

Модуль 5. Основы разработки серверной части мобильных приложений

Тема 5.4. Облачные платформы. REST взаимодействие

4 часа

Оглавление

5.4. Облачные платформы. REST взаимодействие	2
5.4.1. Облачные технологии	2
5.4.2. Модели развёртывания	3
5.4.3. Модели обслуживания	4
Программное обеспечение как услуга	4
Платформа как услуга	4
Инфраструктура как услуга	4
5.4.4. Платформа как услуга	5
PaaS для разработчиков.....	5
Основные компоненты PaaS.....	6
Выбор поставщика.....	6
Heroku — облачная PaaS-платформа	7
Задание 5.4.1.....	7
5.4.5. REST взаимодействие.....	7
Принципы REST	8
Retrofit — библиотека для работы с REST API	10
Упражнение 5.4.1	12
5.4.6. REST аутентификация и OAuth авторизация	19
Упражнение 5.4.2	22
Этап 1. Клиент - локальный сервер.....	22
Этап 2. Сервер на облачном сервисе Heroku	35
Источники.....	38
Благодарности	38

5.4. Облачные платформы. REST взаимодействие

5.4.1. Облачные технологии

Облачные технологии — информационно-технологическая концепция, подразумевающая обеспечение повсеместного и удобного сетевого доступа к общим вычислительным ресурсам. Например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности.

Для того чтобы лучше представить, что такое облачные технологии, можно привести простой пример: раньше пользователь для доступа в электронную почту прибегал к определенному ПО (мессенджеры и программы), установленному на его ПК, теперь же он просто заходит на сайт той компании, чьи услуги электронной почты ему нравятся, непосредственно через браузер, без использования посредников. Т.е. вся разница заключается исключительно в методе хранения и обработке данных. Если все операции происходят на Вашем компьютере (с использованием его мощностей), то это — не «облако», а если процесс происходит на сервере в сети, то это именно то, что и принято называть «облачной технологией».

Достоинства

- **Доступность.** Доступ к информации, хранящейся в облаке, может получить каждый, кто имеет компьютер, планшет, любое мобильное устройство, подключенное к сети интернет.
- **Мобильность.** У пользователя нет постоянной привязанности к одному рабочему месту. Из любой точки мира менеджеры могут получать отчетность, а руководители — следить за производством.
- **Экономичность.** Одним из важных преимуществ называют уменьшенную затратность. Пользователю не надо покупать дорогостоящие, большие по вычислительной мощности компьютеры и ПО, а также он освобождается от необходимости нанимать специалиста по обслуживанию локальных IT-технологий.
- **Арендность.** Пользователь получает необходимый пакет услуг только в тот момент, когда он ему нужен, и платит, собственно, только за количество приобретенных функций.
- **Гибкость.** Все необходимые ресурсы предоставляются провайдером автоматически.
- **Высокая технологичность.** Большие вычислительные мощности, которые предоставляются в распоряжение пользователя, которые можно использовать для хранения, анализа и обработки данных.
- **Надежность.** Некоторые эксперты утверждают, что надежность, которую обеспечивают современные облачные вычисления, гораздо выше, чем надежность локальных ресурсов, аргументируя это тем, что мало предприятий могут себе позволить приобрести и содержать полноценный дата-центр (от англ. data center), или центр (хранения и) обработки данных (ЦОД/ЦХОД).

Недостатки

- **Необходимость постоянного соединения.** Для получения доступа к услугам «облака» необходимо постоянное соединение с Интернет

- Ограничения ПО. Есть ограничения по ПО, которое можно разворачивать на «облаках» и предоставлять его пользователю. Пользователь имеет ограничения в используемом обеспечении и иногда не имеет возможности настроить его под свои собственные цели
- Конфиденциальность. Конфиденциальность данных, хранимых в публичных «облаках», в настоящее время, вызывает много споров, но в большинстве случаев эксперты сходятся в том, что не рекомендуется хранить наиболее ценные для компании документы на публичном «облаке», так как в настоящее время нет технологии, которая бы гарантировала 100% конфиденциальность данных
- Безопасность. «Облако» само по себе является достаточно надежной системой, однако при проникновении в него злоумышленник получает доступ к огромному хранилищу данных. Еще один минус, — это использование систем виртуализации в которых, в качестве гипервизора, используются ядра стандартных ОС (например, Windows), что позволяет использовать вирусы и уязвимости системы
- Дороговизна оборудования. Для построения собственного облака необходимо выделить значительные материальные ресурсы, что не выгодно только что созданным и малым компаниям
- Дальнейшая монетизация ресурса. Вполне возможно, что компании в дальнейшем решат брать плату с пользователей за предоставляемые услуги.

5.4.2. Модели развёртывания

Частное облако

Частное облако (англ. private cloud) — инфраструктура, предназначенная для использования одной организацией, включающей нескольких пользователей (например, подразделений одной организации), возможно также клиентами данной организации. Частное облако может находиться в собственности как самой организации, так и третьей стороны (или какой-либо их комбинации), и оно может физически существовать как внутри, так и вне юрисдикции владельца.

Общественное облако

Общественное облако (англ. community cloud) — вид инфраструктуры, предназначенный для использования конкретным сообществом пользователей из организаций, имеющих общие задачи. Общественное облако может находиться в совместной собственности одной или более организаций из сообщества или третьей стороны (или какой-либо их комбинации), и оно может физически существовать как внутри, так и вне юрисдикции владельца.

Гибридное облако

Гибридное облако (англ. hybrid cloud) — это комбинация из двух или более различных облачных инфраструктур (частных, публичных или общественных), остающихся уникальными объектами, но связанных между собой стандартизованными или частными технологиями передачи данных и приложений (например, кратковременное использование ресурсов публичных облаков для балансировки нагрузки между облаками).

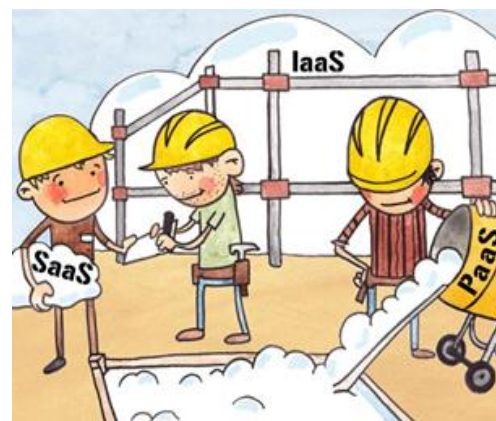


5.4.3. Модели обслуживания

Программное обеспечение как услуга

Программное обеспечение как услуга (SaaS, англ. Software-as-a-Service) — модель, в которой пользователю предоставляется возможность использования прикладного программного обеспечения, работающего в облачной инфраструктуре и доступного из различных клиентских устройств или посредством тонкого клиента, например, из браузера (веб-почта) или посредством интерфейса программы. Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения, или даже индивидуальных возможностей приложения (за исключением ограниченного набора пользовательских настроек конфигурации приложения) осуществляется облачным провайдером.

В качестве примера можно привести Google Docs, Google Calendar и т.п. онлайн-программы.



Платформа как услуга

Платформа как услуга (PaaS, англ. Platform-as-a-Service) — модель, когда пользователю предоставляется возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нём новых или существующих приложений (собственных, разработанных на заказ или приобретённых). В состав таких платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения — системы управления базами данных, связующее программное обеспечение, среды исполнения языков программирования — предоставляемые облачным провайдером.

Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения осуществляется облачным провайдером, за исключением разработанных или установленных приложений, а также, по возможности, параметров конфигурации среды (платформы).

В 2011 году мировой рынок публичных PaaS оценён в сумму около \$700 млн, в числе 10 крупнейших провайдеров указываются Amazon.com (Beanstalk), Salesforce.com (Force.com, Heroku, Database.com), LongJump, Microsoft (Windows Azure), IBM (Bluemix), Red Hat (OpenShift), VMWare (Cloud Foundry), Google (App Engine), CloudBees, Engine Yard

Инфраструктура как услуга

Инфраструктура как услуга (IaaS, англ. Infrastructure-as-a-Service) предоставляется как возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетями и другими фундаментальными вычислительными ресурсами. Например, пользователь может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы, платформенное и прикладное программное обеспечение. Пользователь может контролировать операционные системы, виртуальные системы хранения данных и установленные приложения, а также обладать ограниченным контролем за набором доступных сетевых сервисов (например, межсетевым экраном, DNS). Контроль и управление основной физической и виртуальной инфраструктурой

облака, в том числе сети, серверов, типов используемых операционных систем, систем хранения осуществляется облачным провайдером.

