

5.1. IP-сети

Сайт: IT Академия SAMSUNG

Курс: MDev @ IT Академия Samsung

Книга: 5.1. IP-сети

Напечатано.: Егор Беляев

Дата: Суббота, 18 Апрель 2020, 19:33

Оглавление

- 5.1.1. Об интернете и протоколах TCP/IP
- 5.1.2. Адресация в IP-сетях
- 5.1.3. Версия интернет-протокола IPv4
- 5.1.4. Автоматизация процесса назначения IP-адресов
- 5.1.5. Доменные имена (DNS), URL-ссылки
- 5.1.6. Сервисы работы с IP-адресами
- 5.1.7. Популярные сетевые команды

5.1.1. Об интернете и протоколах TCP/IP

Прежде чем перейти к изучению тем по клиент-серверной разработке приложений, остановимся кратко на основных понятиях, связанных с интернетом и протоколами, которые обеспечивают его работу.

В июле 1961 года американский ученый Л. Клейнрок опубликовал первую статью по теории пакетной коммутации. Он предложил передавать данные через сети. Для этого он предложил разделить их на небольшие пакеты в несколько десятков байт и потом в адресате из них собрать исходное сообщение. Уже в 1969 году коллектив ученых под его руководством в Калифорнийском университете и ученые Стэнфордского исследовательского института впервые продемонстрировали передачу данных с использованием набора сетевых протоколов TCP (Transmission Control Protocol). Пакет данных прошел по маршруту Сан-Франциско — Лондон — Университет Южной Калифорнии, при этом не потеряв ни одного бита. Этот день, 29 октября 1969 года, многие историки считают днем рождения интернета.

Интернет-протокол — это набор правил, по которым компьютеры взаимодействуют между собой. Без протоколов не было бы интернета, потому что устройства в сети не понимали бы друг друга.

В 1978 году TCP был разделен на две отдельные группы:

- за разбивку передаваемого сообщения на пакеты данных и их сборку в пункте получения стал отвечать TCP;
- за передачу пакетов данных с контролем получения — IP-протокол (Internet Protocol).

По отношению к протоколам TCP/IP употребляют понятие «стек протоколов». Так происходит потому, что сейчас это уже большое множество протоколов, которые «слоями» покрывают друг друга: верхние работают на высоком логическом уровне, не вдаваясь в подробности, которые задают протоколы более низкого уровня. Сейчас в стеке протоколов TCP/IP по одной из классификаций выделяют четыре уровня.

1. **Application layer.** Самый верхний прикладной уровень. Здесь работают протокол HTTP для WWW, FTP (передача файлов), SMTP (электронная почта), DNS (преобразование символьных имен в IP-адреса) и многие другие.
2. **Transport layer.** Протоколы транспортного уровня TCP, UDP решают задачу передачи пакетов данных и определяют, для какого конкретно приложения они предназначены.
3. **Internet layer.** IP-протоколы сетевого уровня занимаются определением кратчайшего пути передачи пакетов данных, переводом логических адресов и имен в физические и др. На данном уровне работает сетевое устройство — маршрутизатор.
4. **Link layer.** Самый приближенный к физическим устройствам канальный уровень протоколов. Он предназначен для решения задачи передачи данных узлам, находящимся в том же сегменте локальной сети. Одним из примеров такого протокола является Ethernet.

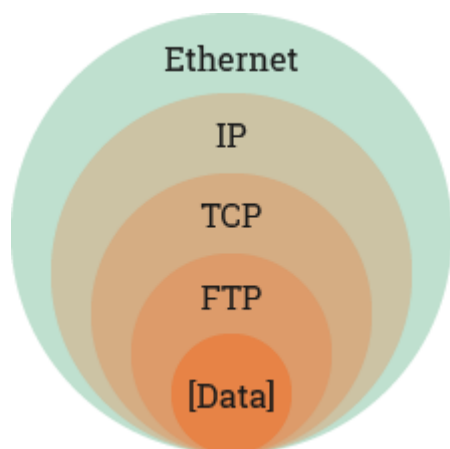


Рис. 5.1.

Теперь рассмотрим, как работает передача данных «слоями». Для передачи используется принцип инкапсуляции: сформированный пакет данных (см. Data на рис. 5.1) заворачивается (инкапсулируется) в заголовок первым протоколом (например, FTP), затем все это (включая заголовок FTP) инкапсулируется вновь следующим протоколом (скажем, TCP), затем следующим (например, IP), и, наконец, финальным, физическим протоколом (Ethernet). Каждый уровень скрывает внутренность, но добавляет ту информацию, которая необходима для выполнения текущих задач.

Когда другой компьютер получает этот пакет, оборудование (сетевая карта) исключает Ethernet-заголовок (разворачивает пакет), ядро ОС исключает заголовки IP и TCP, программа получатель исключает заголовок FTP, и, наконец, мы получаем исходные данные.

5.1.2. Адресация в IP-сетях

Для работы с протоколами TCP/IP используют три типа адресов:

- MAC-адрес (физический адрес);
- IP-адрес (сетевой адрес);
- DNS-имя (символьное доменное имя).

MAC-адрес (от Media Access Control) — это уникальный идентификатор, присваиваемый каждой единице активного оборудования, или некоторым их интерфейсам в компьютерных сетях Ethernet.

Свой MAC-адрес есть как у адаптера сети Wi-Fi, так и у адаптера проводной сети Ethernet. MAC-адрес — это шестибайтный номер. Обычно он записывается в шестнадцатиричной системе счисления, например, следующим образом: E1:39:F5:7A:E2:22. MAC-адреса устройств являются уникальными. Это обусловлено тем, что каждый производитель оборудования получает в пользование диапазон из шестнадцати миллионов адресов в комитете IEEE Registration Authority. Три старших байта идентифицируют производителя, три младших назначаются самим производителем. MAC-адреса формируют основу уровней каналов. Затем ее используют протоколы сетевого уровня.

Вы можете узнать MAC-адрес сети на ваших устройствах двумя способами:

1. Меню -> Настройки -> Сведения об устройстве (Об устройстве) -> Состояние -> MAC-адрес Wi-Fi.
2. Меню -> Настройки -> Wi-Fi -> Меню -> Дополнительно -> MAC-адрес.

IP-адрес (от Internet Protocol Address) — представляет собой основной тип адресов, с помощью которых происходит обмен пакетами на сетевом уровне. Такие адреса состоят из 4 байт и записываются четырьмя десятичными числами от 0 до 255, разделенными точками, например, 195.223.68.11. IP-адрес назначается администратором во время конфигурации компьютеров и маршрутизаторов.

DNS-имя (Domain Name System) — символьный имя-идентификатор, такой как, например, *myitschool.ru*. Доменное имя представляет собой буквенные адреса. Они гораздо удобнее для восприятия и использования, чем последовательность цифр IP-адреса.

5.1.3. Версия интернет-протокола IPv4

Сегодня IP-адрес может быть в двух форматах: IPv4, который имеет длину 4 байта, например, 192.168.0.1, и протокола IPv6, состоящего из 16 байт, например, 1021:0:6ed2:7a1b:2182:20:4db5:a1d4.

Протокол IPv4 появился еще в 1981 году. В настоящий момент это самая часто используемая версия. Однако в связи с постоянным ростом числа устройств в сети интернет, количества адресов, состоящих из 4 байт, уже не хватает. Для этого была запущена версия протокола IPv6. Она способна поддерживать значительно большее количество уникальных адресов. В настоящее время наблюдается рост использования протокола IPv6. Если на конец 2013 года доля IPv6 в мировом сетевом трафике составляла около 3%, то на сегодняшний день она превышает 15% и продолжает расти.

