса в рамках сервера игры «12 Подвигов» был выбран инструмент Django Rest Framework [5], который хорошо сочетается с идеологией Django.

## Выбор СУБД

В качестве базы данных была выбрана PostgreSQL [14].

Любое изменение модели данных (добавление или удаление полей, изменение типа данных) предполагает необходимость вносить изменения в базу данных вручную, используя синтаксис команд SQL, либо создавать базу данных заново, теряя при этом все данные. Ручные изменения в базе данных не только неудобны, но зачастую могут быть несовместимыми и ошибочными, что может привести к повреждению данных у конечных пользователей. Кроме того, будет трудно отменить уже внесенные изменения или проанализировать историю изменений в базе данных за какой-либо период.

Избежать неудобств ручного подхода позволяет стратегия изменения базы данных, при которой вмешательство человека минимизируется. При помощи комбинации методик и инструментов можно организовать непротиворечивое и повторяемое управление изменениями в структуре базы данных и в самих данных. Для автоматизации процесса изменения структуры базы данных используется механизм миграций.

Миграции данных – это удобный и быстрый способ изменять схему базы данных время от времени неизменным и простым образом [16]. Каждую миграцию можно рассматривать как новую версию базы данных. Схема изначально ничего не содержит, а каждая миграция изменяет ее, добавляя или убирая таблицы, столбцы или записи. Всегда можно откатить БД на любую, более раннюю версию миграции.

В качестве инструмента для миграции данных для веб-приложения был выбран Django South [13]. Утилита South позволяет хранить историю миграций и предоставляет возможность корректировать код миграции.

Изменение базы с помощью South происходит в два этапа:

* South определяет, что нужно изменить и создает файл-миграцию;
* Миграция применяется к базе данных, отражая изменения моделей.

## Вывод

Изучив существующие возможности реализации сервера, проанализировав их главные достоинства и недостатки, для реализации сервера сетевой игры "12 Подвигов" было решено разрабатывать систему на выбрана платформе Django на языке Python. Для реализации RESTсервиса была выбрана платформа django-rest-framework. В качестве базы данных была выбрана СУБД PostgreSQL.

# Требования к серверу игры «12 Подвигов»

Смысл мобильной социальной игры «12 Подвигов» заключается в следующем. Каждый день игроку на мобильное приложение приходит карточка с рисунком-комиксом и описанием задания, которое нужно выполнить в течение одного или нескольких дней (в зависимости от сложности задания). Игроки смогут запрашивать специальные тематические задания или выполнять задания по группам. К выполненным заданиям есть возможность прикладывать фотографии, коллажи, видеоматериалы в качестве отчета о выполнении задания. Игроки могут оценивать качество выполнение заданий другими участниками игры, оставлять свои комментарии, делиться своими заданиями в социальных сетях.

## Функциональные требования

Сервер и веб-сайт игры «12 Подвигов» должен обеспечивать возможность добавления, редактирования, хранения и предоставления всего контента игры, включая списки администраторов и игроков, задания для пользователей, статистику по выполненным заданиям, профили игроков и др.

Веб-сайт должен предоставлять удобный интерфейс для добавления и редактирования этой информации, обеспечивать возможность хранения и перевода заданий на разные языки, обеспечивать автоматическую генерацию карточки с заданием на основе изображения и текста задания.

Также, сервер должен обеспечивать REST-интерфейс для авторизации пользователей и для получения заданий внешними клиентами, включая внешние веб-сайты и мобильные клиенты.

*Функциональные требования* определяют функциональность программного обеспечения, то есть описывают, какие возможности должна предоставлять разрабатываемая система. Функциональные требования включают в себя бизнес-требования и пользовательские требования. Нами были выявлены следующие функциональные требования для реализации веб-приложения мобильной социальной игры «12 Подвигов».

1. Система должна предоставлять интерфейс для добавления и редактирования заданий.
2. Система должна предоставлять интерфейс для администрирования пользователей.
3. Система должна поддерживать доступ к каталогу заданий посредством REST- сервиса.
4. Система должна обеспечивать возможность регистрации новых пользователей посредством REST- сервиса.
5. Система должна генерировать карточки с заданиями на основе рисунка и текста с заданием.
6. Система должна назначать задания игрокам в соответствии с их профилем и обеспечивать максимальный период не повторения заданий.
7. Система должна предоставлять возможность регистрации игрока через мобильное приложение.

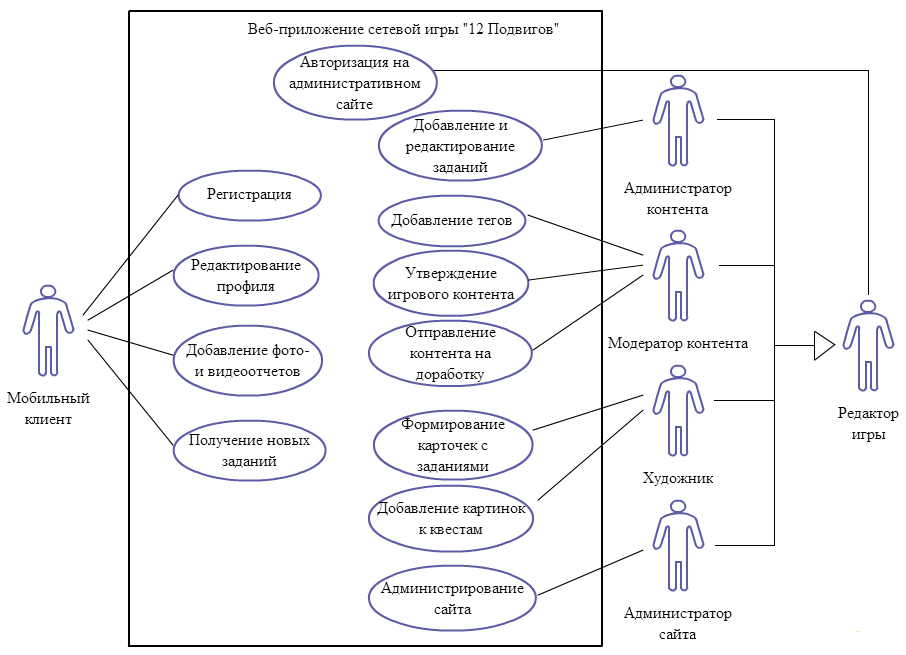
## Нефункциональные требования

*Нефункциональные требования* описывают свойства и ограничения, накладываемые на информационную систему. Нефункциональные требования определяют бизнес-правила, системные требования и т.д. Для реализации веб-приложения мобильной социальной игры «12 Подвигов» были зафиксированы следующие нефункциональные требования:

1. Система должна быть разработана на платформе Django.
2. Система должна предоставлять возможности взаимодействия на основе архитектурного стиля REST.
3. Система должна быть реализована в соответствии с трехзвенной клиент-серверной архитектурой: клиент - сервер приложений - сервер БД.

## Описание прецедентов

Можно выделить пять основных актеров, взаимодействующих с разрабатываемой системой (Рис. 1):



**Рис. 1.** Диаграмма прецедентов веб-приложения мобильной социальной игры «12 Подвигов»

*Администратор сайта* является суперпользователем, который занимается администрированием пользователей сайта: добавляет и удаляет пользователей, назначает пользователям роли, в соответствии с которыми разные функции веб-приложения доступны одним пользователям и скрыты от других.

*Администратор контента* добавляет информацию о новых заданиях в социальной игре.

*Модератор контента* может добавлять теги и темы заданий, просматривать информацию о всех существующих заданиях и редактировать их, а также утверждать задания к игре или отправлять контент на доработку, при этом отправляя сообщение создателю задания со всеми недочетами.

*Художник* создает и добавляет картинки к заданиям и формирует карточки с заданиями путем наложения текста задания на картинку.

*Мобильный клиент* получает карточки с заданиями с сервера для выполнения заданий игроками. Игрок может выкладывать фото- и видеоматериалы, свидетельствующие о выполнении задания, а также просматривать, как справляются с подобными заданиями другие игроки и оценивать их.

С каждым актером системы связано один или несколько прецедентов. На данной диаграмме некоторые низкоуровневые прецеденты объединены в прецеденты более высокого порядка.

*Администрирование сайта* – администратор сайта может работать с пользователями, группами пользователей и их правами на выполнения некоторых действий.

*Добавление и редактирование заданий* – администратор контента может добавлять информацию о новых заданиях мобильной игры в базу данных.

*Добавление тегов* – модератор добавляет в базу данных информации о тегах, которые ограничивают возможность выполнении каждого задания определенной группой лиц.

*Утверждение игрового контента*–модератор просматривает новые задания и утверждает их к игре, если к ним нет нареканий.

*Отправление контента на доработку* – модератор может запретить часть заданий, в этом случае случаи контент отправляется на доработку к его создателю.

Добавление картинок к заданиям – художник добавляет к заданиям картинки-комиксы с персонажами, олицетворяющими выполнение задания.

*Формирование карточек c заданиями –* художник может изменять положение элементов на сгенерированной сервером карточке, в случае, если они расположены не достаточно корректно.

*Регистрация –* игрок может зарегистрироваться для участия в игре через мобильное приложение.

*Редактирование профиля –* игрок может изменять данные своего профиля (никнейм, пол, возраст, аватар) аналогично другим социальным сетям.