5.4.4. Платформа как услуга

Концепция "Платформа как услуга" вызывает больше всего разночтений, поскольку ее трудно идентифицировать и отличить от концепций "Инфраструктура как услуга" и "Программное обеспечение как услуга". Определяющим фактором уникальности PaaS является то, что она позволяет разработчикам создавать и развертывать Web-приложения на предлагаемой инфраструктуре. Другими словами, PaaS позволяет воспользоваться практически безграничными вычислительными ресурсами облачной инфраструктуры.

Естественно, безграничные вычислительные ресурсы – это иллюзия, поскольку они ограничены размером инфраструктуры. Однако, например, инфраструктура Google, по оценкам, содержит более миллиона компьютеров архитектуры x86.



РaaS для разработчиков

Общее заблуждение разработчиков состоит в том, что облачные вычисления касаются только сетевых администраторов. При этом игнорируются многие возможности, которые облачные вычисления предоставляют разработчикам и отделам обеспечения качества.

Рассмотрим некоторые вопросы, которые часто приводят к проблемам в период разработки программного обеспечения. Например, очень большой проблемой может быть процесс настройки серверной среды, в которой будет размещаться Web-приложение, создаваемое группой разработчиков. Даже на самых крупных предприятиях обычно имеется только один сетевой администратор, обслуживающий все группы разработчиков. Если не использовать PaaS, настройка среды разработки или тестирования обычно требует выполнения следующих действий:

1. Приобрести и развернуть сервер.
2. Установить операционную систему, среду времени исполнения, репозиторий управления исходным кодом и все остальное необходимое программное обеспечение промежуточного уровня.
3. Настроить операционную систему, среду времени исполнения и дополнительное программное обеспечение промежуточного уровня.
4. Перенести или скопировать существующий код.
5. Протестировать и запустить код, чтобы убедиться в корректности работы всей системы.

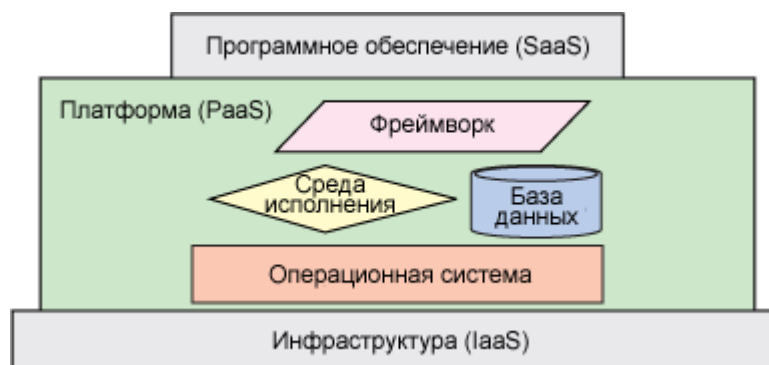
Скорее сетевой администратор уже загружен работой до предела, поэтому развертывание новой среды может представлять собой болезненный процесс. Еще одной большой проблемой для разработчиков Web-приложений (как клиентской, так и серверной части) является дублирование среды времени исполнения на локальном рабочем месте для собственного тестирования.

Теперь представьте, что вы работаете в группе разработчиков, использующей PaaS. В этой ситуации у вас будет виртуальная машина, содержащая полную серверную среду, которую буквально можно переносить на USB-флэшке.

Основные компоненты PaaS

Возможно, лучший способ разобраться в PaaS - это разбиение на главные компоненты: платформу и сервис. Будем рассматривать предоставляемый сервис как стек решений. Логично предположить, что двумя главными компонентами PaaS являются вычислительная платформа и стек решений.

Для лучшего понимания этих двух компонентов рассмотрим их определения. Вычислительная платформа в своем простейшем виде представляет собой место, где может без проблем работать программное обеспечение, если оно отвечает стандартам этой платформы. Типичными примерами платформ являются: Windows™, Apple Mac OS X и Linux® для операционных систем; Google Android, Windows Mobile® и Apple iOS для мобильных вычислений; Adobe® AIR™ или Microsoft® .NET Framework для программных инфраструктур. Важно помнить, что мы не говорим о самом программном обеспечении, а только о платформе, на которой оно должно работать.



Набор решений состоит из приложений, которые будут помогать в процессе разработки и развертывания программ: операционная система, среда времени исполнения, репозиторий управления исходными кодами и любое другое необходимое программное обеспечение промежуточного уровня.

Выбор поставщика

Набор предлагаемых решений – это именно то, чем различаются между собой компании-поставщики PaaS. Его нужно очень тщательно исследовать, прежде чем принимать окончательное решение.

Вот несколько основных вопросов, которые необходимо учесть:

- Какие инфраструктуры и языки поддерживаются?
- Сколько приложений можно создать?
- Какого рода содержимое разрешено? Инфраструктуры, поддерживающие PaaS, обычно оперируют концепцией под названием "вычисления со множественной арендой" (multi-tenant computing), когда много "арендаторов" "живут" на одном сервере, разделенные посредством экземпляров виртуальных машин, управляемых гипервизором. Поставщик PaaS может устанавливать определенные ограничения на виды приложений и содержимого, которые вы планируете разместить.
- Базы данных какого типа поддерживаются?

- Поддерживается ли протокол SSL (HTTPS)?

Heroku — облачная PaaS-платформа

В наших заданиях мы будем использовать Heroku. Это облачная PaaS-платформа, поддерживающая ряд языков программирования. Компанией Heroku владеет Salesforce.com. Heroku, одна из первых облачных платформ, появилась в июне 2007 года и изначально поддерживала только язык программирования Ruby, но на данный момент список поддерживаемых языков также включает в себя Java, Node.js, Scala, Clojure, Python и PHP. На серверах Heroku используются операционные системы Debian или Ubuntu.

Нами была выбрана именно эта PaaS платформа, потому что она бесплатная и поддерживает все необходимые решения для освоения необходимого в рамках Программы материала.

Задание 5.4.1.

1. Создайте свой аккаунт на <https://www.heroku.com/>. В правом верхнем углу нажмите на кнопку Sign up для регистрации:



Заполните открывшуюся форму. В поле “Pick your primary development language” укажите язык Java.

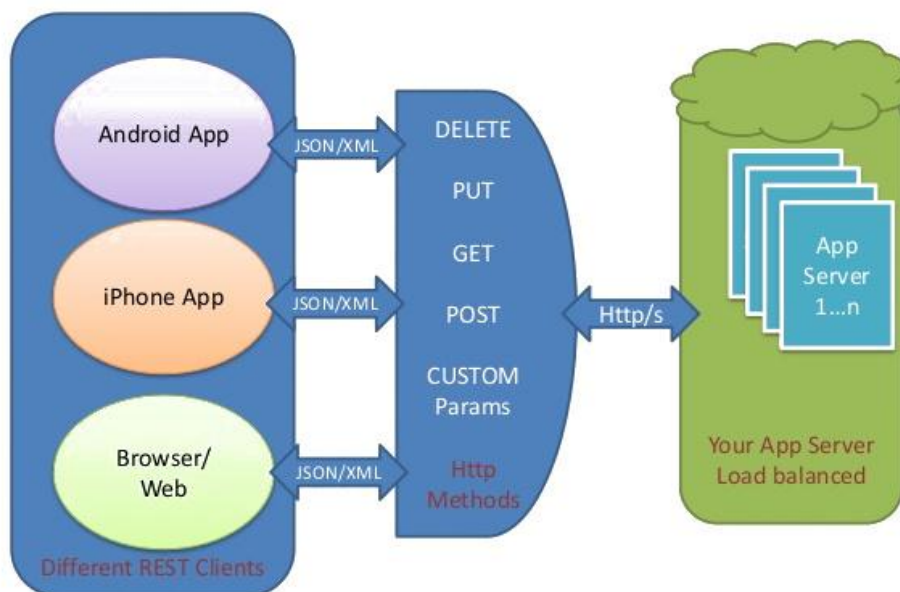
2. Изучите описание платформы. Какие решения она поддерживает?

5.4.5. REST взаимодействие

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-сервисов. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML, как это делают другие протоколы, например, SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.





Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат. Это означает, что URL по сути является первичным ключом для единицы данных. Т.е., например, третья книга на книжной полке будет иметь вид `/book/3`, а 35 страница в этой книге — `/book/3/page/35`. Отсюда и получается строго заданный формат. Причем совершенно не имеет значения, в каком формате находятся данные по адресу `/book/3/page/35` — это может быть и HTML, и отсканированная копия в виде jpeg-файла, и документ Microsoft Word.