Вернемся к все же более актуальному на сегодняшний день протоколу IPv4. Проанализируем, например, адрес 192.168.104.115. В этих четырех числах закодированы номер сети и номер узла (компьютера) в сети. Для того чтобы идентифицировать эти части из IP-адреса, используют маски подсети.

Маска подсети — битовая маска, определяющая, какая часть IP-адреса узла сети относится к адресу сети, а какая — к адресу самого узла в этой сети (при этом, в отличие от IP-адреса, маска подсети не является частью IP-пакета).

Так же, как и IP-адрес, маска состоит из четырех чисел в диапазоне от 0 до 255 включительно. Однако она устроена особым образом по принципу: «n единиц, потом — нули» в двоичном коде. Для получения адреса подсети необходимо выполнить поразрядную конъюнкцию маски сети и IP-адреса устройства. В разрядах маски, где стоят 1, значение соответствующих разрядов в IP-адресе не изменится, а там, где 0, — обнулится. Проиллюстрируем это на примерах.

Пример 5.1

Пусть имеем IP-адрес 192.168.105.123 и маску подсети 255.255.254.0, которая в двоичном виде имеет вид:

`11111111.11111111.11111110.00000000`

Это значит, что первые 23 бита адреса — это адрес сети (192.168.104.0), а оставшиеся 9 битов — номер узла (компьютера) в этой сети (251).

Можно использовать другую запись, которая значит то же самое:

192.168.105.123/23 — «/23» говорит о том, что в маске 23 единицы.

В такой сети может быть 510 узлов, а не 512, как можно было бы ожидать. Дело в том, что младший адрес (192.168.104.0) используется для обозначения всей сети, а старший (192.168.104.255) — для так называемой широковещательной рассылки (сообщение отправляется всем компьютерам данной сети). Все узлы с адресами от 192.168.104.1 до 192.168.104.254 находятся в той же сети, что и данный компьютер.

Пример 5.2

Рассмотрим IP-адрес из примера 5.1, но с другой маской 255.255.255.248. В двоичной системе она содержит 29 единиц и 3 нуля.

Адрес:	192	.	168	.	105	.	123
	11000000.	10101000.	01101001.	01111	001		
Маска:	11111111.	11111111.	11111111.	11111	000		
	255	.	255	.	255	.	248

Узел с адресом 192.168.105.123/29 — это узел номер 3 (011_2) в сети 192.168.105.120 ($11000000.10101000.01101000.01111000_2$).

Так как на адрес узла выделено три бита (в маске три нуля), в такой сети доступно только $2^3 = 8$ адресов. Кроме того, учитывая, что два из них специальные (первый — номер сети, последний — широковещательный адрес), то в такую сеть может входить не более 6 узлов.



Необходимо помнить, что IP-адрес назначается не компьютеру, а каналу передачи данных (проводной сетевой карте, Wi-Fi-адаптеру, модему). Таким образом, один компьютер может иметь множество IP-адресов, например, если на нем установлены две сетевые карты.

Ниже приведены диапазоны IP-адресов (так называемые «серые» адреса), которые не используются в сети интернет. Они рекомендованы для локальных сетей:

- 10.0.0.0–10.255.255.255(10.0.0.0/8)
- 172.16.0.0–172.31.255.255(172.16.0.0/12)
- 192.168.0.0–192.168.255.255(192.168.0.0/16)

Несколько особое значение имеет IP-адрес, который начинается со 127. Его используют при взаимодействии процессов и тестировании программ в пределах одного устройства. Когда данные передаются по адресу 127.0.0.1, образуется «петля» (loopback). При этом не происходит передачи данных по сети, они сразу же возвращаются как только что принятые. При этом можно использовать не только 127.0.0.1, но и любой адрес из диапазона от 127.0.0.1 до 127.255.255.254.



Адреса сетей назначаются либо централизованно (сеть является частью Internet), либо произвольно (сеть работает автономно). Номера узлов в сети в обоих случаях назначаются по усмотрению администратора, не выходя из разрешенного для сети диапазона. Роль координатора в централизованном распределении IP-адресов ранее выполняла организация InterNIC (Internet Network Information Center). Однако в настоящее время в связи с ростом сети эта задача адресов стала очень сложной, и InterNIC передала часть своих полномочий другим организациям и крупным поставщикам интернет-услуг. Зачастую поставщики интернет-услуг получают диапазоны адресов у компании InterNIC, а затем распределяют их между своими абонентами.

5.1.4. Автоматизация процесса назначения IP-адресов

В локальных сетях при их создании администратор должен назначать IP-адреса всем устройствам. Эта задача регламентируется протоколом DHCP (Dynamic Host Configuration Protocol) и может быть решена следующими способами.

1. Ручное распределение. Данный способ весьма трудоемкий даже при небольшом размере сетей.
2. Автоматическое распределение. Для этого администратору необходимо задать диапазон IP-адресов. При этом во время старта системы DHCP-клиент (компьютер) посылает в сеть запрос на получение адреса. DHCP-сервер отвечает на этот запрос и посылает ответ, содержащий IP-адрес. При этом они находятся в одной сети. Между клиентом и его IP-адресом, как и в случае ручного распределения, имеется постоянное соответствие. Оно определяется в момент первого назначения IP-адреса сервером клиенту. Далее при всех последующих запросов сервер возвращает тот же IP-адрес.
3. Динамическое распределение. При таком подходе DHCP-сервер устанавливает IP-адрес клиенту на некоторое ограниченное время, которое называется временем аренды (lease duration). Такой способ предоставляет возможность повторно использовать IP-адрес для назначения другому узлу сети.

Одним из примеров работы протокола DHCP может служить ситуация, когда DHCP-клиент удаляется из подсети. Таким образом выданный ему IP-адрес становится свободным. Далее, когда устройство подключается к другой сети, ему автоматически выдается новый IP-адрес. При этом ни пользователь, ни администратор сети не участвуют в этом процессе. Это свойство весьма важно для мобильных устройств.

DHCP-сервер может установить клиенту не только IP-адрес. Кроме этого, он способен назначить и другие параметры стека TCP/IP, которые необходимы для более эффективной работы клиента (маску, IP-адрес сервера DNS, IP-адрес маршрутизатора по умолчанию и др.).

5.1.5. Доменные имена (DNS), URL-ссылки

Нам уже известно, что для идентификации узла в сети TCP/IP используют IP-адрес. Например, укажем в адресной строке браузера адрес `http://194.135.112.158`, мы попадем на страницу учебного портала ИТ ШКОЛЫ SAMSUNG. Однако пользователям не очень удобно работать с числовыми адресами. И что делать, если сайт необходимо перенести на другой сервер? Ведь это значит, что IP-адрес сменится и пользователи не смогут найти нашу страницу.

Для решения этих проблем в 1984 году была разработана система доменных имен (англ. DNS — Domain Name System), которая позволила установить в соответствие с определенными IP-адресами некоторые символьные имена, например, `myitschool.ru`. Это значит, что мы можем менять IP-адрес и при этом доменное имя останется прежним.

В конфигурации доменных имен применяется древовидная иерархия. При этом имена, у которых совпадают старшие составные части образуют домен имен (domain). Например, имена `myitschool.ru`, `yandex.ru` и `mail.ru` входят в домен `ru` (все эти имена имеют одну общую старшую часть — `ru`). А вот сайт `www.samsung.com` в домен `ru` не входит, так как он находится в домене `com`.



Понятие «домен» следует трактовать в рамках определенного контекста, так как он имеет множество значений. Кроме вышеупомянутого значения в компьютерной литературе можно встретить домен коллизий, домены Windows NT и др. Объединяет эти термины то, что они предназначены для описания некоторого подмножества компьютеров с каким-либо определенным свойством.