*Получение новых заданий* – мобильный клиент может запрашивать новые карточки с заданиями для игрока в соответствии c его заполненным профилем.

*Добавление фото- и видеоотчетов* – игрок может добавлять фотографии и видеозаписи о выполнении заданий, оставлять комментарии к записям и оценивать мультимедиа материалы других игроков.

## Вывод

В процессе анализа требований было составлено техническое задание, в котором сформулированы основные функциональные и нефункциональные требования, предъявляемые к разрабатываемой системе, определены главные пользователи системы и варианты использования системы пользователями.

# Архитектура сервера игры «12 Подвигов»

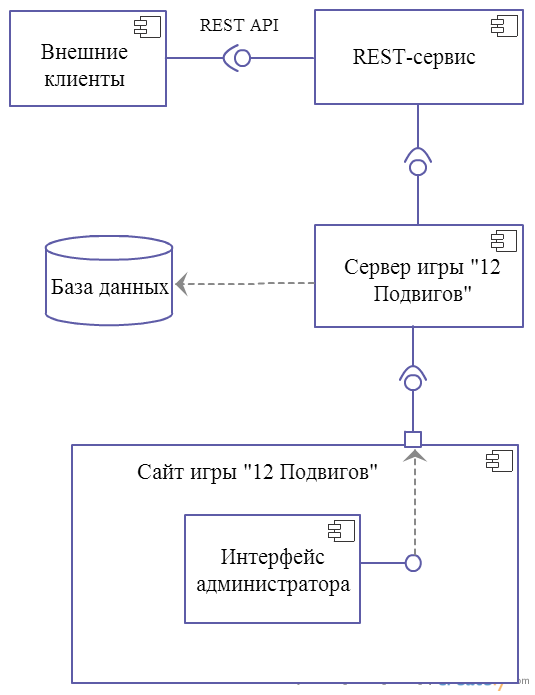
Архитектура игры «12 Подвигов» представляет собой трехзвенную архитектуру: клиент - сервер приложений - сервер базы данных, где в качестве клиентов выступают мобильные приложения.

## Компоненты сервера игры «12 Подвигов»

Для проектирования статического представления архитектуры игры «12 Подвигов» была выбрана диаграмма компонентов. Диаграмма компонентов наглядно изображает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами.

Диаграмма компонентов игры «12 Подвигов» представлена на Рис. 2 и состоит из следующих компонентов:

* Внешние клиенты;
* REST-сервис;
* Сервер игры «12 Подвигов»;
* Сайт игры «12 Подвигов».



**Рис. 2.**Диаграмма компонентов сетевой мобильной игры «12 Подвигов»

*Сервер игры «12 Подвигов»* хранит информацию об администраторах и игроках, списки заданий и изображения с карточками заданий. На сервер вынесены многие функции обработки данных, включая генерацию карточек с заданиями на основе текста с заданием и картинки-комикса, выдачу игрокам заданий, соответствующих их профилю, получение случайного задания и т.д.

*Сайт игры «12 Подвигов»* предоставляет интерфейс администратора игры. Интерфейс администратора предназначен для добавления, редактирования и отображения информации о заданиях и администрирования пользователей игры. Интерфейс администратора формируется с использованием стандартного подхода к созданию административного интерфейса, предоставляемого платформой Django. Интерфейс администратора предоставляет отдельные методы для работы с системой, в зависимости от того, какими правами обладает пользователь, вошедший в систему. Доступ к административному интерфейсу имеют только редакторы игры «12 подвигов».

Взаимодействие между клиентами и сервером игры происходит посредством REST API, предоставляемым *REST-сервисом*. REST-сервис обеспечивает передачу информации на базе протокола HTTP. REST-сервис реализует следующие функции:

* создание новых игроков по логину и паролю;
* создание, редактирование и получение профиля игроков;
* выдача заданий игрокам.

*Внешние клиенты* (мобильные устройства) постоянно взаимодействуют с сервером игры «12 Подвигов» посредством REST-сервиса для регистрации новых игроков и редактирования профиля игрока. В то же время внешние клиенты регулярно запрашивают у сервера карточки с заданиями, которые предлагается выполнить игрокам.

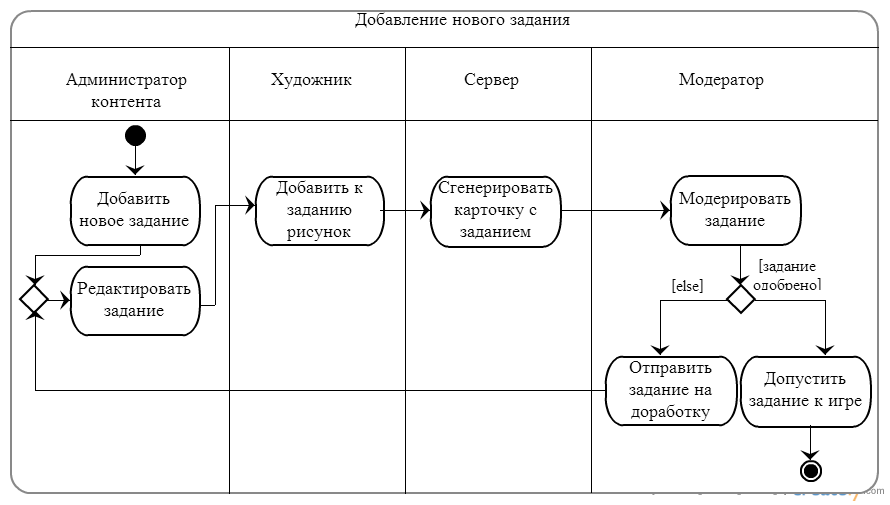
## Процессы работы сервера игры «12 Подвигов»

Наиболее интересные процессы работы сервера игры описаны в этом разделе. Для проектирования динамического преставления архитектуры системы были выбраны диаграммы деятельности и последовательности. Данные диаграммы описывают взаимодействие групп объектов в различных условиях их поведения. Диаграмма деятельности показывает, как поток управления переходит от одной деятельности к другой. Диаграмма последовательности моделирует взаимодействие объектов во времени.

### Процесс добавления нового задания

Одним из важных аспектов развития игры является регулярное добавление новых заданий. Процесс добавления нового задания происходит в несколько этапов (Рис. 3):

1. Администратор контента добавляет новое задание к игре, определяя основную информацию о задании (язык, название, текст задания).
2. Художник добавляет к заданию рисунок.
3. Сервер генерирует карточку с заданием на основе рисунка, фона и текста задания. Художник может отредактировать представление карточки, исправив параметры текста, положение картинки на карточке и др.
4. Модератор просматривает добавленные задания, добавляет список тегов, ограничивающих группу лиц, которыми может быть выполнено данное задание, делает поправки и может либо допустить задание к игре либо отправить задание администратору контента на доработку.
5. В случае, если задание было отправлено на доработку, администратор контента вносит поправки и вновь отправляет задание на модерацию.
6. Если задание было допущено модератором к игре, процесс завершается.

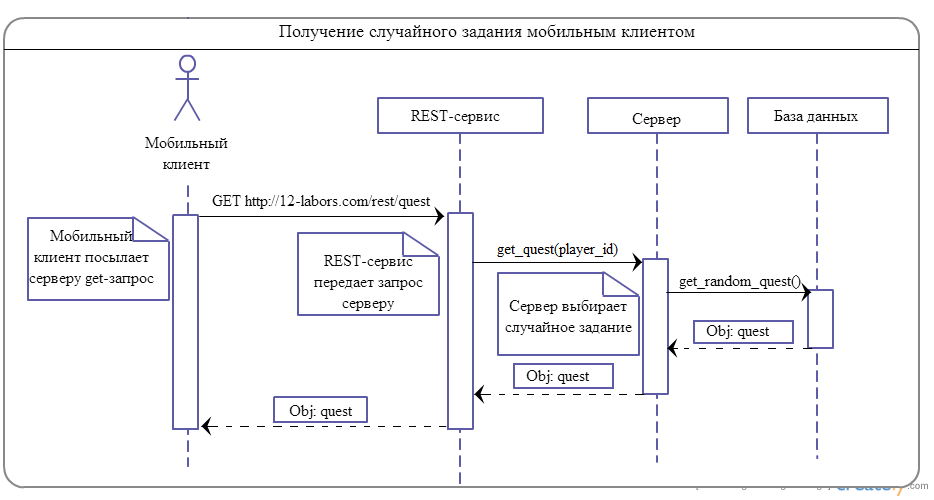
****

**Рис. 3.**Диаграмма деятельности процесса добавления нового задания

### Процесс выдачи задания мобильному клиенту

Мобильный клиент в ходе своей работы регулярно запрашивает новые задания для игроков. Процесс выдачи задания мобильному клиенту представлен на Рис. 4 и выглядит следующим образом:

1. Мобильный клиент отправляет запрос серверу на получение нового задания посредством RESTAPI. Внутри запроса клиент передает информацию для аутентификации и авторизации: зашифрованную пару имя пользователя-пароль.
2. REST-сервис производит авторизацию игрока, преобразуя пару логин-пароль в уникальный идентификатор игрока, который передает серверу
3. Сервер получает запрос от мобильного клиента через REST-сервис. Сервер выбирает задание для игрока из списка допущенных к игре заданий, соответствующих его профилю. Для проверки соответствия задания профилю, сервер проверяет, чтобы список ограничений, наложенный на задание, совпадал со списком ограничений профиля игрока.
4. Сервер возвращает задание в виде карточки мобильному клиенту, и процесс завершается.