Как происходит управление информацией сервиса — это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Для HTTP действие над данными задается с помощью методов: GET, PUT, POST, DELETE. Таким образом, действия CRUD (Create-Read-Updtae-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Вот как это будет выглядеть на примере:

- GET `/book/` — получить список всех книг
- GET `/book/3/` — получить книгу номер 3
- PUT `/book/` — добавить книгу (данные в теле запроса)
- POST `/book/3` — изменить книгу (данные в теле запроса)
- DELETE `/book/3` — удалить книгу

Принципы REST

Клиент-серверная архитектура

Единый интерфейс между клиентом и сервером. Такое разделение подразумевает отсутствие связи между клиентами и хранилищем данных. Это хранилище остаётся внутренним устройством сервера. Серверы и клиенты могут быть мгновенно заменены независимо друг от друга, так как интерфейс между ними не меняется.

Отсутствие состояния

Серверы не связаны с интерфейсами клиентов и их состояниями. На стороне сервера не сохраняется пользовательский контекст между двумя разными запросами. Каждый запрос содержит всю информацию, необходимую обработчику, а состояние сессии хранится на клиенте.

Кэширование

Как и во Всемирной паутине, каждый из клиентов, а также промежуточные узлы между сервером и клиентами могут кэшировать ответы сервера. В каждом запросе клиента должно явно содержаться указание о возможности кэширования ответа и получения ответа из существующего кэша. В свою очередь, ответы могут явно или неявно определяться как кэшируемые или некашируемые для предотвращения повторного использования клиентами в последующих запросах сохранённой информации. Правильное использование кэширования в REST-архитектуре

Единообразие интерфейса

Ограничения на унифицированный интерфейс являются фундаментальными в дизайне REST-сервисов. Каждый из сервисов функционирует и развивается независимо. Ограничения для унификации интерфейса:

1. Идентификация ресурсов.

Индивидуальные ресурсы идентифицированы в запросах, например, с использованием URI в интернет-системах. Сами по себе отделены от представлений, которые возвращаются клиентам. Например, сервер может отсылать данные из базы данных в виде HTML, XML или JSON, ни один из которых не является типом хранения внутри хранилища сервера.

2. Манипуляция ресурсами через представление.

В момент, когда клиенты хранят представление ресурса, включая метаданные, они имеют достаточно данных для модификации или удаления ресурса.

3. «Самодостаточные» сообщения.

Каждое сообщение достаточно информативно для того, чтобы описать каким образом его обрабатывать. К примеру, какой парсер необходимо применить для извлечения данных из сообщения согласно MIME (Internet медиа)-типу.

4. Гипермедиа, как средство изменения состояния сервера.

Клиенты могут изменить состояние системы только через действия, которые динамически идентифицируются на сервере посредством гипермедиа (к примеру, гиперссылки в гипертексте, формы связи, флажки, радиокнопки и прочее). До того момента, пока сервер в явном виде не сообщит обратное, клиенты могут полагаться на то, что любое из предоставленных действий доступно для выполнения на сервере.

Слои

Клиент может взаимодействовать не напрямую с сервером, а через промежуточные узлы (слои). При этом клиент может не знать об их существовании, за исключением случаев передачи конфиденциальной информации.

Retrofit — библиотека для работы с REST API

Раньше в Android единственным доступным средством для выполнения сетевых запросов был клиент Apache. Он оптимален для старых версий Android (Eclair и Froyo). Позже плодом стараний разработчиков Google стал класс `URLConnection`. Он исправлял некоторые недостатки клиента Apache, но как и у Apache у него не было возможности выполнять асинхронные запросы.

В 2013 году появились библиотеки Volley и Retrofit. Volley — библиотека более общего плана, предназначенная для работы с сетью, в то время как Retrofit специально создана для работы с REST API.

В общем случае, при выполнении запроса к REST-серверу, требуется выполнить ряд операций:

- сформировать URL;
- задать HTTP-заголовки;
- выбрать тип HTTP-запроса;
- сформировать тело HTTP-запроса, т.е. преобразовать Java объект в JSON;
- выполнить запрос, воспользовавшись HTTP-клиентом;
- распарсить результаты запроса, т.е. преобразовать полученный JSON обратно в Java объект.

Библиотека Retrofit позволяет описать все перечисленные операции с помощью аннотаций.

Описание API

Описание запросов к серверу происходит в интерфейсе (interface). Над каждым методом должна стоять аннотация, с помощью которой Retrofit «узнает», какого типа запрос (например, `@GET` или `@POST`). Также с помощью аннотаций можно указывать параметры запроса.

Вот так, например, выглядит описание GET-запроса:

```
import retrofit.client.Response;
import retrofit.http.GET;

public interface API {
    @GET("/v1/users")
    Response getUsers();
}
```

Не нужно указывать адрес сайта, который отправляет запрос, нужно лишь указать расположение требуемого файла на сервере. В классе `Response` содержится информация о статусе запроса и ответ от сервера.

А вот так выглядит POST-запрос:

```
import retrofit.client.Response;
import retrofit.http.POST;

public interface API {
    @POST("/v1/registration")
    Response registerUser();
}
```

Можно изменять путь к файлу динамически:

```
@GET("/{version}/users")
Response getUsers(@Path("version") String version);
```

Retrofit заменит слово «{version}» на, то которое будет передано методу. Сам аргумент должен быть аннотирован словом Path и в скобках нужно указать ключевое слово.

Параметры запроса

Тут тоже нет ничего сложного:

```
@GET("/v1/users")
Response getUsers(@Query("gender") String gender);
```

Для того, чтобы задать запросу параметры используется аннотация @Query. Слово, указанное в скобках рядом с аннотацией, будет ключом, а аннотированный аргумент значением.

Синхронные и асинхронные запросы

Есть два способа отправки запроса: синхронный и асинхронный. Для синхронной отправки запроса нужно вынести его в отдельный поток. Пример синхронной отправки запроса:

```
@GET("/users/{id}")
Response getUserInfo(@Path("id") int id);
```

В асинхронном запросе метод ничего не возвращает (void), но обязательно должен принимать на вход объект интерфейса Callback<T>. В качестве параметра интерфейса нужно передать тип возвращаемого объекта. Вот пример асинхронного запроса:

```
@GET("/users/{id}")
void getUserInfo(@Path("id") int id, Callback<Response> cb);
```

Retrofit 2.0.0

Раньше для выполнения синхронных и асинхронных запросов необходимо было писать разные методы. Теперь при попытке создать сервис, который содержит void метод, будет получена ошибка. В Retrofit 2.0.0 интерфейс Call инкапсулирует запросы и позволяет выполнять их синхронно или асинхронно.

Раньше

```
public interface AirportsService {

    @GET("/places/coords_to_places_ru.json")
    List<Airport> airports(@Query("coords") String gps);

    @GET("/places/coords_to_places_ru.json")
    void airports(@Query("coords") String gps, Callback<List<Airport>>
callback);

}
```

Сейчас

```
AirportsService service = ApiFactory.getAirportsService();
Call<List<Airport>> call = service.airports("55.749792,37.6324949");

//sync request
call.execute();

//async request
Callback<List<Airport>> callback = new RetrofitCallback<List<Airport>>() {
    @Override
    public void onResponse(Response<List<Airport>> response) {
        super.onResponse(response);
    }
};
call.enqueue(callback);
```

Упражнение 5.4.1

Продemonстрируем возможности REST взаимодействия на примере одного из API Яндекс (<https://tech.yandex.ru/#catalog>) - Предиктора.

Яндекс.Предиктор подсказывает наиболее вероятное продолжение слов или фраз, набираемых пользователем. Это упрощает ввод текста, особенно на мобильных устройствах. При этом Предиктор учитывает возможные опечатки.

Описание API

Для доступа к API Предиктора по HTTP предлагаются интерфейсы XML, JSON. Все интерфейсы обеспечивают одинаковую функциональность и используют одни и те же входные параметры.

XML-интерфейс возвращает ответ в виде XML-документа, JSON-интерфейс вместо XML-элементов возвращает JSON-объекты с теми же именами и семантикой.

Доступ ко всем методам API осуществляется по ключу. Получить ключ можно по ссылке <https://tech.yandex.ru/keys/get/?service=pdct>.

Методы API

Метод getLangs Возвращает список языков, поддерживаемых сервисом.

Описание:

```
string[] getLangs(string key);
```

Входные параметры:

Параметр	Тип	Описание
key	string	API-ключ

Пример запроса:

XML-интерфейс:

```
https://predictor.yandex.net/api/v1/predict/getLangs?key=API-ключ
```

JSON-интерфейс:

```
https://predictor.yandex.net/api/v1/predict.json/getLangs?key=API-ключ
```

Возвращает:

В XML-интерфейсе возвращает ответ в виде XML-документа с корневым элементом `ArrayOfString`.