Если один домен является составной частью другого домена, то его можно назвать поддоменом (subdomain). Так как домены конструируются по принципу древовидной структуры, поддомен в свою очередь также является доменом (поддерево является деревом). Зачастую поддомен называют по имени, отличительной от других поддоменов его старшей составляющей. Устройства, включенные в один домен, при этом могут иметь IP-адреса, принадлежащие к разным сетям.

В сети Internet корневой домен управляется компанией InterNIC. Для каждой страны назначаются домены верхнего уровня, следуя стандарту ISO 3166. Для обозначения стран используются двухбуквенные сокращения (например, `ru` — Россия, `de` — Германия и т. п.). Кроме того, домены верхнего уровня назначаются на организационной основе. Для различных типов организаций приняты следующие обозначения:

- `com` — коммерческие организации;
- `edu` — образовательные организации;
- `org` — некоммерческие организации;
- `net` — организации, поддерживающие сети;
- `biz` — организации, связанные с бизнесом.

Подробнее со списком доменов верхнего уровня можно познакомиться здесь.

После доменов верхнего уровня зачастую в доменных именах присутствуют поддомены. Каждый поддомен может администрироваться в отдельно взятых регионах, организация и т. п. Далее поддомен может быть разбит еще на несколько поддоменов и т. д.

Большинство продаваемых доменов — это домены второго уровня. Имя домена второго уровня регистрируется на конкретное лицо или организацию. Такие услуги оказывают специальные организации — регистраторы доменных имен. Домены уровнем ниже второго чаще всего не обладают особой ценностью и их можно арендовать бесплатно.

Ранее в доменных именах допускалось использование только символов латинского алфавита, цифр и дефиса. В настоящее время возможна регистрация домена, содержащего другие знаки, входящие в кодировку UNICODE. Например, в России закреплен домен рф, в котором можно зарегистрировать домены второго уровня.

Резюмируя вышеизложенное, можно сделать вывод о том, что в сети интернет используется две системы адресов: доменные имена и IP-адресация. Для определения соответствия между ними на специальных DNS-серверах хранятся таблицы пар IP-адрес — доменное имя. По запросу доменного имени они возвращают ответ в виде IP-адреса клиента или наоборот. При вводе в браузер доменного имени (адреса сайта) сначала на DNS-сервер отправляется запрос с целью определить IP-адрес сервера. В случае положительного результата направляется запрос на получение веб-страницы, при этом драйвер протокола IP использует именно полученный IP-адрес, а не доменное имя.

Следует заметить, что одному доменному имени может соответствовать несколько IP-адресов (и наоборот). Данный подход применяется для распределения нагрузки на популярные ресурсы (например, www.google.com, www.vk.com и т. д.).

Адрес имеет не только каждый компьютер в интернете, но и каждый документ. Для обозначения такого адреса используется английское сокращение URL — Uniform Resource Locator — универсальный указатель ресурса. Типичный URL-адрес состоит из четырех частей: протокола, имени сервера (или его IP-адреса), каталога и имени документа (файла). Например, адрес

<http://myitschool.ru/is/Education/Markbook.aspx>

включает:

- протокол HTTP — протокол для обмена гипертекстовыми документами (это веб-страница);
- доменное имя сервера myitschool.ru;
- каталог на сервере [/is/Education/](http://myitschool.ru/is/Education/);
- имя файла [Markbook.aspx](http://myitschool.ru/is/Education/Markbook.aspx).

Иногда каталог и имя файла не указывают, например: <http://myitschool.ru>. Это означает, что мы обращаемся к главной странице сайта. Она может иметь разные имена в зависимости от настроек сервера (чаще всего — [index.htm](http://myitschool.ru/index.htm), [index.html](http://myitschool.ru/index.html), [index.php](http://myitschool.ru/index.php)).

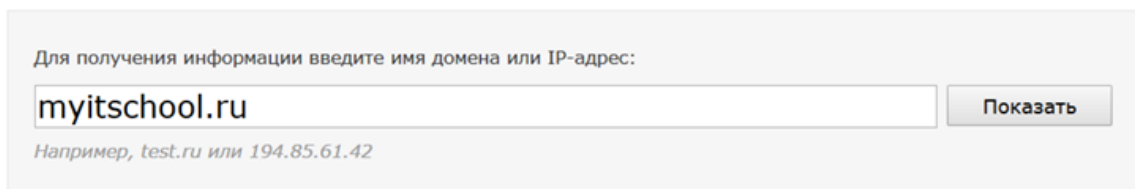
5.1.6. Сервисы работы с IP-адресами

В сети интернет существует большое количество сервисов для работы с IP-адресами. Одним из них является сервис WHOIS.

WHOIS — сетевой протокол прикладного уровня, базирующийся на протоколе TCP. Основное применение — получение регистрационных данных о владельцах доменных имен, IP-адресов и автономных систем. Протокол осуществляет доступ к публичным серверам баз данных регистраторов IP-адресов и регистраторов доменных имен.

Рассмотрим работу протокола WHOIS на примере сервиса RU-CENTER (АО «Региональный Сетевой Информационный Центр») — первого и крупнейшего в России профессионального регистратора доменов. Для этого нужно зайти на сайт <http://www.nic.ru> и затем выбрать сервис WHOIS (можно просто сразу перейти по адресу <http://www.nic.ru/whois/>). В окне «Для получения информации введите имя домена или IP-адрес» следует ввести IP-адрес интересующего нас сайта или его доменное имя, например, `myitschool.ru`. На экран будет выведена информация по IP-адресу или доменному имени (см. рис. 5.2), в том числе:

- имя владельца;
- имя регистратора;
- контактная информация владельца;
- дата регистрации доменного имени.



Информация о домене MYITSCHOOL.RU

Домен занят.

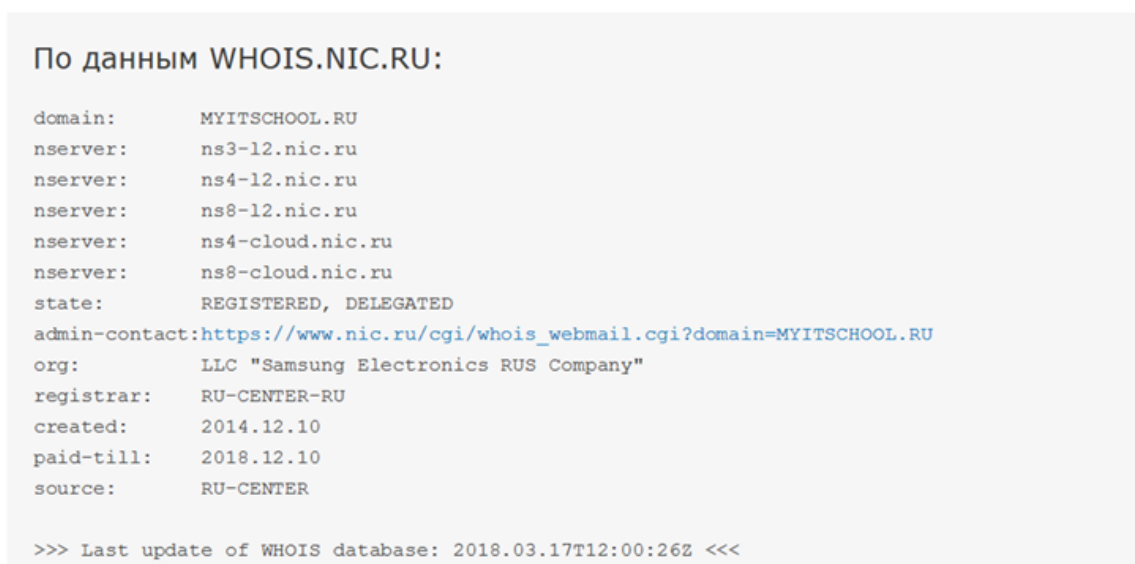


Рис. 5.2.