**Рис. 4.**Диаграмма последовательности процесса выдачи задания мобильному клиенту

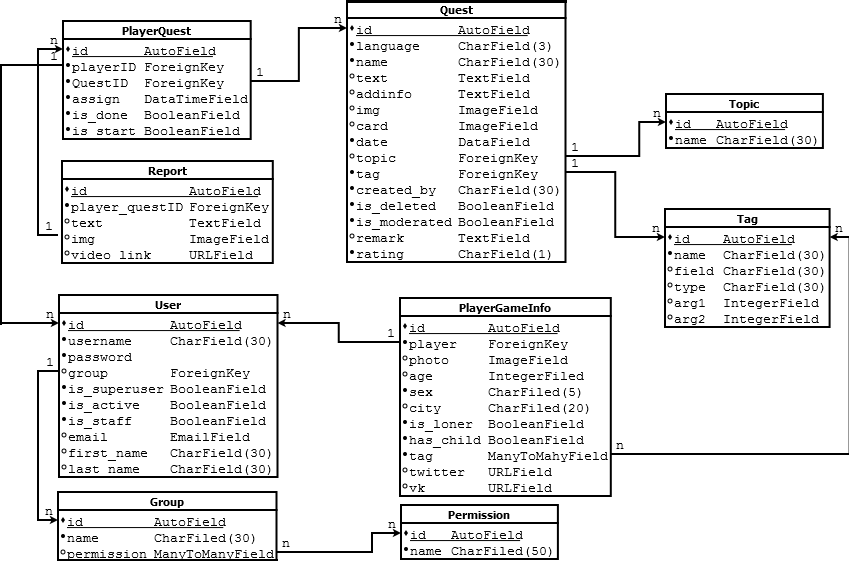
## Вывод

В результате была спроектирована архитектура сервера игры «12 Подвигов», состоящая из трех основных компонентов: сервер игры, сайт и REST-сервис. Все компоненты являются слабосвязанными, и изменение одного из них не влияет на другие.

# Реализациясервераигры«12 Подвигов»

## Схема базы данных

Для реализации сервера мобильной социальной игры «12 Подвигов» была спроектирована и реализована следующая база данных (Рис. 5).



**Рис.** **5.** Схема базы данных сервера игры «12 Подвигов»

Рассмотрим основные таблицы базы данных.

В таблице *Quest* хранится информации о заданиях игры. Каждое задание имеет обязательные поля, вроде названия задания, текста задания и карточки и необязательные поля - дополнительная информация о задании. Для обеспечения возможности выдачи заданий на разных языках определено поле «язык». Поле «рейтинг» позволяет отсортировать список заданий для проведения модерации по приоритету заданий.

Таблица *Тag* содержит ограничения, которые накладываются на задания, поскольку для разных групп пользователей не могут быть выданы одинаковые задания. Таблица определяет следующие поля: поле для ограничения, тип ограничения (isTrue, isFalse, between и др.) и параметры.

Таблица *Topic* предоставляет возможность сортировать задания по темам (спорт, языковая практика, программирование).

Таблица *User* предназначена для хранения информации обо всех пользователях системы (редакторах игры и игроках). Таблица содержит два обязательных поля: имя пользователя и пароль, и несколько необязательных полей, включая email, активность аккаунта, дата последнего входа в систему. Логическое поле is\_staff определяет, имеет ли пользователь доступ к интерфейсу администратора.

Таблица *Group* позволяет объединять пользователей в группы пользователей (например, группа художников, группа модераторов игры). Группы – это механизм назначения разрешений сразу нескольким пользователям.

Таблица *Permissions* содержит флаги (да/нет), показывающие, разрешено ли пользователю выполнять некоторую операцию (например, возможность добавлять, изменять или удалять некоторый объект).

Таблица *PlayerGameInfo* – это расширение таблицы User. Содержит дополнительную информацию о зарегистрированных игроках, включая пол, возраст, город, семейное положение, наличие детей. Данные профайла заполняются исключительно об игроках.

Таблица *PlayerQuest* связывает игрока с заданием в тот момент, когда игрок запрашивает новое задание. Таблица фиксирует даты начала и завершения задания, а также понравилось ли задание игроку.

Таблица *Report* содержит ссылки на социальные сети (instagram), в которых игрок выкладывает фото- и видеоматериалы, свидетельствующие о выполнении задания.

## Реализация сервера игры

Сервер игры предназначен для реализации многих алгоритмов обработки данных. На сервер вынесены функции генерации карточки с заданием, алгоритмы обработки профиля игрока, выдачи случайного задания в соответствии с профилем.

Генерация карточек с заданием происходит на основе фона и текста с заданием. Для реализации этой функции были использованы следующие модули библиотеки работы с изображениями PIL:

* Image – содержит методы и свойства для открытия, сохранения и анализа изображений;
* ImageDraw – используется для рисования простой 2D графики;
* ImageFont – содержит функционал для работы с TrueType и OpenType шрифтами.

Генерация карточки с заданием происходит в 2 этапа:

1. наложение стандартного фона на картинку, приложенную художником;
2. наложение текста названия задания (сверху) и описания задания (снизу) на изображение.

Код функции наложения фона представлен на Рис. 6. Для наложения картинки на фон используется метод paste из модуля Image, который вставляет одно изображение в другое. Метод paste принимает два аргумента: путь до вставляемого изображения, и позицию, куда вставляется изображение.

defset\_background\_to\_image(quest\_obj):

background\_path = MEDIA\_ROOT + 'cards/background.png'

background = Image.open(background\_path)

bg\_w,bg\_h=background.size

quest\_img\_path = quest\_obj.card.path

quest\_img = Image.open(quest\_img\_path)

img\_w, img\_h=quest\_img.size

offset = ((bg\_w-img\_w)/2, (bg\_h-img\_h)/2)

background.paste(quest\_img,offset)

background.save(quest\_img\_path)

**Рис. 6.** Код функции наложения фона на картинку

Для наложения текста с заданием на изображения используется метод draw.text, который рисует строку в заданную позицию. Метод draw.text принимает в качестве параметров четыре аргумента: позицию, с которой начинается ввод текста, строка, которая будет напечатана, шрифт и цвет текста. Параметр шрифт определяется при помощи метода ImageFont.truetype, в который передается путь до файла со шрифтом и размер шрифта. Код функции наложения текста на изображение приведен на Рис. 7.

def add\_text\_to\_card(quest\_obj):

quest\_card\_path = quest\_obj.card.path

quest\_card = Image.open(quest\_card\_path)

(MAX\_W,MAX\_H)=quest\_card.size

draw = ImageDraw.Draw(quest\_card)

#Пишем заголовок

card\_font = "BadScript-Regular.ttf"

font = ImageFont.truetype(MEDIA\_ROOT + card\_font,int(quest\_obj.font\_name))

import textwrap

para=textwrap.wrap(quest\_obj.name, width=40)

line\_dimensions = [draw.textsize(line, font=font) for line in para]

current\_h = int(quest\_obj.position\_name)

for line, (w, h) in zip(para, line\_dimensions):

draw.text(((MAX\_W - w) // 2, current\_h), line, font=font,

fill="#aa0000")

current\_h += h

#Пишем содержание

font = ImageFont.truetype(MEDIA\_ROOT + card\_font,int(quest\_obj.font\_text))

para=textwrap.wrap(quest\_obj.text, width=38)

line\_dimensions = [draw.textsize(line, font=font) for line in para]

current\_h = int(quest\_obj.position\_text)

for line, (w, h) in zip(para, line\_dimensions):

draw.text(((MAX\_W - w) // 2, current\_h2), line, font=font,

fill="#aa0000")

current\_h2 += h

quest\_card.save(quest\_card\_path)

**Рис. 7.** Код функции наложения текста на изображения

В результате выполнения этих функций сервером будет сгенерирована карточка с заданием на основе исходной картинки, текста с заданием и фонового изображения. Пример такой карточки представлен на Рис. 8.



**Рис. 8.** Пример сгенерированной карточки с заданием

Пример исходной картинки для этой карточки с заданием представлен на Рис. 9.



**Рис. 9.** Пример картинки к заданию игры

Для того чтобы предоставлять пользователю адекватные и непротиворечивые задания, необходимо анализировать его профиль. Анализ профиля происходит на основе заполненной игроком информации о себе (город, возраст, пол, семейное положение и т.д.). В результате анализа профиля, сервер связывает с игроком определенный набор тегов (например, isMen, 18+, hasPartner и др.). Анализ профиля сервером происходит каждый раз, когда пользователь изменять информацию о себе и нажимает кнопку «сохранить».

Те же самые теги выставляет модератор заданиям, в тот момент, когда задания допускаются к игре. В результате чего сервер может сопоставлять теги, связанные с каждым заданием с тегами, присвоенными игрокам, и на их основе выдавать игрокам только те задания, которые соответствуют и не противоречат профилю игрока.

Задания для каждого игрока выбираются из списка заданий, допущенных к игре, и выдаются игроку случайным образом. Причем задания для игрока обязательно должны соответствовать его профилю. Игрок может получить одно и то же задание несколько раз, только в том случае, если подобное задание у него отмечено как завершенное.