Например:

```
<ArrayOfString>
  <string>ru</string>
  <string>en</string>
  <string>pl</string>
  <string>uk</string>
  <string>de</string>
  <string>fr</string>
  <string>es</string>
  <string>it</string>
  <string>tr</string>
</ArrayOfString>
```

В JSON-интерфейсе вместо XML-элементов возвращаются JSON-объекты с теми же именами и семантикой:

```
["ru","en","pl","uk","de","fr","es","it","tr"]
```

Коды ошибок

Код	Значение	Описание
ERR_KEY_INVALID	401	Ключ API невалиден.
ERR_KEY_BLOCKED	402	Ключ API заблокирован.

Метод `complete`

Возвращает наиболее вероятное продолжение текста, а также признак конца слова.

Описание:

```
CompleteResponse complete(string key, string q, string lang, int limit);
```

Входные параметры могут передаваться либо с помощью HTTP GET-запроса (см. пример), либо с помощью HTTP POST-запроса, где параметры передаются в `body` HTTP-запроса.

Пример запроса:

XML-интерфейс:

```
https://predictor.yandex.net/api/v1/predict/complete?key=API-
ключ&q=hello+wo&lang=en
```

JSON-интерфейс:

```
https://predictor.yandex.net/api/v1/predict.json/complete?key=API-
ключ&q=hello+wo&lang=en
```

Список входных параметров.

Параметр	Тип	Описание
Обязательные		
key	string	API-ключ
lang	string	Язык текста (например, "en"). Список языков можно получить с помощью метода getLangs.
q	string	Текст, на который указывает курсор пользователя. При подборе подсказки учитывается слово, на которое указывает курсор и 2 слова слева от него, поэтому не требуется передавать текст длиннее трех-четырех слов.
Необязательные		
limit	int	Максимальное количество возвращаемых строк (по умолчанию 1).

Возвращает

В XML-интерфейсе возвращает структуру CompleteResponse, содержащую текст подсказки. Например:

```
<?xml version="1.0" encoding="utf-8"?>
<CompleteResponse endOfWord="false" pos="-2">
  <text>
    <string>world</string>
    ...
  </text>
</CompleteResponse>
```

Элементы XML-схемы ответа:

Элемент	Описание
CompleteResponse	Корневой элемент, содержит элемент text. Атрибуты: endOfWord - признак конца слова (true/false).

	<p>Метод может одновременно вернуть "продолжение" запроса и признак endOfWord = true (например, для текста "здравствуй" возвращается продолжение "здравствуйте", а также признак конца слова).</p> <p>pos - позиция в слове, для которого возвращается продолжение. Позиция отсчитывается от последнего символа в запросе, переданном в элементе q.</p> <p>Обычно, pos принимает отрицательные значения. Например, для текста q="кот в сапо" будет возвращено pos=-4. Если запрос содержит завершённую фразу, то может быть возвращена подсказка для следующего слова. При этом pos будет принимать значение 0 (например, при q="кот в ") или 1 (например, q="кот в").</p>
text	<p>Содержит элементы string с наиболее вероятными вариантами продолжения заданного текста.</p> <p>Если метод не может "продолжить" текст, то элемент не возвращается.</p>



В JSON-интерфейсе вместо XML-элементов возвращаются JSON -объекты с теми же именами и семантикой:

```
{"endOfWord":false,"pos":-2,"text":["world"]}
```

Коды ошибок

Код	Значение	Описание
ERR_OK	200	Операция выполнена успешно.
ERR_KEY_INVALID	401	Ключ API невалиден.
ERR_KEY_BLOCKED	402	Ключ API заблокирован.
ERR_DAILY_REQ_LIMIT_EXCEEDED	403	Превышено суточное ограничение на количество запросов (с учетом вызовов метода getLangs).
ERR_DAILY_CHAR_LIMIT_EXCEEDED	404	Превышено суточное ограничение на объем подсказанного текста (с учетом вызовов метода getLangs).
ERR_TEXT_TOO_LONG	413	Превышен максимальный размер текста (1000 символов).
ERR_LANG_NOT_SUPPORTED	501	Указанный язык не поддерживается.

Создадим клиент-серверное приложение, использующее библиотеку Retrofit2 и сервис Яндекс.Предиктор.

	<p>Для подключения библиотеки Retrofit2 необходимо скопировать jar-файлы этой библиотеки в папку lib проекта. Jar файлы можно скачать по ссылке: https://drive.google.com/open?id=0B6zyQHe5N8jnenoxanpYaGdrUUk</p>
	<p>Для подключения библиотеки Retrofit2 необходимо добавить следующие строки в файл build.gradle проекта:</p> <pre>dependencies { compile 'com.squareup.retrofit2:retrofit:2.0.0-beta4' compile 'com.squareup.retrofit2:converter-gson:2.0.0-beta4' }</pre>

Реализуем приложение в Android Studio. Создадим пакет ru.myitschool.predictor.

Файл разметки:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    tools:context="ru.myitschool.predictor.MainActivity" >  
  
    <EditText  
        android:id="@+id/editText1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_alignParentTop="true"  
        android:layout_centerHorizontal="true"  
        android:layout_marginTop="38dp"  
        android:ems="10" />  
  
    <TextView  
        android:id="@+id/textView1"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_below="@+id/editText1"  
        android:layout_centerHorizontal="true"  
        android:layout_marginTop="69dp"  
        android:text="@string/callback" />  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:text="@string/ccopy"  
        android:id="@+id/ccopy"  
        android:layout_centerVertical="true"  
        android:layout_alignParentEnd="true" />
```

```
</RelativeLayout>
```

В классе MainActivity сохраним API ключ в PREDICTOR_KEY. Этот ключ можно легко получить на странице сервиса <https://tech.yandex.ru/keys/get/?service=pdct>.

```
package ru.myitschool.predictor;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.EditText;
import android.widget.TextView;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class MainActivity extends Activity {
    private static String PREDICTOR_URI_JSON = "https://predictor.yandex.net/";
    private static String PREDICTOR_KEY =
"pdct.1.1.20160224T140053Z.cb27d8058bc81d29.10b405aa732895274b7d14c9f7a55a116a832d9
3";
    EditText editText;
    TextView textView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        editText = (EditText) findViewById(R.id.editText1);
        textView = (TextView) findViewById(R.id.textView1);
        editText.addTextChangedListener(new TextWatcher() {
            @Override
            public void afterTextChanged(Editable s) {
                // Прописываем то, что надо выполнить после изменения текста
                getReport();
            }

            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int
after) {
            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int
count) {
            }

        });
    }
}
```

```

    }

    void getReport() {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(PREDICTOR_URI_JSON)
            .addConverterFactory(GsonConverterFactory.create())
            .build();

        RestApi service = retrofit.create(RestApi.class);

        Call<Model> call = service.predict(PREDICTOR_KEY,
            editText.getText().toString(), "ru");

        call.enqueue(new Callback<Model>() {

            @Override
            public void onResponse(Call<Model> call, Response<Model> response) {
                try {
                    String textWord = response.body().text[0].toString();
                    textView.setText("Предиктор : " + textWord);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void onFailure(Call<Model> call, Throwable t) {

            }

        });
    }

    @Override
    public void onStart() {
        super.onStart();
    }

    @Override
    public void onStop() {
        super.onStop();
    }
}

```

Класс для объекта, который мы создадим, получив ответ от Яндекса