Не меньший интерес представляет сервис <http://2ip.ru/>. С его помощью можно узнать собственный IP-адрес, а также множество другой информации: имя провайдера, географию вашего местонахождения, используемую операционную систему, версию браузера и т. д.

Также есть возможность протестировать скорость интернет-соединения, выполнить онлайн-команду `ring` (которую мы подробнее изучим ниже), определить используемую CMS (систему управления сайтом), узнать физическое расстояние от географического расположения одного сайта или IP-адреса до другого, и многое другое.

5.1.7. Популярные сетевые команды

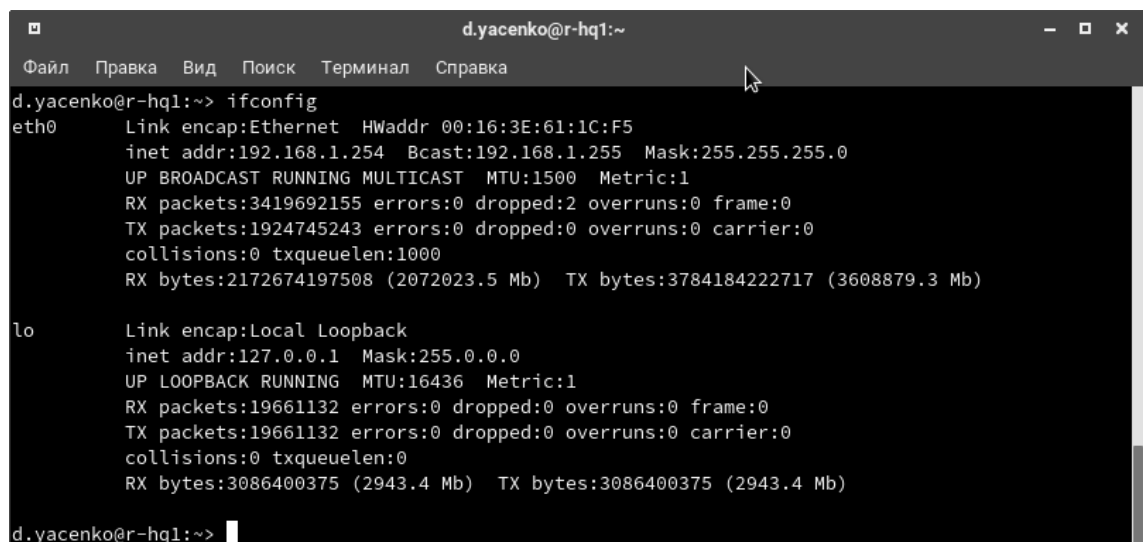
При работе нередко возникают задачи, связанные с проверкой доступности компьютеров и правильности работы службы DNS. Для этой цели администраторы зачастую используют утилиты командной строки. В Linux для работы в командной строке нужно запустить программу Терминал (Console), а в Windows — командный процессор cmd. Сделать это в Windows можно следующими способами.

1. Зайти в меню «Пуск» и выбрать «Программы» -> Служебные -> Командная строка.
2. Зайти в меню «Пуск», выбрать «Выполнить» (клавиша Windows + R), в окне «Открыть» написать cmd и нажать Ок или Enter на клавиатуре.

Теперь мы можем пользоваться консольными командами. Рассмотрим некоторые из них, а также возможности, которые они нам предоставляют.

Команда ipconfig (ifconfig)

Данная команда является одной из самых полезных. Она позволяет посмотреть конфигурацию адресов TCP/IP на текущий момент для всех установленных на данном компьютере сетевых адаптеров и коммутируемых соединений (см. рис. 5.3).



```
d.yacenko@r-hq1:~  
d.yacenko@r-hq1:~$ ifconfig  
eth0      Link encap:Ethernet  HWaddr 00:16:3E:61:1C:F5  
          inet addr:192.168.1.254  Bcast:192.168.1.255  Mask:255.255.255.0  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:3419692155  errors:0  dropped:2  overruns:0  frame:0  
          TX packets:1924745243  errors:0  dropped:0  overruns:0  carrier:0  
          collisions:0  txqueuelen:1000  
          RX bytes:2172674197508 (2072023.5 Mb)  TX bytes:3784184222717 (3608879.3 Mb)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          UP LOOPBACK RUNNING  MTU:16436  Metric:1  
          RX packets:19661132  errors:0  dropped:0  overruns:0  frame:0  
          TX packets:19661132  errors:0  dropped:0  overruns:0  carrier:0  
          collisions:0  txqueuelen:0  
          RX bytes:3086400375 (2943.4 Mb)  TX bytes:3086400375 (2943.4 Mb)  
  
d.yacenko@r-hq1:~$
```

Рис. 5.3.

Команду ipconfig (для ОС семейства Linux — ifconfig) следует первой использовать для диагностики возможных проблем с TCP/IP соединением. Чтобы получить справку по всем возможным ключам, запустите команду Ipconfig /? в Windows, Ifconfig -help в Linux.

Наиболее часто используют параметр /all, который позволяет получить большую часть информации, содержащейся в диалоговом окне свойств.

Команда ping

Данная команда используется для того, чтобы протестировать работоспособность сети. После ввода команды в командную строку команда отправляет запросы удаленному компьютеру с использованием специального протокола ECHO. После получения запроса удаленный компьютер тут же отправляет его обратно. Таким образом можно узнать наличие связи. В качестве параметра команды можно указывать как IP-адрес, так и доменное имя интересующего вас узла (рис. 5.4).

```
d.yacenko@linux-4cui:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
d.yacenko@linux-4cui:~> ping myitschool.ru  
PING myitschool.ru (194.135.112.158) 56(84) bytes of data.  
64 bytes from 194.135.112.158: icmp_seq=1 ttl=49 time=23.1 ms  
64 bytes from 194.135.112.158: icmp_seq=2 ttl=49 time=22.2 ms  
64 bytes from 194.135.112.158: icmp_seq=3 ttl=49 time=22.2 ms  
64 bytes from 194.135.112.158: icmp_seq=4 ttl=49 time=22.5 ms  
64 bytes from 194.135.112.158: icmp_seq=5 ttl=49 time=22.3 ms  
64 bytes from 194.135.112.158: icmp_seq=6 ttl=49 time=22.9 ms  
64 bytes from 194.135.112.158: icmp_seq=7 ttl=49 time=23.2 ms  
64 bytes from 194.135.112.158: icmp_seq=8 ttl=49 time=22.5 ms  
64 bytes from 194.135.112.158: icmp_seq=9 ttl=49 time=22.4 ms  
^C  
--- myitschool.ru ping statistics ---  
9 packets transmitted, 9 received, 0% packet loss, time 8012ms  
rtt min/avg/max/mdev = 22.252/22.643/23.286/0.396 ms  
d.yacenko@linux-4cui:~> |
```

Рис. 5.4.

С помощью команды `ping` удастся протестировать соединение между двумя узлами на низком физическом уровне. В случае, если запрос успешно вернулся, сеть работает нормально. Необходимо отметить, что если неисправности сети отсутствуют даже в том случае, если несколько пакетов утеряны. Как правило, это связано с перегруженностью сети, а также с тем, что многие маршрутизаторы отводят пакетам диагностики низкий приоритет. Случай, когда хотя бы один из посланных запросов вернулся, означает, что сеть работает исправно.

Для получения справки по команде следует использовать `ping /?` в Windows, `ping -help` в Linux.