На Рис. 10 представлен код функции выдачи случайного задания get\_random\_quests\_id(), которой в качестве параметра передается идентификатор игрока. Поэтому идентификатору функция анализирует профиль игрока, а также проверяет, какие задания игрок выполняет на данный момент. На основании этих данных функция возвращает идентификатор нового задания для игрока. Если на данный момент в игре нет новых заданий, подходящих данному игроку, функция возвращает ему соответствующий ответ.

def get\_random\_quest\_id(self, player\_id):

random\_quest\_ids=Quest.objects.values\_list('id',flat=True).order\_by('id').

filter(is\_moderated = True)

player\_game\_id = Player\_game\_info.objects.all().get(player=player\_id)

not\_done\_quest\_ids=Player\_Quest.objects.values\_list('questID', flat=True).

order\_by('questID').filter(playerID=player\_game\_id,

is\_done = False)

not\_done\_quest\_ids = list(set(not\_done\_quest\_ids))

set\_random\_quets\_ids = set(random\_quest\_ids)

set\_not\_done\_quets\_ids = set(not\_done\_quest\_ids)

ids = list(set\_random\_quets\_ ^ set\_not\_done\_quets\_ids)

try:

return choice(ids)

except:

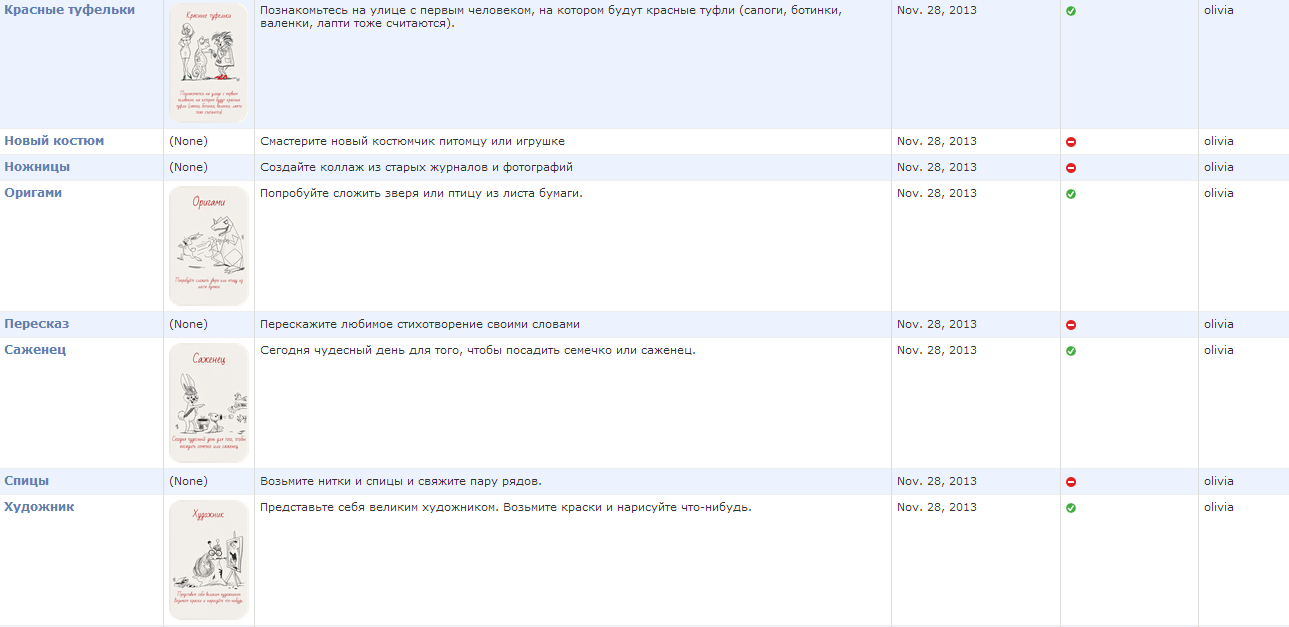
return 0

**Рис. 1015.**Код функции выдачи случайного задания

## Интерфейс администратора

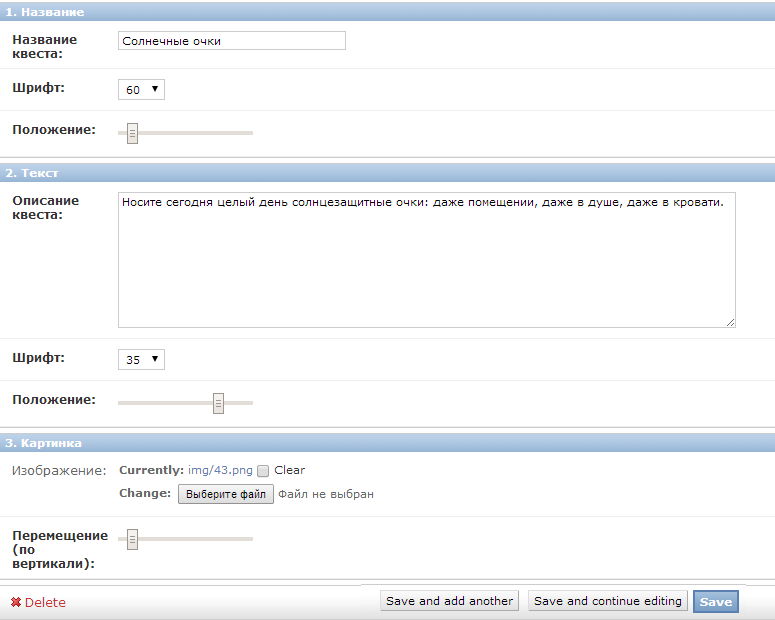
Стандартный интерфейс администратора для добавления, изменения и удаления объектов в/из базы данных предоставляется платформой Django. Интерфейс администратора, необходимый для нашей игры должен делать несколько больше, чем просто взаимодействовать с базой данных. Поэтому стандартный административный интерфейс был изменен под условия нашего приложения путем внесения новых модулей и изменений в исходники платформы Django.

Для возможности более удобного администрирования существующего контента необходимо иметь возможность просматривать сгенерированные карточки с заданиями. Для отображения картинок в административном интерфейсе на страницах просмотра списка всех заданий и редактирования заданий был написан ряд функций, которые встраивают html код в код подпрограмм на python. Пример административного интерфейса с таким функционалом представлен на Рис. 11.



**Рис. 11.** Страница просмотра списка заданий

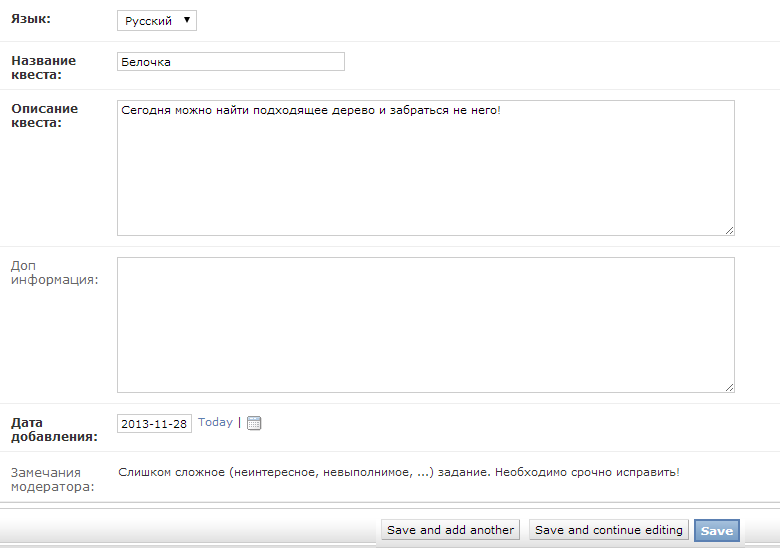
В случае если сервер автоматически сгенерировал карточку с заданием не достаточно хорошо, художник должен иметь возможность изменять положение элементов картинки по горизонтали, а также изменять размер шрифта. Интерфейс редактирования карточки представлен на Рис. 12.