```

public class Model {
    public boolean endOfWord;
    public int pos;
    public String[] text;
}

```

Interface започа Retrofit

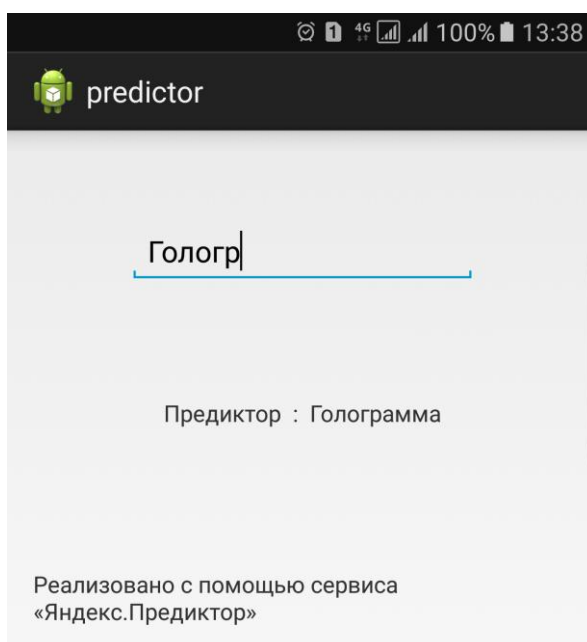
```
package ru.myitschool.predictor;

import retrofit2.Call;
import retrofit2.http.GET;
import retrofit2.http.Query;

public interface RestApi {

    @GET("api/v1/predict.json/complete")
    Call<Model> predict(
        @Query("key") String key,
        @Query("q") String q,
        @Query("lang") String lang
    );
}
```

В результате получим простое приложение, которое демонстрирует работу REST-сервиса Яндекс. Будет показано, какие слова предиктор предлагает в ответ на введенные начала слов



5.4.6. REST аутентификация и OAuth авторизация

Рассмотрим различные способы REST аутентификации.

Basic Authentication

Basic Authentication — пользователь или REST клиент указывает свой логин и пароль для получения доступа к REST сервису. Логин и пароль передаются по сети как незашифрованный текст кодированный простым Base64 и может быть легко декодирован любым пользователем. Поэтому при использовании такого метода, должен использоваться https протокол для передачи данных.

Digest authentication

Digest authentication — это почти тоже самое что первый метод, только логин и пароль передаются в зашифрованном виде, а не как обычный текст. Логин и пароль шифруются MD5

алгоритмом, и его достаточно сложно расшифровать. При этом подходе можно использовать незащищенное http соединение и не бояться, что пароль будет перехвачен злоумышленниками.

Digital Signature (public/private key pair)

Идея этого подхода заключается в использовании криптосистемы с открытым ключом. Суть состоит в том, что любой может обратиться к REST сервису и получить беспорядочный набор символов, а точнее зашифрованный ответ от сервера и только владелец приватного ключа сможет его расшифровать.

1. Когда регистрируется новый пользователь на сервере генерируется пара ключей для этого пользователя — публичный и приватный
2. Приватный отсылается пользователю и только он сможет расшифровать сообщение (ключ должен отправляться по безопасному каналу, чтобы никто не мог его перехватить)
3. При каждом REST запросе клиент передает свой логин, чтобы сервис мог зашифровать сообщение нужным публичным ключом
4. Сервис шифрует и отправляет сообщение
5. Клиент принимает его и расшифровывает своим ключом

Certificate Authentication

Можно настроить свой сервер таким образом, что, если клиент при запросе не предоставляет нужный сертификат-ответ от сервера, он получит ответ, что сертификат отсутствует или не подходит.

Token Authentication

Суть этого способа заключается в том, что пользователь, используя свои учетные данные логинится в приложении и получает токен для доступа к REST сервису. Доступ к сервису, который выдает токены, должен обязательно быть осуществлен через https соединение, доступ к REST сервису можно сделать через обычный http. Токен должен содержать логин, пароль, так же может содержать expiration time (время хранения объекта) и роли пользователя, а так же любую нужную для вашего приложения информацию. После того, как токен готов и, к примеру, все его параметры разделены двоеточием или другим удобным символом или сериализованы как json или xml объект, его необходимо зашифровать, прежде чем отдать пользователю. Только REST сервис должен знать, как расшифровывать этот токен. После того, как токен приходит на REST сервис, он его расшифровывает и получает все необходимые данные для аутентификации и, если необходимо, авторизации REST клиента.



Аутентификация и авторизация: это нужно знать разработчику!

Эти понятия часто смешивают, потому что эти процессы всегда связаны, но их необходимо различать.

Аутентификация — это подтверждение подлинности объекта. Например через пароль, ответ на секретный вопрос, отпечаток пальца и т.д..

Авторизация — это проверка прав на доступ и предоставление доступа после того как объект/пользователь/компьютер/сервис был аутентифицирован.

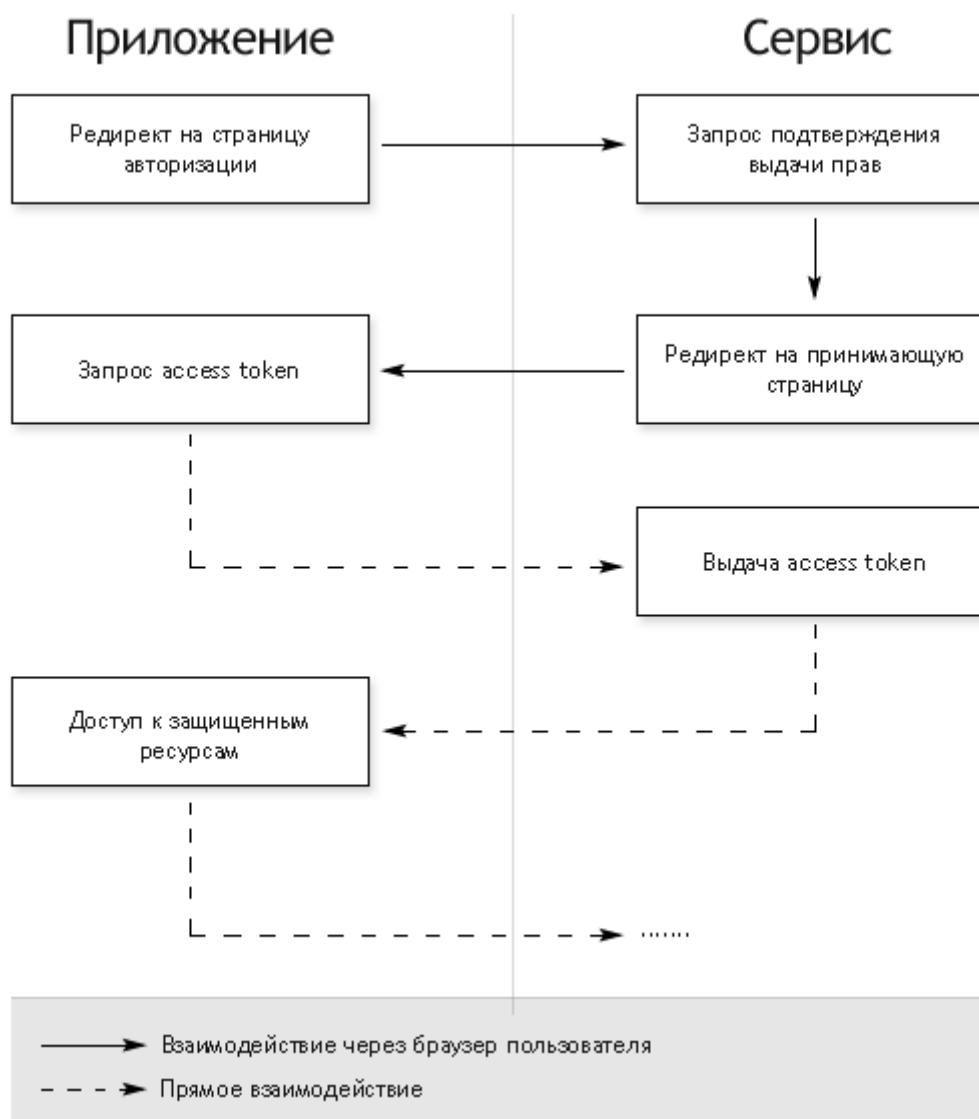
Разберем эти понятия на примере входа на наш учебный портал <http://myitschool.ru/is/>:

- Аутентификация: мы указываем свой логин и пароль, система находит

	<p>пользователя с таким логином, сверяет хеш введенного пароля с хранимым в системе. Если они совпадают, то система считает, что пользователь подлинный.</p> <ul style="list-style-type: none"> • Авторизация: после аутентификации система, исходя из роли и прав, определенных для этого пользователя (учащийся, учитель, администратор и т.п.), определяет свое поведение по отношению к нему (интерфейс, доступ к функциям и т.п.).
--	--

OAuth-авторизация

С распространением социальных сетей все большее распространение получил открытый протокол авторизации OAuth 2.0, который позволяет предоставить третьей стороне ограниченный доступ к защищенным ресурсам пользователя без необходимости передавать ей (третьей стороне) логин и пароль [5].



Основные причины популярности протокола OAuth 2.0:

- Пользователям не нравится заполнять одни и те же данные форм регистрации в каждом приложении, которым они хотят воспользоваться (интернет-магазины, сайты, игры и пр.). Придумывать каждый раз новую пару логин-пароль — забудется, вводить одно и то же на

всех сайтах — небезопасно. Поэтому, когда приложение предлагает воспользоваться для входа своим аккаунтом ВКонтакте или mail.ru, это очень удобно!

- Разработчикам приложений реализовать эту возможность очень просто. OAuth использует Http протокол. Вся стандартную для регистрации информацию легко получить у сервис-провайдера (социальная сеть, почтовый сервис).
- Помимо решения задачи простой аутентификации и авторизации приложения могут получать ограниченный доступ к данным пользователя у сервис-провайдера. В результате появились сервисы печати фотографий из аккаунтов пользователей в соцсетях, развлекательные приложения, которые выдают заключения на основе автоматического анализа профиля пользователя и многое другое.

Упражнение 5.4.2

Создадим клиент-серверное приложение, которое будет работать по следующей схеме

1. Приложение реализует OAuth-авторизацию через социальную сеть ВКонтакте¹
2. Приложение получает от ВКонтакте информацию из профиля пользователя, выводит ее на экран (для визуализации) и отправляет ее на сервер
3. Сервер десериализует полученную информацию, выделенные Имя и Фамилию пользователя отправляет в приложение
4. Приложение выводит полученные Имя и Фамилию на экран.

Серверную часть приложения реализуем средствами Java,*PHP.

В первом варианте в качестве сервера будем использовать локальный компьютер.

Во втором варианте это же приложение реализуем более подходящим для практического использования способом - разместим сервер в облаке Heroku.

Этап 1. Клиент - локальный сервер

Серверная часть

В основу серверной части можно взять разработку из Упражнения 5.3.2. Удалим классы, которые отвечали за вывод информации на веб страницу (нам это не нужно) и реализуем необходимую функциональность. Для разработки серверной части, как и в предыдущем упражнении, мы использовали Eclipse и библиотеку Spring.