Зачастую команду `ping` используют с параметром `-t`. С помощью данного параметра запросы к удаленному компьютеру повторяются до тех пор, пока не произойдет прерывание (комбинация клавиш `Ctrl+C`). Подобная практика полезна при устранении неисправностей и помогает проводить мониторинг результатов внесенных изменений.

Следующими распространенными вариантами выполнения команды `ping` являются запуски с ключами `-n` и `-l`. Они позволяют расширить параметры опроса. Ключ `-n` указывает фиксированное количество отправляемых пакетов, а параметр `-l` — размер одного пакета-передачи, ограниченного максимальным значением в 64 кБ.

Команда `tracert` (`tracert`)

Эта команда похожа на команду `ping`: обе команды посылают в удаленному компьютеру эхо-пакеты протокола Internet и далее ждут их возвращения. Ключевое отличие пакетов команды `tracert` от пакетов `ping` состоит в различном сроке их жизни. В первом случае пакеты отмечаются специальной меткой, которая означает, что они не могут быть проигнорированы ни одним из маршрутизаторов. Команда `tracert` (`tracert`) последовательно опрашивает и измеряет время задержки всех маршрутизаторов на пути прохождения пакета, пока не будет достигнут целевой хост (рис. 5.5).

```
d.yacenko@linux-4cui:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
d.yacenko@linux-4cui:~> traceroute rbc.ru  
traceroute to rbc.ru (80.68.253.9), 30 hops max, 60 byte packets  
 1 gw1.dolphin-ru.aanet.ru (192.168.1.4)  0.319 ms  0.278 ms  0.232 ms  
 2 router.ricom.org (91.232.188.1)  1.330 ms  1.252 ms  1.234 ms  
 3 78.25.81.157 (78.25.81.157)  36.111 ms  36.093 ms  36.061 ms  
 4 10.222.61.97 (10.222.61.97)  22.791 ms  22.756 ms  10.222.61.109 (10.222.61.109)  112.868 ms  
 5 10.222.61.58 (10.222.61.58)  21.518 ms  21.510 ms  21.483 ms  
 6 10.222.48.10 (10.222.48.10)  22.505 ms  22.380 ms  22.327 ms  
 7 10.222.71.30 (10.222.71.30)  21.714 ms  10.222.71.26 (10.222.71.26)  22.984 ms  22.835 ms  
 8 * * *  
 9 83.169.204.34 (83.169.204.34)  20.425 ms  83.169.204.26 (83.169.204.26)  20.777 ms  83.169.204.34 (83.169.204.34)  20.233 ms  
10 m9-cr05-ae11.0.msk.stream-internet.net (212.188.16.133)  21.968 ms  21.950 ms  21.880 ms  
11 m9-cr03-ae1.199.msk.stream-internet.net (195.34.53.50)  20.298 ms  19.885 ms  19.851 ms  
12 rbc.msk.stream-internet.net (212.188.16.214)  19.070 ms  18.709 ms  18.977 ms  
13 redirector.rbc.ru (80.68.253.9)  19.518 ms  18.730 ms  19.436 ms  
d.yacenko@linux-4cui:~> |
```

Рис. 5.5.

Таким образом, с помощью `tracert` можно получить подробный маршрут прохождения пакетов данных между компьютером, на котором была запущена `tracert`, и любым удаленным компьютером сети. Данный факт делает `tracert` весьма полезным инструментом обнаружения неисправности в сети. В случае обнаружения проблемы с подключением к веб-узлу или к какой-нибудь другой службе Internet можно определить участок, на котором она возникла.

Для получения помощи по возможным ключам команды используйте `tracert /?` в Windows, `traceroute -help` в Linux.

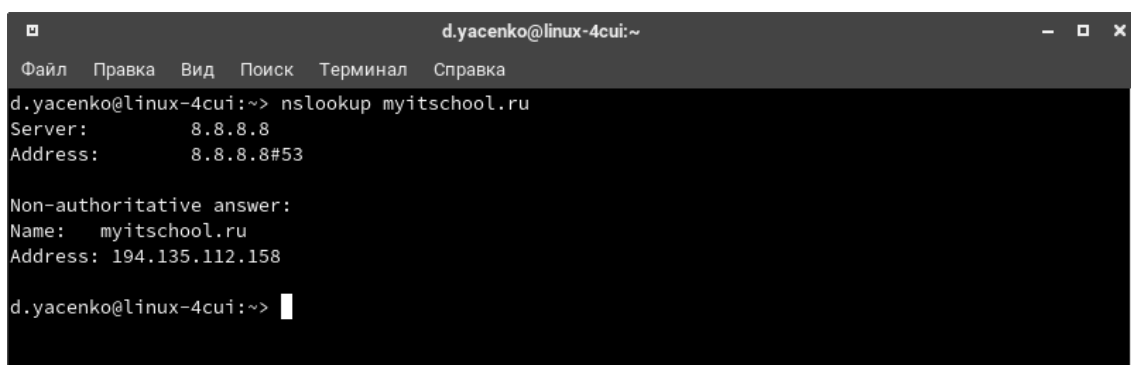
Команда `nslookup`

С помощью команды `nslookup` можно опрашивать конкретные DNS-серверы по конкретным типам записей DNS.

Для получения помощи по возможным ключам команды используйте `nslookup /help` в Windows, `man nslookup` в Linux (Ubuntu).

При запуске `nslookup` без параметров мы попадаем в интерактивный режим, где можно вводить различные команды, доменные имена и IP-адреса. Выход — команда `exit` или сочетание `Ctrl+C`.

Следующая команда должна показать IP-адрес сайта: `nslookup myitschool.ru` (рис. 5.6).

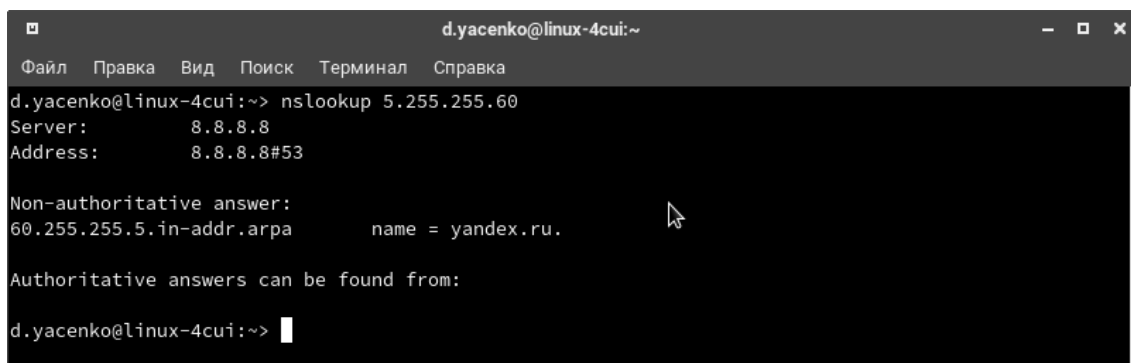


```
d.yacenko@linux-4cui:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
d.yacenko@linux-4cui:~> nslookup myitschool.ru  
Server:      8.8.8.8  
Address:     8.8.8.8#53  
  
Non-authoritative answer:  
Name:   myitschool.ru  
Address: 194.135.112.158  
  
d.yacenko@linux-4cui:~> 
```

Рис. 5.6.

Сообщение «Не заслуживающий доверия ответ» (Non-authoritative answer) говорит о том, что выполняющий запрос DNS-сервер не является владельцем зоны `myitschool.ru`, то есть записи для узла `myitschool.ru` в его базе отсутствуют, и для разрешения имени использовался рекурсивный запрос к другому DNS-серверу.