**Рис. 12 .**Интерфейс редактирования карточки

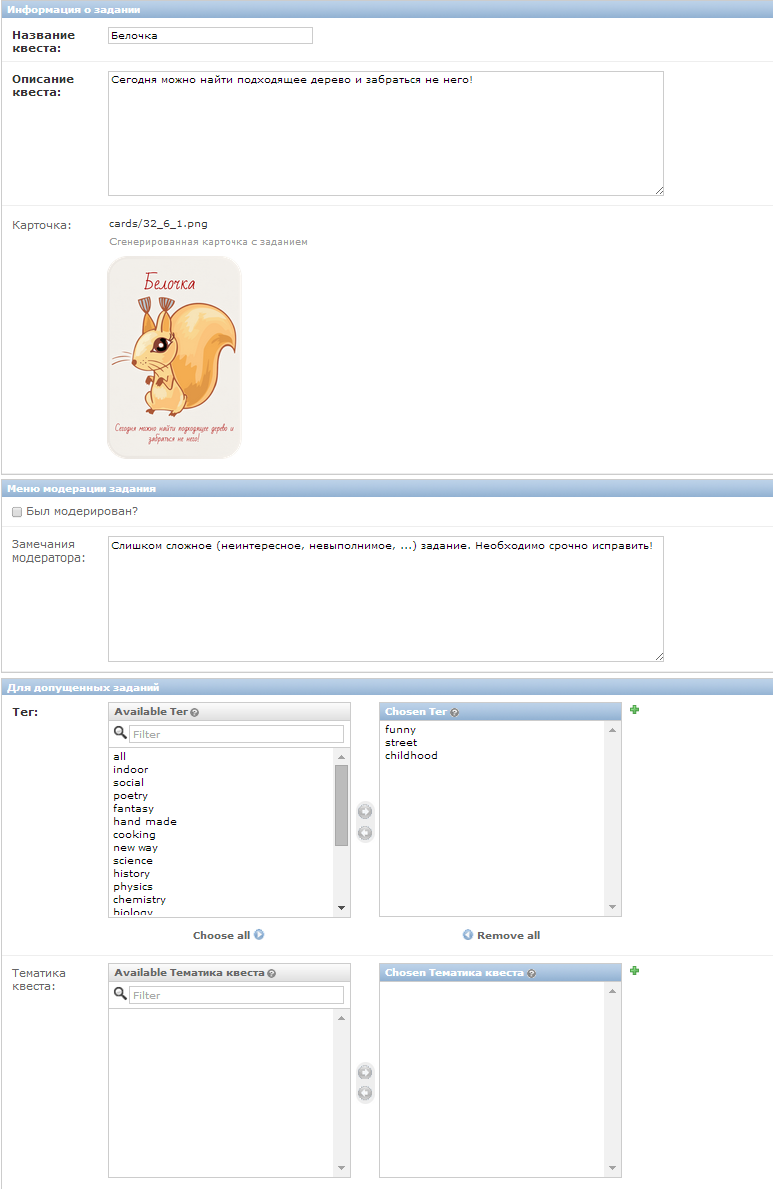
Разные группы пользователей могут иметь разные полномочия в отношении одной и той же сущности. Для реализации этой возможности были использованы Meta-классы, которые определяют дополнительные настройки для модели и позволяют одну и ту же модель отображать иначе (например, скрывать и добавлять поля модели) для различных групп пользователей.

На Рис. 13 изображено представление модели Quets для администраторов игрового контента. Администраторы контента могут только добавлять и редактировать информацию о задании: название, текст задания, дополнительная информация, язык. Администраторы могут прочитать замечания от модератора, если таковы имеются, но редактировать это поле администраторы не могут. Каждый администратор игрового контента может просматривать только список заданий, созданных им самим.



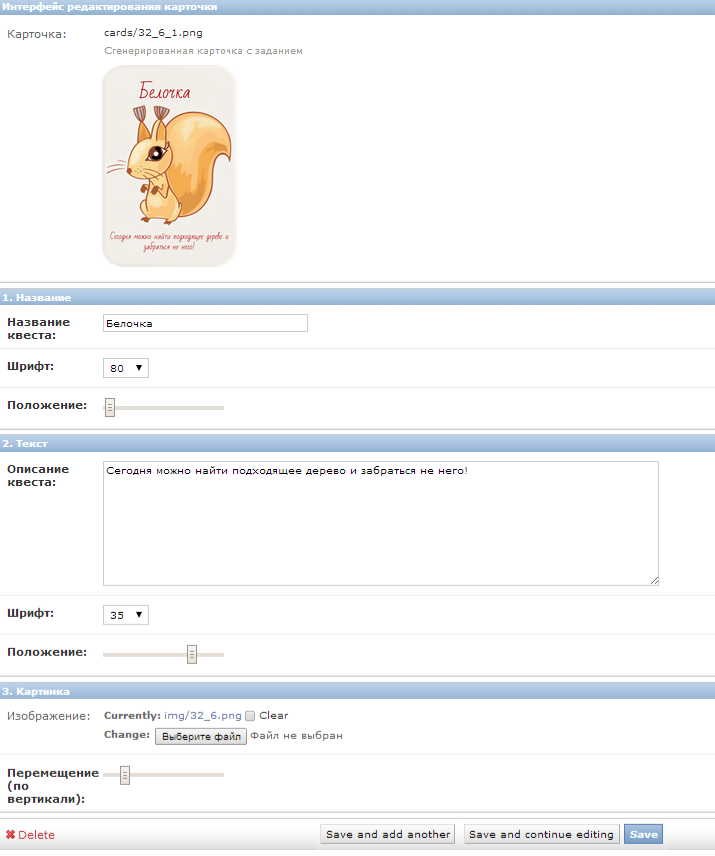
**Рис. 13.** Отображение модели Quests для администраторов игрового контента

На Рис. 14 показано отображение модели Quets для модераторов игры. Модераторы могут просматривать список всех заданий, существующих в игре и редактировать любой из них. Создавать новые задания самостоятельно модераторы не могут. В случае, если задание не понравилось модератору, он может отправить его создателю сообщение, содержащее свои замечания, и отправить задание на доработку. Если задание модератором одобрено, то он добавляет к нему список тегов и выставляет флаг «был модерирован», тогда задание будет допущено к игре.



**Рис. 14.** Отображение модели Quests для модераторов

На Рис. 15 представлено отображение модели Quests для художников игры. Художники добавляют к заданию рисунок. При нажатии на кнопку «сохранить» сервер генерирует карточку с заданием на основе этого рисунка. Художник может просмотреть сгенерированную сервером карточку (она расположена на странице ниже) и изменить положение элементов на рисунке, а также шрифт заголовка и текста. После нажатия на кнопку «сохранить» карточка будет перегенерирована сервером в соответствии с новыми данными.



**Рис. 15.** Отображение модели Questsдля художников игры

## Реализация REST-сервиса

REST-сервис служит важным связующим звеном между мобильными клиентами и сервером игры и предоставляет REST-API для их взаимодействия.REST-сервис реализует функции создания нового игрока, получения и редактирования профиля игрока, получения игроком нового случайного задания и завершения предыдущего.

Работу с информацией об игроках обеспечивает коллекция ресурсов players. Для регистрации новых игроков необходимо выполнить запрос POST /rest/players/, которому в качестве параметров необходимо передать имя пользователя и пароль. Пример вызова метода представлен на Рис. 16. Если игрок с таким именем еще не зарегистрирован в системе, то операция создания игрока завершится успешно. Если игрок с таким именем уже существует, то сервер выдаст ответ “The user with this user name already exist” с HTTP-кодом ошибки 400 – «BAD REQUEST». Вместе с созданием нового игрока в таблице User создается новая пустая запись (профиль), связанная с этим игроком, в таблице Player\_game\_info, которая будет хранить дополнительную информацию об игроке.

POST http://12-labors.com/rest/players/ -d 'username = new\_rest\_player'

-d 'password = rest\_password'

**Рис. 16.** Пример вызова метода создания нового игрока

Авторизация игроков для обеспечения им доступа к некоторым запрошенным ресурсам происходит на основе токенов. Генерация токенов происходит при помощи метода obtain\_auth\_token, который предоставляет Django Rest Framework. Методу obtain\_auth\_token соответствует URL rest/api-token-auth/(см. Рис. 21). Пользователь может получить свой токен, используя следующий метод, передав ему в качестве параметра значение auth\_data, которые генерируется на основе пары логин-пароль и использует алгоритм кодирования base64 (см. Рис. 17). В случае если такой пары логин-пароль не существует, пользователю будет выдано соответствующее сообщение об ошибке.

POST http://12-labors.com/rest/api-token-auth/ -d auth\_data = 'cGxheWVyXzEgcGxheWVyXzFfcGFzc3dvcmQ='

**Рис. 17.** Пример вызова метода для получения токена

Игрок может изменить данные своего профиля. Для этого необходимо вызвать метод PUT, передав ему в качестве параметров свой токен, а также список полей для изменения (см. Рис. 18). При вызове метода PUT/rest/profile/ в адресном пути не указывается идентификатор игрока, так как он неявно определяется через ключ авторизации, который обязательно передается внутри запроса. Для загрузки картинок необходимо использовать ключ –F, который возвращает все загруженные файлы, присутствующие в теле запроса.

PUT http://12-labors.com/rest/profile/ -H "Content-Type:multipart/form-data" -H "Authorization: Tokenb3f6363583a06a49c9b86f9cdca2eeaf25b35c9e" -F "photo=@/Users/photo\_1.png" –F age=23 –F has\_child=True

**Рис. 18.** Пример вызова метода для изменения профиля игрока

Работу с информацией об игроках обеспечивает коллекция ресурсов quests. Для получения случайного задания для игрока необходимо вызвать метод GET/rest/quests/randomQuest, передав ему в качестве параметра токен игрока (см. Рис. 19). Get-метод возвращает новое задание, соответствующее профилю игрока, и создает фиксирует связь игрока с этим заданиям, путем создания новой записи в таблице Player\_Quest.

GET http://12-labors.com/rest/quests/randomQuest/ -H 'Authorization: Token 0ab8f709b81d14164663f0f1a3890f901a712cdd'

**Рис. 19.** Пример вызова метода получения случайного здания

Чтобы окончить выполнение задание, игроку необходимо выставить флаг is\_done=True, свидетельствующий о его завершении. Для этого необходимо вызвать метод PUT /rest/quests/{id}/, где {id} – идентификатор завершаемого задания. Пример вызова данного метода представлен на Рис. 20.