```
package ru.itschool;

import java.util.Arrays;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class Application {
```

¹ В случае блокировки ресурса vk.com на площадке можно использовать собственные смартфоны в качестве Wifi-точки

```

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(Application.class, args);

    }
}

```

Класс, который получает, десериализует объект и возвращает только Имя и Фамилию:

```

package ru.itschool.controller;

import ru.itschool.model.VK_User;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
public class VK_UserController {
    @RequestMapping(path = "/VK", method = RequestMethod.POST)
    public @ResponseBody String VK(@RequestBody VK_User vkUser)
    {
        System.out.println(vkUser.first_name + " " + vkUser.last_name);
        return vkUser.first_name + " " + vkUser.last_name;
    }
}

```

Класс VK_User для хранения десериализованной информации:

```

package ru.itschool.model;

public class VK_User {

    public String uid;
    public String first_name;
    public String last_name;
    public String screen_name;
    public String sex;
    public String bdate;
    public String photo_big;

}

```

*На PHP аналогичный код будет выглядеть так:

```

<?php

$userInfo = json_decode(file_get_contents('php://input'), true);

if (isset($userInfo['uid']))
    echo $userInfo['first_name']." ".$userInfo['last_name'] ;
else
    echo "No correct data";

```

?>

Клиентская часть

При авторизации в соответствии с требованиями сервиса ВКонтакте необходимы параметры:

Параметр	Описание
Client_id Обязательно	Идентификатор Вашего приложения.
Redirect_uri Обязательно	Адрес, на который будет переадресован пользователь после прохождения авторизации (домен указанного адреса должен соответствовать основному домену в настройках приложения и перечисленным значениям в списке доверенных redirect uri - адреса сравниваются вплоть до path-части).
Display Обязательно	Указывает тип отображения страницы авторизации. Поддерживаются следующие варианты: <ul style="list-style-type: none"> page — форма авторизации в отдельном окне; popup — всплывающее окно; mobile — авторизация для мобильных устройств (без использования Javascript) Если пользователь авторизуется с мобильного устройства, будет использован тип mobile .
scope	Битовая маска настроек доступа приложения , которые необходимо проверить при авторизации пользователя и запросить, в случае отсутствия необходимых.
response_type	Тип ответа, который Вы хотите получить. Укажите code , чтобы осуществлять запросы со стороннего сервера.
v	Версия API, которую Вы используете. Актуальная версия: 5.50.
state	Произвольная строка, которая будет возвращена вместе с результатом авторизации.

Получим необходимые регистрационные данные приложения на сайте <http://vk.com/editapp?act=create>. Заполняем поля формы:

Создание приложения

Название:

Тип:

☐ Standalone-приложение
☒ Веб-сайт
☐ IFrame/Flash приложение

Адрес сайта:

Базовый домен:

Подключить сайт

Подтверждаем действия через отправку уведомления на телефон. В разделе “Настройки” видим необходимые данные: **ID приложения** и **Защищенный ключ**. Эти данные мы и вставим в код клиента.



Если будет стоять задача создать другое приложение с авторизацией ВКонтакте, то для него необходимо получить собственные регистрационные данные, аналогично тому, как мы показали для данного упражнения.

Публикация ID и ключа приложения в открытом доступе запрещена.

Для работы с сервисом ВКонтакте необходимо также знать ряд ссылок:

- страницы для авторизации OAUTH_URL = "http://oauth.vk.com/authorize"
- для получения токена TOKEN_URL = "https://oauth.vk.com/access_token"
- для получения данных из профиля пользователя VK_API_URL = "https://api.vk.com/method/users.get"

Главный класс MainActivity открывает WebView с формой авторизации ВКонтакте. После авторизации на главной активности выводятся результаты работы: отправленная на сервер JSON-строка и ответ сервера .

Обратите внимание! Так как мы запускаем сервер на своей машине, в переменной OUR_SERVER необходимо указать свой IP адрес;

```
package ru.myitschool.oauth_vk;

import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.AsyncTask;
```

```
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.a0.R;

import org.json.JSONObject;

public class MainActivity extends Activity {

    private static String CLIENT_ID = "5400655";
    private static String CLIENT_SECRET = "wWtRxNOBD6yfzkclsaSE";

    private static String TOKEN_URL = "https://oauth.vk.com/access_token";
    private static String OAUTH_URL = "http://oauth.vk.com/authorize";
    private static String RESPONSE_TYPE = "code";
    private static String VK_API_URL = "https://api.vk.com/method/users.get";

    private static String REDIRECT_URI = "http://localhost";
    private static String OUR_SERVER = "http://192.168.1.35:8080/VK/";

    WebView web;
    Button auth;
    SharedPreferences pref;
    TextView Access;
    TextView serverResponse;
    String responsePOST;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        pref = getSharedPreferences("AppPref", MODE_PRIVATE);
        Access = (TextView) findViewById(R.id.Access);
        serverResponse = (TextView) findViewById(R.id.response);
        auth = (Button) findViewById(R.id.auth);
        auth.setOnClickListener(new View.OnClickListener() {
            Dialog auth_dialog;

            @Override
            public void onClick(View arg0) {
                // TODO Auto-generated method stub
                auth_dialog = new Dialog(MainActivity.this);
                auth_dialog.setContentView(R.layout.auth_dialog);
                web = (WebView) auth_dialog.findViewById(R.id.webv);
                web.getSettings().setJavaScriptEnabled(true);
                web.loadUrl(OAUTH_URL + "?client_id=" + CLIENT_ID
                    + "&redirect_uri=" + REDIRECT_URI + "&response_type="
                    + RESPONSE_TYPE);
                web.setWebViewClient(new WebViewClient() {

                    boolean authComplete = false;
```



```

        Intent resultIntent = new Intent();

        @Override
        public void onPageStarted(WebView view, String url,
                                   Bitmap favicon) {
            super.onPageStarted(view, url, favicon);
        }

        String authCode;

        @Override
        public void onPageFinished(WebView view, String url) {
            super.onPageFinished(view, url);

            if (url.contains("?code=") && authComplete != true) {
                Uri uri = Uri.parse(url);
                authCode = uri.getQueryParameter("code");
                Log.i("", "CODE : " + authCode);
                authComplete = true;
                resultIntent.putExtra("code", authCode);
                MainActivity.this.setResult(Activity.RESULT_OK,
                                           resultIntent);
                setResult(Activity.RESULT_CANCELED, resultIntent);

                SharedPreferences.Editor edit = pref.edit();
                edit.putString("Code", authCode);
                edit.commit();
                auth_dialog.dismiss();
                new TokenGet().execute();
                Toast.makeText(getApplicationContext(),
                               "Authorization Code is: " + authCode,
                               Toast.LENGTH_SHORT).show();
            } else if (url.contains("error=access_denied")) {
                Log.i("", "ACCESS_DENIED_HERE");
                resultIntent.putExtra("code", authCode);
                authComplete = true;
                setResult(Activity.RESULT_CANCELED, resultIntent);
                Toast.makeText(getApplicationContext(),
                               "Error Occured", Toast.LENGTH_SHORT).show();

                auth_dialog.dismiss();
            }
        }
    });
    auth_dialog.show();
    auth_dialog.setTitle("IT ШКОЛА SAMSUNG");
    auth_dialog.setCancelable(true);
});
}

private class TokenGet extends AsyncTask<String, String, JSONObject> {
    private ProgressDialog pDialog;
    String Code;