Команда `nslookup 195.208.64.58` должна отобразить имя узла, соответствующее введенному IP-адресу (рис. 5.7).



```
d.yacenko@linux-4cui:~  
Файл  Правка  Вид  Поиск  Терминал  Справка  
d.yacenko@linux-4cui:~> nslookup 5.255.255.60  
Server:      8.8.8.8  
Address:     8.8.8.8#53  
  
Non-authoritative answer:  
60.255.255.5.in-addr.arpa      name = yandex.ru.  
  
Authoritative answers can be found from:  
  
d.yacenko@linux-4cui:~> 
```

Рис. 5.7.

5.2. Веб-сервер, HTTP-запросы и ответы

Сайт: IT Академия SAMSUNG
Курс: MDev @ IT Академия Samsung
Книга: 5.2. Веб-сервер, HTTP-запросы и ответы
Напечатано.: Егор Беляев
Дата: Суббота, 18 Апрель 2020, 19:34

Оглавление

5.2.1. HTTP-протокол

5.2.2. Структура HTTP-запроса

5.2.3. Ответы сервера

5.2.4. Веб-сервер

5.2.5.* Реализация сервера на PHP

5.2.1. HTTP-протокол

Сегодня любой потребитель интернет-ресурсов сталкивается с HTTP-протоколом. Этот сетевой протокол, будучи разработанным в 90-х годах, и до сегодняшнего дня является основным для передачи информации в мировой сети. **HTTP (*HyperText Transfer Protocol*)** — протокол передачи гипертекста, протокол прикладного уровня по классификации ISO-OSI. Сегодня чаще всего используется версия 1.1 протокола. Полное описание протокола можно увидеть в документе RFC 2616.

При взаимодействии в интернете участников взаимодействия обычно разделяют на две роли — клиент и сервер. Где сервер (serv — обслуживание) — поставщик информации/услуг, а клиент — потребитель. HTTP также предполагает клиент-серверное взаимодействие. То есть программа-клиент (например, браузер) отправляет HTTP-запрос, сервер его принимает, обрабатывает и отправляет клиенту HTTP-ответ. Таким образом, взаимодействие клиента и сервера по HTTP похоже на обмен почтовыми посылками через службу доставки.

Несмотря на то что в WWW протокол используется для передачи текстовой и медиаинформации, многие API используют его для передачи данных. При этом данные чаще всего кодируются как строковые, или в более универсальном представлении как JSON/XML.

5.2.2. Структура HTTP-запроса

клиент и сервер: запросы и ответы



Рассмотрим, каким образом происходит обмен между клиентом и сервером по HTTP. При этом каждый запрос порождает следующую последовательность шагов.

1. DNS-запрос — поиск ближайшего DNS-сервера, чтобы из URL (например, yandex.ru) получить реальный IP-адрес.
2. Установка TCP-соединения с сервером по полученному IP-адресу.
3. Отправка данных HTTP-запроса.
4. Ожидание ответа, пока пакеты дойдут до сервера, он их обработает и вернет ответ назад.
5. Получение данных HTTP-ответа. Первые две операции, как правило, занимают больше всего времени.

Структура запроса

Каждый HTTP-запрос состоит из трех частей, которые следуют в указанном порядке.

1. Строка запроса — указан метод запроса (HTTP-метод), URI, версия протокола.
2. Заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения.
3. Тело сообщения — данные сообщения.

HTTP-спецификация определяет более строгое описание структуры запроса или ответа:

```
message = <start-line>
        *(<message-header>)
        CRLF
        [<message-body>]
<start-line> = Request-Line | Status-Line
<message-header> = Field-Name ':' Field-Value``
```

CRLF — обязательная «новая» строка между секцией заголовка и телом. Заголовок (message-header) и тело (message-body) запроса может отсутствовать, но строка запроса (start-line) есть всегда.

Строка запроса. Методы GET и POST

Строка запроса состоит из трех разделов, разделенных пробелом — Method URI Protocol. Рассмотрим пример запроса, который может использовать браузер при обращении к странице `http://myitschool.ru/book` — `GET /book HTTP/1.1`. В этом примере через пробел указываются метод GET, URI, указывающий нужный раздел сайта и версию протокола:

Метод	URI	Протокол
GET	/book	HTTP/1.1

В поле метод указывается операция, которую нужно осуществить с указанным ресурсом. Сервер может выполнять тот набор методов, который в нем запрограммирован. Однако в большинстве фреймворков реализованы следующие методы:

- **GET** — получить ресурс. При этом URI содержит информацию, позволяющую найти ресурс;
- **POST** — создать новый ресурс;
- **PUT** — обновить существующий ресурс;
- **DELETE** — удалить существующий ресурс.

Рассмотрим два наиболее распространенных метода более подробно.

- Метод GET предназначен для получения требуемой информации и передачи данных в адресной строке. Удобство использования метода GET заключается в том, что адрес со всеми параметрами можно использовать неоднократно, сохранив его, например, в закладки браузера, а также менять значения параметров прямо в адресной строке.
- Метод POST посылает на сервер данные в запросе браузера. Это позволяет отправлять большее количество данных, чем доступно методу GET, поскольку у него установлено ограничение в 4 Кб. Большие объемы данных используются в форумах, почтовых службах, заполнении базы данных, при пересылке файлов и др.

Таким образом между методами POST и GET следующие различия (см. табл. 5.1).

Характеристика	GET	POST
Тело запроса	Отсутствует	Содержит данные
Максимальный объем	255 символов	8 КБ
Кэширование	Да	Нет

Табл. 5.1.

Кроме того, следует помнить, что передача дополнительных параметров у GET и POST запросов происходит по-разному. У POST-запроса дополнительные параметры передаются в теле запроса, а у GET-запроса дополнительные параметры передаются в URL. Синтаксис следующий:

```
URI[?param1=value1[&paramN=valueN]]
```

Например, если в браузере набрать адрес `https://www.google.ru/search?q=myitschool+rostov&ie=utf-8`, то браузер отправит серверу `google.ru` следующий GET-запрос:

```
GET /search?q=myitschool+rostov&ie=utf-8 HTTP/1.1
```

Параметры этого запроса следующие:

Имя параметра	Значение
q	myitschool+rostov
ie	utf-8



Не следует передавать в параметрах GET-запросов конфиденциальных данных, так как они присутствуют в запросе в открытом виде и будут легкодоступны посторонним. Например, в истории просмотра браузера или в каких-либо других журналах.

Заголовки запроса

Заголовки запроса задают его параметры. В зависимости от назначения заголовки HTTP-запросов разделяют на 4 группы:

- general;
- request specific;
- response specific;
- entity.

Из General-заголовков (применяются как для запроса, так и ответа сервера) наиболее распространенные:

- Date указывает дату запроса, пример: Date: Fri, 29 Jun 2016 09:20:45 GMT;
- MIME-version указывает версию MIME (по умолчанию 1.0), пример: MIME-version: 1.0;
- Pragma содержит указания для таких промежуточных агентов, как прокси и шлюзы, пример: Pragma: no-cache.

Request-заголовки:

- Host содержит доменное имя вебсервера, например: Host: www.google.ru;
- Authorization содержит информацию об аутентификации, например: Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==;
- From — поле, в котором браузер может посылать полный e-mail адрес пользователя серверу, например: From: info@myitschool.ru;
- User-Agent указывает наименование и версию браузера, ОС, например: User-Agent: Mozilla/3.0.