PUT http://12-labors.com/rest/quests/2/ -H "Authorization: Token 708d1b0a9afcabf51718c06ffb175bb6c5fd3952" –d is\_done=true

**Рис. 20.** Пример вызова метода для завершения задания

Все URL-адреса, используемые для вызова методов REST-сервиса, хранятся в файле rest/urls.py (см. Рис. 21). В файле urls.py определяется соответствие между URL-адресами и обрабатывающим их кодом. Функция create\_player и классы RandomQuest, PlayetQuest, Profile импортируются из файла rest/views.py.

urlpatterns = patterns('',

url(r'^quests/randomQuest/', RandomQuest.as\_view()),

url(r'^quests/(?P<pk>[0-9]+)/$', PlayerQuest.as\_view()),

url(r'^players/', create\_player),

url(r'^profile/', Profile.as\_view()),

url(r'^api-token-auth/',

'rest\_framework.authtoken.views.obtain\_auth\_token')

)

**Рис. 21.**Файл urls.py REST-сервиса

Код функции create\_player создания нового игрока представлен на Рис. 22.

@api\_view(['POST'])

def create\_player(request):

model = get\_user\_model()

username, password = request.POST['username'], request.POST['password']

if (User.objects.filter(username = username).count() == 0):

user = User.objects.create\_user(username, u'', password)

Group.objects.get(name='Players').user\_set.add(user)

serializer = UserSerializer(user)

profile = Player\_game\_info.objects.create(player = user)

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response('There is a player with this

username',status=status.HTTP\_400\_BAD\_REQUEST)

**Рис. 22.** Функция создания нового игрока

Реализация REST-методов работы с профилем игрока, получения случайного задания и завершения задания расположена в классах Profile, RandomQuest и PlayerQuest соответственно и представлена на Рис. 23, Рис. 24, Рис. 25.

class Profile(APIView):

authentication\_classes = (TokenAuthentication,)

permission\_classes = (IsAuthenticated,)

def get\_player\_id(self):

token = self.request.META['HTTP\_AUTHORIZATION'].split(' ')[1]

return Token.objects.all().get(key = token).user.id

def get(self, request, format=None):

player\_id = self.get\_player\_id()

try:

profile = Player\_game\_info.objects.get(player = player\_id)

except Player\_game\_info.DoesNotExist:

return Response("Profile does not exist",

status=status.HTTP\_400\_BAD\_REQUEST)

serializer = ProfileSerializer(profile)

return Response(serializer.data)

def put(self, request, format=None):

player\_id = self.get\_player\_id()

profile = Player\_game\_info.objects.get(player = player\_id)

serializer = ProfileSerializer(profile, data=request.DATA,

files=request.FILES)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors,

status=status.HTTP\_400\_BAD\_REQUEST)

**Рис. 23.** Реализация REST-методов класса Profile для работы с профилем игрока

class RandomQuest(APIView):

authentication\_classes = (TokenAuthentication,)

permission\_classes = (IsAuthenticated,)

def get\_player\_id(self):

token = self.request.META['HTTP\_AUTHORIZATION'].split(' ')[1]

return Token.objects.all().get(key = token).user.id

def get(self, request, format = None):

player\_id = self.get\_player\_id()

quest\_id = QuestRandomizer().get\_quest\_id\_for\_player(player\_id)

try:

quest\_game\_id = Quest.objects.all().get(id=quest\_id)

except:

return Response('No new quest for you',

status=status.HTTP\_400\_BAD\_REQUEST)

player\_game\_id = Player\_game\_info.objects.all().get(player=player\_id)

player\_quest = Player\_Quest.objects.create(playerID = player\_game\_id,

questID = quest\_game\_id)

serializer = QuestSerializer(Quest.objects.all().filter(id=quest\_id))

return Response(serializer.data)

**Рис. 24.** Реализация REST-методов класса RandomQuest для получения случайного квеста

class PlayerQuest(APIView):

authentication\_classes = (TokenAuthentication,)

permission\_classes = (IsAuthenticated,)

def get\_player\_id(self):

token = self.request.META['HTTP\_AUTHORIZATION'].split(' ')[1]

return Token.objects.all().get(key = token).user.id

def get(self, request, pk, format=None):

player\_id = self.get\_player\_id()

player\_game\_id = Player\_game\_info.objects.all().get(player=player\_id)

try:

player\_quest = Player\_Quest.objects.get(playerID =

player\_game\_id, questID=pk, is\_done = False)

except Player\_Quest.DoesNotExist:

return Response('Quest has been completed',

status=status.HTTP\_400\_BAD\_REQUEST)

serializer = PlayerQuestSerializer(player\_quest)

return Response(serializer.data)

def put(self, request, pk, format=None):

player\_id = self.get\_player\_id()

player\_game\_id = Player\_game\_info.objects.all().get(player=player\_id)

try:

player\_quest = Player\_Quest.objects.get(playerID =

player\_game\_id,questID = pk, is\_done=False)

except Player\_Quest.DoesNotExist:

return Response('Quest has been completed',

status=status.HTTP\_400\_BAD\_REQUEST)

player\_quest.date\_done = datetime.datetime.today()

serializer = PlayerQuestSerializer(player\_quest, data=request.DATA,

files=request.FILES)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

**Рис. 25.** Реализация REST-методов класса PlayerQuest для завершения задания

## Вывод

В результате была выполнена реализация всех компонентов сервера игры «12 Подвигов» в соответствии с функциональными и нефункциональными требованиями, предъявляемыми к системе. В том числе был реализован REST-сервис, предоставляющий необходимый REST API для работы клиентского приложения, и выполнена настройка интерфейса администратора.

# Тестирование

## Модульное тестирование

*Модульное тестирование (или unit-testing)*–это тестирование отдельных модулей исходного кода программы. Unit-тесты пишутся для каждой нетривиальной задачи или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение этих ошибок. Цель модульного тестирования — изолировать отдельные части программы и показать, что эти части способны работать корректно, независимо друг от друга.

Для написания модульных тестов использовался модуль django unit tests. Библиотека для автоматизированного тестирования включает в себя следующие элементы:

* TestCase – модуль тестирования. Проверка валидности ответа для определенного набора данных.
* TestSuite – содержит в себе тесты, который должны быть запущены вместе.
* TestFixture – содержит в себе предварительные настройки для тестов (соединение с БД, создание временных директорий).
* TestRunner/Loader – модуль, запускающий тесты и выводящий результаты в нужном формате.

Unit-тесты были написаны для проверки правильности работы функций создания, редактирования и удаления моделей, работы с изображениями и профилем игрока.

Проверка работы функций создания, редактирования и удаления моделей на примере модели Quests приведена на Рис. 26. Для тестирования этих функций создан класс QuestObjectTest. Методы setUp() и tearDown() стандартны для всех создаваемых классов тестов, наследуемых от класса unittest.TestCase. Метод setUp() вызывается автоматически перед каждым методом в тесте. В нем производятся необходимые предварительные настройки, например настройки базы данных. Метод tearDown() вызывается автоматически после каждого метода в тесте, даже в тех случаях, если тест не прошел. Метод будет вызван, если предварительный запуск setUp() прошел успешно. При тестировании моделей тест раннер обеспечивает создание новой тестовой базы данных. Любой тест, который взаимодействует с базой данных, никак не повлияет на реальную базу данных. Независимо от того, прошли тесты успешно или нет, тестовая база данных уничтожается в конце процесса тестирования.

class QuestObjectTest(unittest.TestCase):

def setUp(self):

self.quest1 = Quest.objects.create(name="First", text="First Quest

for the game", img="img/1.png",date="2013-12-12",

rating="3")

deftearDown(self):

self.quest1=None

def testSaveObject(self):

self.quest1.save()

obj = Quest.objects.get(name="First")

self.assertEqual(obj.name, 'First')

def testDeleteOblects(self):

self.quest1.save()

Quest.objects.filter(name="First").delete()

self.assertNotEqual(Quest.objects.filter(name="First"), 'First')

**Рис. 26.** Пример модульного теста для проверки корректности работы модели Quest

Тесты класса ImageTest проверяют корректность функций работы с изображениями: загрузка картинки, редактирование, переименование и удаление (см. Рис. 27). При изменении или удалении загружаемой картинки важно не просто ее заменить/удалить в модели, но и физически удалить ее из папки media/img, в которой она расположена.

class ImageTest(unittest.TestCase):

def setUp(self):

self.quest1 = Quest.objects.create(id=1, name="FirstQuest",

text="First Quest for the game", img="5894\_59.png",

date="2013-12-27", rating="3")

self.quest2 = Quest.objects.create(id=2, name="FirstQuest",

text="First Quest for the game", img="fhdh53ib.png",

date="2013-12-27", rating="4")

self.quest3 = Quest.objects.get(id=1)

def tearDown(self):

self.quest1=None

self.quest2=None

self.quest3=None

def testImageSave(self):

self.quest1.img = "card/2783.png"

self.quest1.save()

self.assertEqual(self.quest1.img, 'card/2783.png')

self.assertNotEqual(Quest.objects.filter(img="5894\_59.png"),

'5894\_59.png')

def testImageDelete(self):

self.quest1.img = "card/2783.png"

self.quest1.delete()

self.assertEqual(self.quest1.img, None)

self.assertNotEqual(Quest.objects.filter(img="2783.png"),

'2783.png')

def testImageNameAndUrl(self):

self.assertEqual(self.quest1.img.name, '5894\_59.png')

self.assertEqual(self.quest1.img.url, 'media/5894\_59.png')

self.quest1.img = get\_image\_path(self.quest1, self.quest1.img.name)

self.quest1.save()

self.assertEqual(self.quest1.img.name, 'img/1.png')

self.assertEqual(self.quest1.img.url, 'media/img/1.png')

self.assertEqual(self.quest2.img.name, 'fhdh53ib.png')

self.assertEqual(self.quest2.img.url, 'media/fhdh53ib.png')

self.quest2.img = get\_image\_path(self.quest2, self.quest2.img.name)

self.quest2.save()

self.assertEqual(self.quest2.img.name, 'img/2.png')

self.assertEqual(self.quest2.img.url, 'media/img/2.png')

**Рис. 27.** Класс тестов для проверки функций работы с изображениями

Для запуска тестов используется команда python manage.py test. По умолчанию эта команда запустит все тесты каждого приложения проекта. Чтобы запустить тесты только для одного приложения, необходимо добавить имя этого приложения в команду, например python manage.py test quest.