    @Override
    protected void onPreExecute() {

```

```

        super.onPreExecute();
        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setMessage("Connecting VK ...");
        progressDialog.setIndeterminate(false);
        progressDialog.setCancelable(true);
        Code = pref.getString("Code", "");
        progressDialog.show();
    }

    @Override
    protected JSONObject doInBackground(String... args) {
        GetAccessToken jParserToken = new GetAccessToken();
        JSONObject jsonToken = jParserToken.getToken(TOKEN_URL, Code,
            CLIENT_ID, CLIENT_SECRET, REDIRECT_URI);

        GetUserInfo jParserUser = new GetUserInfo();
        JSONObject jsonUser = jParserUser
            .getUserinfo(VK_API_URL, jsonToken);

        SetToServer jsetToServer = new SetToServer();
        responsePOST = jsetToServer.setPOST(OUR_SERVER, jsonUser);

        return jsonUser;
    }

    @Override
    protected void onPostExecute(JSONObject json) {
        progressDialog.dismiss();
        if (json != null) {

            String jsonResp = json.toString();

            auth.setVisibility(View.GONE);
            Access.setText(jsonResp);
            serverResponse.setText("Ответ сервера: " + responsePOST);

        } else {
            Toast.makeText(getApplicationContext(), "Network Error",
                Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
        }
    }
}

```

Разметка главной активности activity_main.xml:

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <ImageView

```

```

        android:layout_width="match_parent"
        android:layout_height="94dp"
        android:id="@+id/imageView"
        android:src="@drawable/vk_logo" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:id="@+id/auth"
            android:text="@string/enter" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/Access"
            android:autoLink="all"
            android:textIsSelectable="true"
            android:layout_gravity="center"
            android:textSize="10sp"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/response"
            android:layout_gravity="center_horizontal" />

    </LinearLayout>

```

Активности с WebView в файле auth_dialog.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <WebView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/webv"/>

</LinearLayout>

```

strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">VK OAuth</string>
    <string name="action_settings">Settings</string>
    <string name="enter">Войти</string>
</resources>

```

Реализуем отправку регистрационных данных приложения, чтобы получить от ВКонтакте: token и id пользователя.

```
package ru.myitschool.oauth_vk;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;
//Класс для отправки сервису ВКонтакте данных для авторизации
// и получения token, срока его жизни и id пользователя ВКонтакте

public class GetAccessToken {
    static InputStream isT = null;
    static JSONObject jsonObjT = null;
    static String jsonT = "";

    public GetAccessToken() {
    }

    List<NameValuePair> params = new ArrayList<NameValuePair>();
    Map<String, String> mapn;
    DefaultHttpClient httpClient;
    HttpPost httpPost;

    public JSONObject gettoken(String address, String token, String client_id,
String client_secret, String redirect_uri) {
        // Making HTTP request
        try {
            // DefaultHttpClient
            httpClient = new DefaultHttpClient();
            httpPost = new HttpPost(address);

            params.add(new BasicNameValuePair("code", token));
            params.add(new BasicNameValuePair("client_id", client_id));
            params.add(new BasicNameValuePair("client_secret", client_secret));
            params.add(new BasicNameValuePair("redirect_uri", redirect_uri));

            httpPost.setHeader("Content-Type", "application/x-www-form-urlencoded");
            httpPost.setEntity(new UrlEncodedFormEntity(params));
            HttpResponse httpResponse = httpClient.execute(httpPost);
```

```

        HttpEntity httpEntity = httpResponse.getEntity();
        isT = httpEntity.getContent();

    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    try {
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            isT, "utf-8"), 8);
        StringBuilder sb = new StringBuilder();
        String line = null;
        while ((line = reader.readLine()) != null) {
            sb.append(line + "\n");
        }
        isT.close();

        jsonT = sb.toString();
        Log.e("JSONStr", jsonT);
    } catch (Exception e) {
        e.getMessage();
        Log.e("Buffer Error", "Error converting result " + e.toString());
    }
    // Parse the String to a JSON Object
    try {
        jsonObjT = new JSONObject(jsonT);
    } catch (JSONException e) {
        Log.e("JSON Parser", "Error parsing data " + e.toString());
    }
    // Return JSON String
    return jsonObjT;
}
}

```

Класс GetUserInfo, который необходим для получения данных о пользователе ВКонтакте из его открытого профиля и преобразования этих данных в JSON объект.

```

package ru.myitschool.oauth_vk;

import android.util.Log;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.BufferedReader;

```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.List;

// Класс для запроса и получения данных из профиля пользователя ВКонтакте
public class GetUserInfo {

    static InputStream is = null;
    static JSONObject jsonObj = null;
    static String json = "";

    public GetUserInfo() {

        List<NameValuePair> params = new ArrayList<NameValuePair>();
        DefaultHttpClient httpClient;
        HttpPost httpPost;

        public JSONObject getuserinfo(String address, JSONObject jsonToken) {
            // Making HTTP request
            try {
                // DefaultHttpClient
                httpClient = new DefaultHttpClient();
                httpPost = new HttpPost(address);

                String tok = jsonToken.getString("access_token");
                String user_id = jsonToken.getString("user_id");
                //Набор запрашиваемых полей из профиля пользователя ВКонтакте
                String fields =
                    "uid,first_name,last_name,screen_name,sex,bdate,photo_big";

                params.add(new BasicNameValuePair("uids", user_id));
                params.add(new BasicNameValuePair("fields", fields));
                params.add(new BasicNameValuePair("access_token", tok));

                httpPost.setHeader("Content-Type", "application/x-www-form-urlencoded");
                httpPost.setEntity(new UrlEncodedFormEntity(params));
                //Отправка Post запроса сервису ВКонтакте
                HttpResponse httpResponse = httpClient.execute(httpPost);
                //Получение ответа на запрос
                HttpEntity httpEntity = httpResponse.getEntity();
                is = httpEntity.getContent(); //Ответ сервера в виде объекта InputStream

            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
    //Преобразование ответа сервера InputStream в строку
}
```



```

    try {
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            is, "utf-8"), 8);
        StringBuilder sb = new StringBuilder();
        String line = null;
        while ((line = reader.readLine()) != null) {
            sb.append(line + "\n");
        }
        is.close();
        json = sb.toString(); //Информация о пользователе в JSON строке
    } catch (Exception e) {
        e.getMessage();
        Log.e("Buffer Error", "Error converting result " + e.toString());
    }
    // Строку преобразуем в JSON Object
    try {

        jsonObj = new JSONObject(json);

    } catch (JSONException e) {
        Log.e("JSON Parser", "Error parsing data " + e.toString());
    }
    // Return JSON объект
    return jsonObj;
}
}

```

И, наконец, класс SetToServer для отправки JSON объекта с информацией от ВКонтакте на сервер и получения от него ответа. Как мы помним, наш сервер, получив JSON сообщение, возвращает заданные нами поля: Имя и Фамилия.

```

package ru.myitschool.oauth_vk;

import com.google.gson.Gson;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.IOException;

import okhttp3.MediaType;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.RequestBody;
import okhttp3.Response;
import ru.myitschool.oauth_vk.model.VkUser;

// Класс для передачи JSON Объекта с данными пользователя серверу
public class SetToServer {
    OkHttpClient client;
    public static final MediaType JSON = MediaType.parse("application/json;
charset=utf-8");

    String post(String url, String json) throws IOException {
        RequestBody body = RequestBody.create(JSON, json);
        Request request = new Request.Builder()

```

```

        .url(url)
        .post(body)
        .build();
    Response response = client.newCall(request).execute();
    return response.body().string();
}

public String setPOST(String address, JSONObject jsonPOST) {
    Gson gson = new Gson();
    VkUser vkUser;
    client = new OkHttpClient();
    String reponseString = "";
    JSONObject json_row = null;
    JSONArray data = null;