***Протокол HTTPS**

Сам по себе протокол HTTP не предполагает использование шифрования для передачи информации. Однако для HTTP есть распространенное расширение, которое реализует упаковку передаваемых данных в криптографический протокол SSL или TLS. Название этого расширения — HTTPS (HyperText Transfer Protocol Secure). В отличие от стандартного для HTTP 80-го порта, для HTTPS обычно используется TCP-порт 443. HTTPS широко используется для защиты информации от перехвата, а также, как правило, обеспечивает защиту от атак вида man-in-the-middle — в том случае, если сертификат проверяется на клиенте, и на компьютере пользователя не были внедрены сертификаты центра сертификации злоумышленника. На данный момент HTTPS поддерживается всеми популярными веб-браузерами (Протокол HTTP/HTTPS [Электронный ресурс]/ НОУ Интуит. — <http://www.intuit.ru/studies/courses/4455/712/lecture/21291?page=2>).

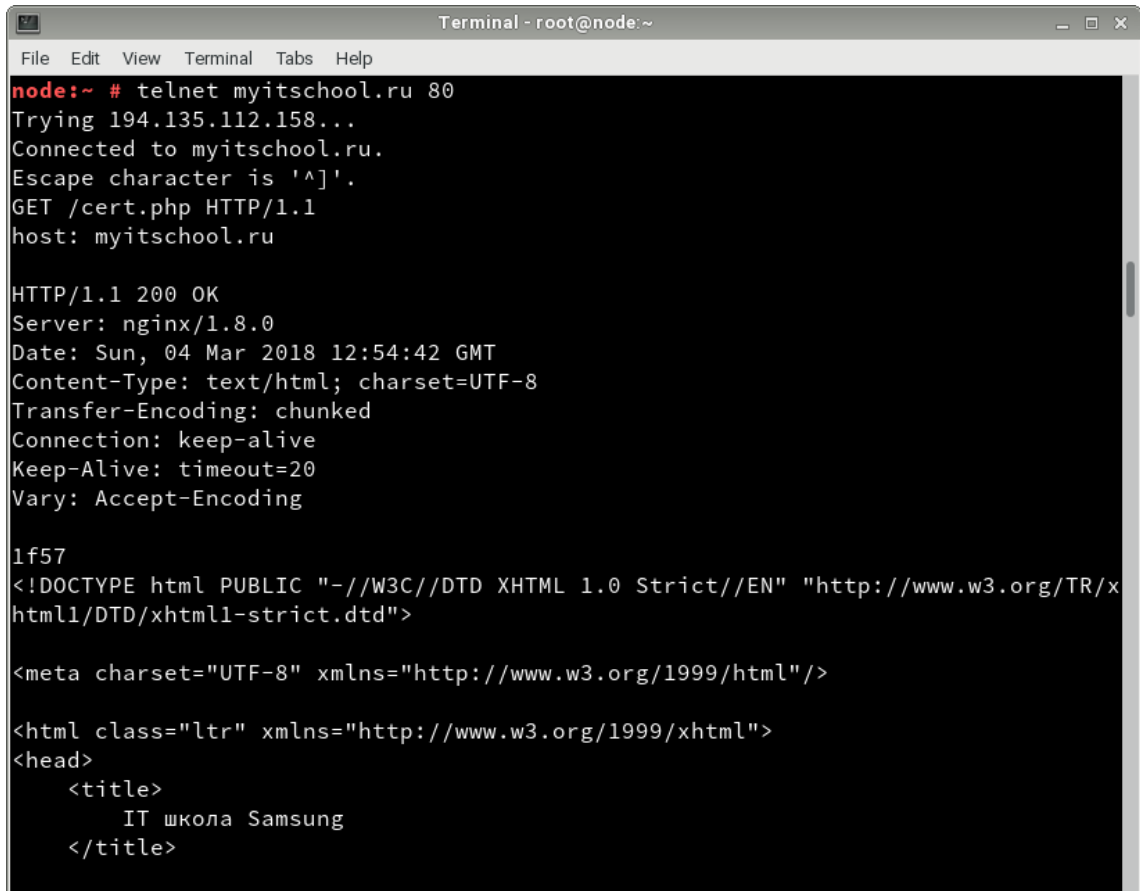
Пример 5.3

Хороший способ познакомиться с протоколом HTTP — это поработать с каким-нибудь веб-ресурсом вручную. Для этого необходимо воспользоваться любой подходящей утилитой командной строки, например, telnet. В ОС Windows, возможно, придется ее установить. Как это сделать описано здесь: <http://windows.microsoft.com/ru-ru/windows/telnet-faq>.

В качестве примера симулируем запрос браузером страницы IT School Samsung — <http://myitschool.ru/cert.php>. Для этого откроем окно командной оболочки (в Windows WIN+r -> cmd -> <Enter>). В открывшемся окне оболочки запустим команду telnet myitschool.ru 80. В открывшемся соединении к серверу введем следующие строки HTTP протокола:

- GET /cert.php HTTP/1.1 <Enter>
- host: myitschool.ru <Enter>
- <Enter>

В случае успешного ответа сервера (код 200) в консоль будет выведен большой HTML-документ. Для закрытия соединения введите CTRL+] и далее exit (см. рис. 5.8).



```
Terminal - root@node:~
File Edit View Terminal Tabs Help
node:~ # telnet myitschool.ru 80
Trying 194.135.112.158...
Connected to myitschool.ru.
Escape character is '^]'.
GET /cert.php HTTP/1.1
host: myitschool.ru

HTTP/1.1 200 OK
Server: nginx/1.8.0
Date: Sun, 04 Mar 2018 12:54:42 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Vary: Accept-Encoding

1f57
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<meta charset="UTF-8" xmlns="http://www.w3.org/1999/html"/>

<html class="ltr" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>
    IT школа Samsung
  </title>
```

Рис. 5.8.

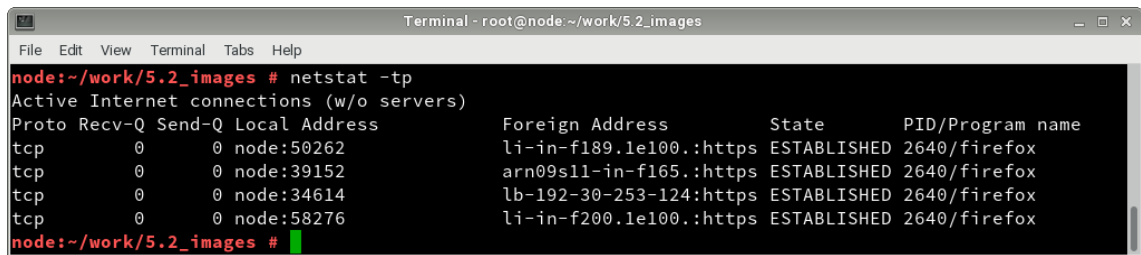
В приведенном примере производился HTTP обмен через сетевое соединение, которое было открыто к серверу при помощи программы telnet. Можно воспользоваться командой netstat, чтобы увидеть список текущих соединений. Для этого необходимо в системе Windows набрать:

```
netstat -f -o
```

Для ОС Linux, Mac и т. д. команда следующая:

```
netstat -tp
```

Будет выведен список TCP-соединений, причем, в столбце Local Address будет указан IP-адрес вашего компьютера и через двоеточие порт, выделенный ОС, с которым связан идентификатор работающего программного процесса (последний столбец) (см. рис. 5.9).