Все написанные тесты для проекта сервера игры для каждого приложения были пройдены успешно.

## Тестирование REST-сервиса

Тестирование методов, предоставляемых REST-сервисом, производилось при помощи ручного тестирования. Для проведения тестирования была настроена специально сконфигурированная виртуальная машина, предоставляющая возможность работы с реальными данными. Тестирование REST-методов производилось при помощи утилиты командной строки cURl (clients URLs). Инструмент cURLпозволяет передавать файлы и данные синтаксисом URL. Данная технология поддерживает множество протоколов, таких как HTTP, FTP, TELNET и др.

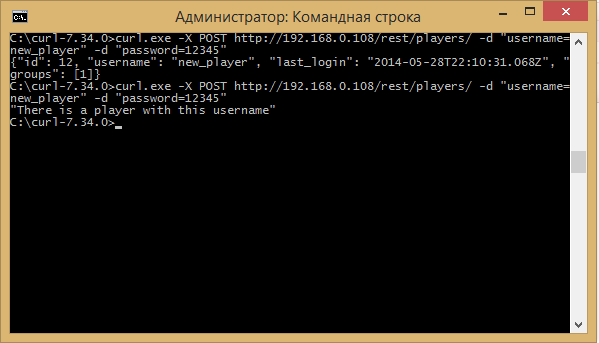
### Регистрация новых игроков

Для регистрации новых игроков используется метод post /rest/players/. Тестирование работы метода представлено в Табл. 1.

**Табл. 1.** Тестирование методов создания игроков

|  |  |  |  |
| --- | --- | --- | --- |
| ***Состояние сервера*** | ***Действие*** | ***Ожидаемый результат*** | ***Действительный результат*** |
| В базе данных не существует игрока с именем пользователя «new\_rest\_user\_1». | curl.exe –X POST http://192.168.0.107/rest/players/ -d “username=new\_rest\_user\_1”–d “password=password\_1” | Создание игрока с заданной парой логин-пароль. | {"id":17, "username": "new\_rest\_user\_1","last\_login": "2014-05-22T05:35:00.355  Z", "groups": [3]} |
| В базе данных уже существует игрок с именем «new\_rest\_user\_1». | curl.exe –X POST <http://192.168.0.107/rest/players/> -d “username=new\_rest\_user\_1” –d “password=password\_1” | There is the player with this username. | There is the player with this username. |

Результаты тестирования метода изображены на Рис. 28.



**Рис. 28.** Результаты тестирования метода создания игроков

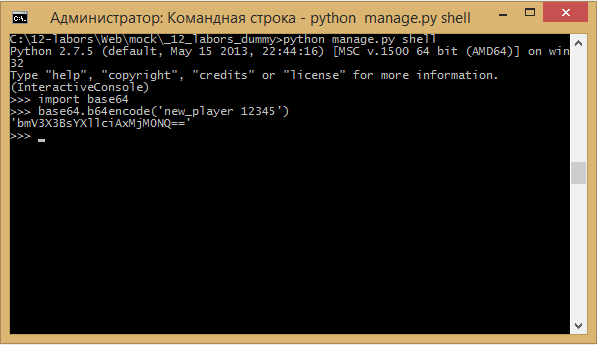
### Авторизация игроков

Для авторизации при помощи механизма токенов игроков используется метод post/rest/api-token-auth/.Тестирование работы метода представлено в Табл. 2.

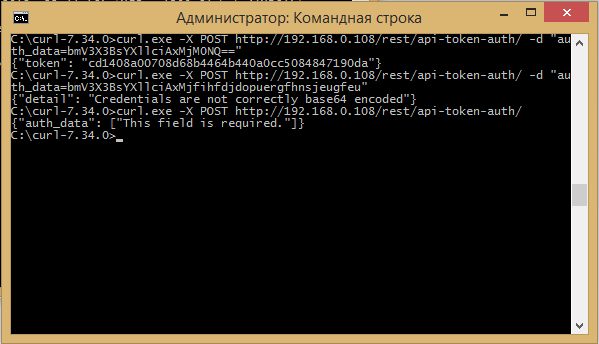
**Табл. 2.** Тестирование методов получения токена для игроков

|  |  |  |  |
| --- | --- | --- | --- |
| ***Состояние сервера*** | ***Действие*** | ***Ожидаемый результат*** | ***Действительный результат*** |
| Пользователь с парой логин-пароль, переданной в запросе существует в базе данных. | base64.encode(‘new\_rest\_user 12345’) | Сервер возвратит значение auth\_data, соответствующее этой паре логин-пароль. | bmV3X3Jlc3RfdXNlciAxMjM0NQ== |
| Пользователь с парой логин-пароль, переданной в запросе не существует в базе данных. | base64.encode(‘new\_rest\_user 12345678’) | Неверное значение пары логин-пароль. | Сообщение об ошибке. |
| Значение auth\_dataсоответствуют одной из пар логин-пароль, реально существующих в базе данных. | curl.exe -X POST http:// 192.168.0.108/rest/api-token-auth/ -d “auth\_data= bmV3X3Jlc3RfdXNlciAxMjM0NQ==” | Сервер вернет токен, соответствующий заданному значению auth\_data. | 522bd169d357e08331bcad97132f47c90188d81e |
| Значение auth\_dataнесоответствуют ни одной из пар логин-пароль. | curl.exe -X POST http:// 192.168.0.108rest/api-token-auth/ -d “auth\_data= bmV3X3Jlc3RfdXNlciArofjkrfg” | Сервер вернет сообщение об ошибке. | Credentials are not correctly base64 encoded. |

Результаты тестирования метода изображены на Рис. 29, Рис. 30.



**Рис. 29.** Получение значения auth\_data



**Рис. 30.** Результаты тестирования метода получения токена игрока

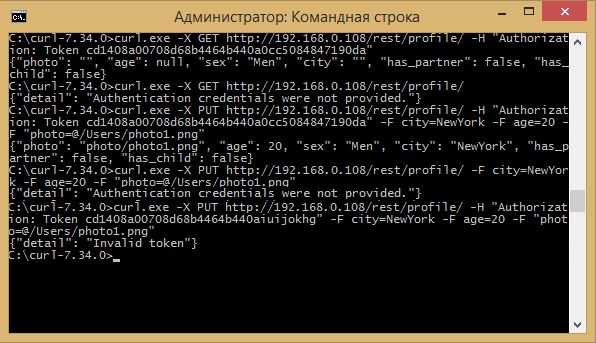
### Редактирование профиля

Для редактирования профиля игроков используется метод /rest/profile/. Тестирование работы метода представлено в Табл. 3.

**Табл. 3.** Тестирование методов работы с профилем игрока

|  |  |  |  |
| --- | --- | --- | --- |
| ***Состояние сервера*** | ***Действие*** | ***Ожидаемый результат*** | ***Действительный результат*** |
| В базе данных существует пустой профиль игрока на момент первого изменения профиля. | curl.exe -X PUT http:// 192.168.0.108rest/profile/  -H "Authorization: Token 522bd169d357e08331bcad97132f47  c90188d81e" -F age=19 -F city=Moscow -F "photo=@/Users/DataBase.png" | Данные профиля игрока будут изменены в соответствии с данными, указанными в запросе. | {"photo": "photo/DataBase\_2.png", "age": 19, "sex": "Men", "city": "Moscow", "ha  s\_partner": false, "has\_child": false} |
| В базе данных существует игрок с заданным значением токена. | curl.exe -X GET http:// 192.168.0.108/rest/profile/ -H "Content-T  ype:multipart/form-data" -H "Authorization: Token 522bd169d357e08331bcad97132f47  c90188d81e" | Игрок сможет просмотреть данные своего профиля. | {"photo": "photo/DataBase\_2.png", "age": 19, "sex": "Men", "city": "Moscow", "ha  s\_partner": false, "has\_child": false} |
| Игрок не авторизован. | curl.exe -X GET http:// 192.168.0.108/rest/profile/ | Игрок не сможет просмотреть данные своего профиля, поскольку он не авторизован. | Authentication credentials were not provided. |
| В базе данных не существует игрок с заданным значением токена. | curl.exe -X GET http:// 192.168.0.108/rest/profile/ -H "Authorization: Token fjkn38y4hr" | Игрок не сможет просмотреть данные своего профиля, поскольку он указал неверный токен для авторизации. | Invalid token. |

Результаты тестирование метода изображены на Рис. 31.