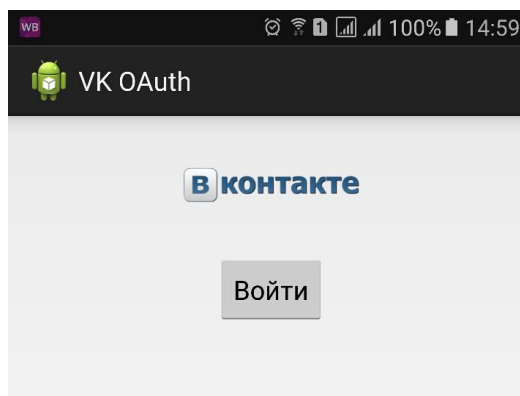
    try {
        data = jsonPOST.getJSONArray("response");
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    try {
        json_row = data.getJSONObject(0);
        vkUser = gson.fromJson(json_row.toString(), VkUser.class);
        reponseString = post(address, gson.toJson(vkUser));
    } catch (JSONException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return reponseString;
}
}

```

Стартовое окно приложения:



После авторизации на активность выводится GSON строка с данными из профиля пользователя и затем ответ сервера.

Полностью разработанное приложение в Android Studio выложено в архиве в учебном курсе.

Этап 2. Сервер на облачном сервисе Heroku

При использовании таких платформ, как Heroku, технология разработки серверной части сводится, как правило к двум шагам:

1. Разработка и отладка программы на локальном компьютере
2. Загрузка и отладка кода на Heroku

Первый шаг мы уже выполнили, остается поместить сервер в облако. В Heroku предлагают использовать для этого Heroku Git, GitHub, Dropbox.



Более подробную информацию о системе контроля версий Git и наиболее распространенном на ее основе сервисе GitHub можно узнать, просмотрев запись вебинара IT ШКОЛЫ SAMSUNG
<https://my.webinar.ru/record/621951/>

Остановимся на первом способе - Heroku Git.

1. Зайдем под ранее созданной учетной записью на сайт <https://www.heroku.com/> (Задание 5.4.1) и создадим новое приложение:

Filter personal apps and pipelines

You currently have no apps

To get started, click the button below and create a new app.

Create New App

Мы назовем наш проект "server-54". Имя проекта должно быть уникальным в рамках домена herokuapp.com, поэтому придумайте и укажите свое наименование.

App Name (optional)

Leave blank and we'll choose one for you.

server-54

server-54 is available

Runtime Selection

Your app can run in your choice of region in the Common Runtime.

Europe

Create App

В качестве способа развертывания сервера (Deployment method) выберем Heroku Git:

Deployment method

Heroku Git
Use Heroku ToolbeltGitHub
Connect to GitHubDropbox
Connect to Dropbox

Deploy using Heroku Git

Use git in the command line or a GUI tool to deploy this app.

Install the Heroku Toolbelt

Download and install the [Heroku Toolbelt](#) or learn more about the [Heroku Command Line Interface](#).

If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.

```
$ heroku login
```

Create a new Git repository

Initialize a git repository in a new or existing directory

```
$ cd my-project/  
$ git init  
$ heroku git:remote -a server-54
```

Deploy your application

Commit your code to the repository and deploy it to Heroku using Git.

```
$ git add .  
$ git commit -am "make it better"  
$ git push heroku master
```

Existing Git repository

For existing repositories, simply add the `heroku` remote

```
$ heroku git:remote -a server-54
```

2. В папке проекта нужно создать файл с именем “Procfile” (без расширения) и сохранить в нем текст:

```
web: java -Dserver.port=$PORT -jar target/demo-0.0.1-SNAPSHOT.jar
```

Данный файл будет указывать компилятору Heroки что необходимо создать web проект.

3. Установить Heroku Toolbelt, если его нет.

Скачать его можно со страницы <https://toolbelt.heroku.com/> (~48 Mb).

Для установки на Linux в терминале необходимо выполнить:

```
wget -O- https://toolbelt.heroku.com/install-ubuntu.sh | sh
```

4. Разместить проект

Запустить командную строку (терминал) и выполнить все описанные в подсказке Heroки команды.
`heroku login`

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

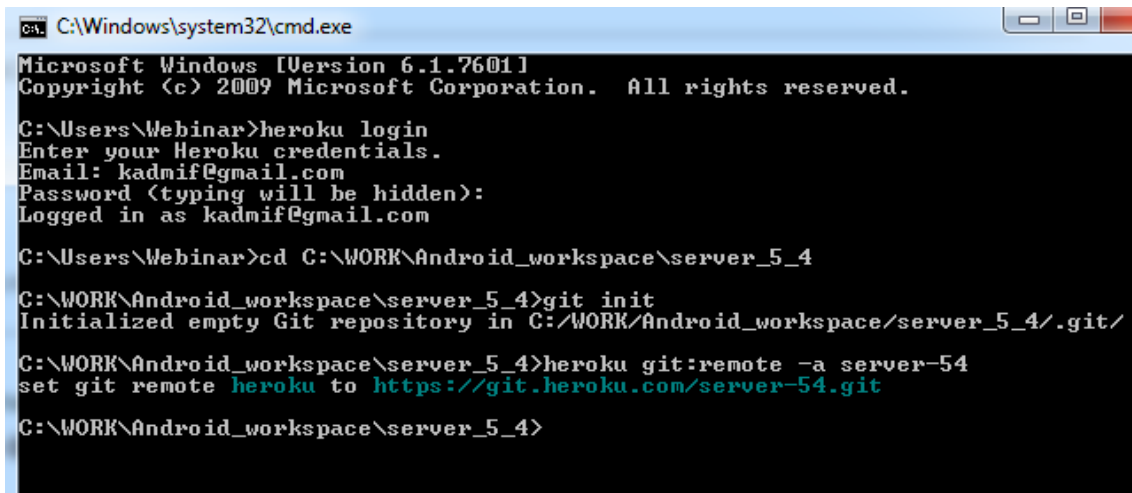
C:\Users\Webinar>heroku login
Enter your Heroku credentials.
Email: kadmif@gmail.com
Password (typing will be hidden):
Logged in as kadmif@gmail.com

C:\Users\Webinar>
```

В первый раз выполнение команды займет некоторое время, потому что будут установлены необходимые зависимости. В результате появится приглашение на ввод почты (логина) и пароля от heroku.com

Далее переходим в папку с проектом, инициализируем его для git и указываем в какой каталог на heroku.com заливать файлы:

- `cd <полный путь до каталога проекта>`
- `git init`
- `heroku git:remote -a <имя проекта на HEROKU>`



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Webinar>heroku login
Enter your Heroku credentials.
Email: kadmif@gmail.com
Password (typing will be hidden):
Logged in as kadmif@gmail.com

C:\Users\Webinar>cd C:\WORK\Android_workspace\server_5_4

C:\WORK\Android_workspace\server_5_4>git init
Initialized empty Git repository in C:/WORK/Android_workspace/server_5_4/.git/

C:\WORK\Android_workspace\server_5_4>heroku git:remote -a server-54
set git remote heroku to https://git.heroku.com/server-54.git

C:\WORK\Android_workspace\server_5_4>
```

Далее добавляем все файлы проекта для закидывания на сервер и добавляем комментарий к commit:

- `git add .`
- `git commit -am "make it better"`

В папке вашего проекта появится скрытая папка .git. И последняя команда закачает файлы на сервер, где он сам их соберет в проект:

- `git push heroku master`

5. Указать адрес сервера в клиентском приложении

Войдите в свой личный кабинет на HEROKU, выберите панель Dashboard и увидите свой проект.

Выберите свой проект и в меню Settings в разделе Domains увидите адрес сервера, который необходимо указать в переменной OUR_SERVER класса MainActivity.

В нашем случае измененная строка будет выглядеть так:

```
private static String OUR_SERVER = "https://server-54.herokuapp.com/VK/";
```

Проект завершен!

Источники

- [1] Облачные вычисления [Электронный ресурс] : Материал из Википедии — свободной энциклопедии : Версия 76886062, сохранённая в 05:40 UTC 4 марта 2016 / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2016. — Режим доступа: <http://ru.wikipedia.org/?oldid=76886062>
- [2] Дэн Орландо. Модели сервисов облачных вычислений: Часть 2. Платформа как сервис [Электронный ресурс] : официальный сайт IBM, 2016. — Режим доступа: <https://www.ibm.com/developerworks/ru/library/cl-cloudservices2paas/>
- [3] Архитектура REST [Электронный ресурс] : Материалы Хабрахабр, 2008.- Режим доступа: <https://habrahabr.ru/post/38730/>
- [4] 6 способов: как добавить security для Rest сервиса в Java [Электронный ресурс] : Материалы Хабрахабр, 2014.- Режим доступа: <https://habrahabr.ru/post/245415/>
- [5] OAuth 2.0 простым и понятным языком [Электронный ресурс] : Материалы Хабрахабр/ Блог компании Mail.Ru Group, 2014.- Режим доступа: <https://habrahabr.ru/company/mailru/blog/115163/>

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю ИТ ШКОЛЫ SAMSUNG Покровской Валерии Павловне