**Рис. 31.** Результаты тестирования методов работы с профилем игрока

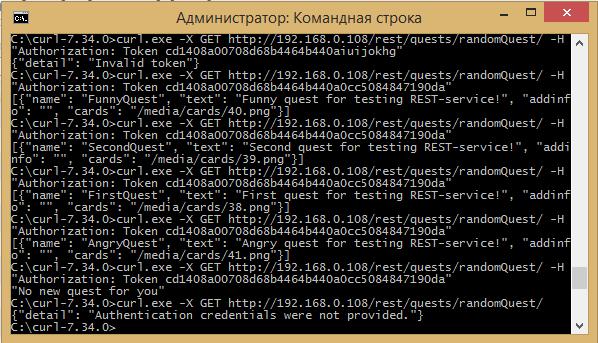
### Получение случайного задания

Для получения случайного задания для игроков используется метод /quests/randomQuest/. Тестирование работы метода представлено в Табл. 4.

**Табл. 4.** Тестирование метода получения случайного задания

|  |  |  |  |
| --- | --- | --- | --- |
| ***Состояние сервера*** | ***Действие*** | ***Ожидаемый результат*** | ***Действительный результат*** |
| В базе данных существуют задания, допущенные к игре и соответствующие профилю данного игрока. | curl.exe -X GET http:// 192.168.0.108/rest/quests/randomQuest/ -H "Authorization:Token 522bd169d357e08331bcad97132f47c90188d81e" | Сервер выдаст игроку новое задание. | [{"name": "Quest5", "text": "New Quest for game", "addinfo": "", "cards": "/medi  a/cards/5.png"}] |
| В базе данных существуют задания, допущенные к игре и соответствующие профилю данного игрока. | curl.exe -X GET http:// 192.168.0.108/rest/quests/randomQuest/ -H"Authorization:Token 522bd169d357e08331bcad97132f47c90188d81e" | Сервер выдаст игроку сообщение об отсутствии новых заданий. | No new quests for you. |
| В базе данных не существует игрока с подобным значением токена. | curl.exe -X GET http:// 192.168.0.108/rest/quests/randomQuest/ -H"Authorization:Token 522bd169d357e08331bcad" | Сообщение об ошибке. Некорректное значение токена. | Invalid token. |
| Игрок пытается получить задание без авторизации. | curl.exe -X GET http:// 192.168.0.108/rest/quests/randomQuest/ | Сообщение об ошибке. Неавторизованный пользователь. | Authentication credentials were not provided. |

Результаты тестирование метода изображены на Рис. 32.



**Рис. 32.** Результаты тестирования метода получения случайного задания

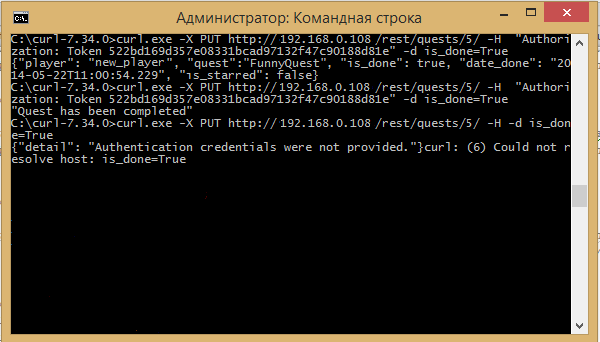
### Завершение задания

Для завершения заданий, выполненных игроками, используется метод /quests/{id}/. Тестирование работы метода представлено в Табл. 5.

**Табл. 5.** Тестирование метода завершения задания

|  |  |  |  |
| --- | --- | --- | --- |
| ***Состояние сервера*** | ***Действие*** | ***Ожидаемый результат*** | ***Действительный результат*** |
| Задание с идентификатором, переданным в запросе, существует в базе данных и отмечено как незавершенное. | curl.exe -X PUT http://192.168.0.108/rest/quests/4/ -H "Authori  zation: Token 522bd169d357e08331bcad97132f47c90188d81e" -d is\_done=True | Задание будет завершено. | {"player": "new\_rest\_user", "quest": "Quest5", "is\_done": true, "date\_done": "20  14-05-22T11:00:54.229", "is\_starred": false} |
| Задание с идентификатором, переданным в запросе, существует в базе данных и уже отмечено как завершенное. | C:\curl-7.34.0>curl.exe -X PUT http://192.168.0.108 /rest/quests/4/ -H "Authori  zation: Token 522bd169d357e08331bcad97132f47c90188d81e" -d is\_done=True | Сообщение об ошибке. Задание с этим id уже завершено. | Quest has been completed. |
| Игрок пытается завершить задание без авторизации или игрок с переданным в запросе токеном не существует. | C:\curl-7.34.0>curl.exe -X PUT http://192.168.0.108 /rest/quests/4/ -H  -d is\_done=True | Сообщение об ошибке. Нет авторизации игрока. | Authentication credentials were not provided. |

Результаты тестирование метода изображены на Рис. 33.



**Рис. 33.** Результаты тестирования метода завершения задания

## Вывод

В результате было выполнено автоматизированное тестирование всех нетривиальных методов работы сервера. Автоматическое тестирование приложений является одним из полезных методов для решения ряда проблем при написании нового кода и модификации старого. С помощью тестов код проверяется на наличие ошибок и достижения ожидаемого результата. Общее покрытие исходного кода проекта модульными тестами составляет 60%. Также было выполнено ручное тестирование REST-сервиса. Все предложенные REST-сервису тесты были выполнены успешно.

# Заключение

В рамках выполнения данного проекта, мною были решены следующие задачи:

1. изучен язык программирования Python, платформа Django;
2. изучены основы архитектуры REST, библиотеку django-rest-framework;
3. выполнен анализ требований к системе;
4. выполнено проектирование системы;
5. реализована система в соответствии с установленными требованиями;
6. выполнено модульное и функциональное тестирование системы.

В ходе проделанной работы мной были изучены возможности разработки сервера мобильных игр, включая облачные технологии создания бэк-энд решений.

В процессе работы над дипломным проектом была спроектирована архитектура сервера игры «12 Подвигов» и полностью реализованы все компоненты сервера в соответствии с поставленными функциональными и нефункциональными требованиями и техническим заданием.

Было произведено модульное тестирование сервера для проверки правильности работы отдельных функций, методов и классов внутри проекта. Было также проведено интеграционное тестирование системы для проверки взаимодействия работы сервера игры с клиентским приложением. Все предложенные тесты были пройдены успешно.

Разработанный в рамках дипломной диссертации сервер планируется развивать и оснащать новым функционалом и новыми возможностями.

# Литература

1. Головатый А., Каплан-Мосс Д. Django. Подробное руководство, 2-е издание. Пер. с англ. СПб: Символ-плюс, 2013. 560 с.
2. Чан У., Биссекс П., Форсье Д. Django. Разработка веб-приложений на Python. СПб: Символ-плюс, 2009. 456 с.
3. Саммерфилд М. Программирование на Python 3. Подробное руководство. Пер. с англ. СПб: Символ-плюс, 2011. 608 с.
4. Richardson L., Ruby S. RESTful Web Services. O’Reilly, 2007. 440 c.
5. Официальный веб-сайт платформы DjangoRest [Электронный ресурс]. URL: [http://www.django-rest-framework.org](http://www.django-rest-framework.org/)
6. Документация Django на русском языке [Электронный ресурс]. URL: <http://djbook.ru/>
7. Официальный сайт облачной платформы Windows Azure [Электронныйресурс].URL: <http://www.windowsazure.com/en-us/solutions/mobile/>
8. Официальный сайт облачного сервиса Windows Azure Mobile Services [Электронный ресурс]. URL: <http://www.windowsazure.com/en-us/services/mobile-services/>
9. Официальный сайт облачной платформы Amazon [Электронный ресурс]. URL: <http://aws.amazon.com/web-mobile-social/>
10. Официальный сайт облачной платформы Google Cloud[Электронный ресурс]. URL: <https://cloud.google.com/solutions/mobile/>
11. Официальный сайт сервиса Google Cloud Endpoints. [Электронный ресурс].URL: <https://cloud.google.com/products/cloud-endpoints/>
12. Официальный сайт сервиса Google Backend Starter [Электронный ресурс]. URL: <https://cloud.google.com/developers/articles/mobile-backend-starter>
13. Документация Django для механизма миграций данных [Электронный ресурс]. URL: <https://docs.djangoproject.com/en/dev/topics/migrations/>
14. Стоунз Р., Меттью Н. PostgreSQL. Основы. СПб: Символ-плюс, 2002. 640 с.
15. Официальный документ исследований компании J’son & Partners Consulting: Mobile Advertising Market, 2011-2016. URL: <http://www.json.ru/files/reports/2012-11-12_Mobile_ADV_MW_EN.pdf>
16. Robert Daigneau. Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services. Addison-Wesley Professional. 2011 - 352 p.