A terminal window titled "Terminal - root@node:~/work/5.2_images" showing the output of the command "netstat -tp". The output lists active Internet connections (w/o servers) for TCP, showing local addresses, foreign addresses, states, and PID/Program names. The connections are established with Firefox. The terminal prompt is "node:~/work/5.2_images #".

```
node:~/work/5.2_images # netstat -tp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 node:50262             li-in-f189.1e100.:https ESTABLISHED 2640/firefox
tcp        0      0 node:39152             arn09s11-in-f165.:https ESTABLISHED 2640/firefox
tcp        0      0 node:34614             lb-192-30-253-124:https ESTABLISHED 2640/firefox
tcp        0      0 node:58276             li-in-f200.1e100.:https ESTABLISHED 2640/firefox
node:~/work/5.2_images #
```

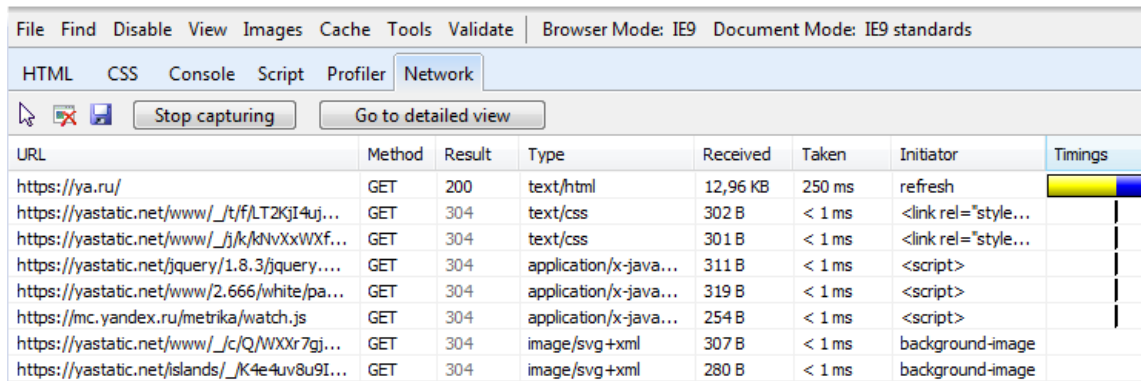
Рис. 5.9.

Эта пара IP-адрес и порт и задают клиентский сокет (socket — розетка, разъем). Когда с сервера на указанный порт приходит ответ, операционная система ищет связанный с ним сокет и помещает в него данные.

На сервере сокет работает по другому механизму. Приложение имеет возможность создать «слушающий» сокет и, привязав его к порту ОС сервера, получать адресованные ему пакеты данных.

5.2.3. Ответы сервера

Познакомимся с тем, как внутри устроены запросы и ответы к серверу. Для этого откроем страницу ya.ru (или любую другую на выбор). Далее нужно открыть в настройках браузера «Developer Tools» (Ctrl+Shift+I в Chrome, F12 в IE), перейти в вкладку «Network» (в IE еще необходимо нажать кнопку «Start Capture» на панели). Обновим текущую страницу, чтобы появилась информация. Будет выведена табличка, в которой каждая строчка — это некоторый запрос к серверу. В современном мире даже самая простая страница отправляет десяток запросов к серверу (рис. 5.10).



URL	Method	Result	Type	Received	Taken	Initiator	Timings
https://ya.ru/	GET	200	text/html	12,96 KB	250 ms	refresh	
https://yastatic.net/www/_/t/f/LT2KjI4uj...	GET	304	text/css	302 B	< 1 ms	<link rel="style...	
https://yastatic.net/www/_/j/k/kNvXxWXf...	GET	304	text/css	301 B	< 1 ms	<link rel="style...	
https://yastatic.net/jquery/1.8.3/jquery....	GET	304	application/x-java...	311 B	< 1 ms	<script>	
https://yastatic.net/www/2.666/white/pa...	GET	304	application/x-java...	319 B	< 1 ms	<script>	
https://mc.yandex.ru/metrika/watch.js	GET	304	application/x-java...	254 B	< 1 ms	<script>	
https://yastatic.net/www/_/c/Q/WXr7gj...	GET	304	image/svg+xml	307 B	< 1 ms	background-image	
https://yastatic.net/islands/_/K4e4uv8u9I...	GET	304	image/svg+xml	280 B	< 1 ms	background-image	

Рис. 5.10.

В этом примере видно, как и какие HTTP-запросы (request) отправляются к веб-серверу. И здесь же видно полученные от сервера HTTP-ответы (response). HTTP-ответ устроен примерно, как и запрос. В теле ответа можно увидеть получаемую от сервера веб-страницу, а в заголовках ответа есть очень важное поле — код статуса ответа (*status*).

Этот код помогает клиенту понять, как именно интерпретировать ответ сервера, а также предусмотреть обработку нестандартных ситуаций. Спецификация HTTP определяет трехзначные коды, описывающие состояния HTTP-ответа, которое получено от сервера. Причем они объединены в 5 групп, начинающихся с 1 до 5. В таблице 5.2 ниже приведены наиболее часто используемые коды.

Код	Описание кода состояния HTTP
-----	------------------------------

Информационные коды. Этот класс состояний был добавлен в HTTP/1.1 и носит условный характер. Например, сервер может послать заголовок <i>Expect: 100-continue</i> , что будет означать для клиента сигнал к продолжению отправки оставшейся части запроса. Если запрос отправлен полностью, то клиент проигнорирует этот заголовок. HTTP/1.0-клиенты проигнорируют его в любом случае	
1xx	Успешное выполнение запроса. Эта группа состояний говорит клиенту, что все прошло хорошо и запрос успешно обработан. Чаще всего встречается код 200 ОК. В случае GET-метода вместе с таким кодом в теле HTTP-ответа отправляется запрашиваемый ресурс
2xx	200 Запрос был обработан успешно
	201 Объект создан
	202 Информация принята
	203 Информация, которая не заслуживает доверия
	204 Нет содержимого
	205 Сбросить содержимое
	206 Частичное содержимое (например, при «докачке» файлов)
Перенаправление (чтобы выполнить запрос, нужны какие-либо действия). Коды этого диапазона означают, что клиенту необходимо сделать другой запрос, чтобы получить требуемые ресурсы. Наиболее частый сценарий — запрос на другой URL	
3xx	

Код	Описание кода состояния HTTP
300	Несколько вариантов на выбор
301	Ресурс перемещен на постоянной основе
302	Ресурс перемещен временно
303	Смотрите другой ресурс
304	Содержимое не изменилось
305	Используйте прокси-сервер
4xx	Проблема связана не с сервером, а с запросом. Эти коды используются, когда сервер полагает, что в результате запроса клиент допустил ошибку, например, запрашивает несуществующий ресурс. Наиболее распространенный код — 404. С ним сталкивались, пожалуй, все. Он говорит клиенту, что запрошенный ресурс не существует на сервере
400	Некорректный запрос
401	Нет разрешения на просмотр ресурса
402	Требуется оплата
403	Доступ запрещен
404	Ресурс не найден
405	Недопустимый метод
406	Неприемлемый запрос
407	Необходима регистрация на прокси-сервере
408	Время обработки запроса истекло
409	Конфликт
410	Ресурса больше нет
411	Необходимо указать длину
412	Не выполнено предварительное условие
413	Запрашиваемый элемент слишком велик
414	Идентификатор ресурса (URI) слишком длинный
415	Неподдерживаемый тип ресурса
5xx	Ошибки на сервере. Этот класс состояний говорит о серверной ошибке в процессе обработки клиентского запроса. Чаще всего встречается ошибка 500
500	Внутренняя ошибка сервера
501	Функция не реализована
502	Дефект шлюза
503	Служба недоступна
504	Время прохождения через шлюз истекло
505	Неподдерживаемая версия HTTP

Табл. 5.2.

5.2.4. Веб-сервер

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, обычно вместе с HTML-страницей, изображением, файлом или другими данными. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает. Клиент передает веб-серверу запросы на получение ресурсов, идентифицируемые по URL-адресами. **Ресурсы** — это хранимые на сервере HTML-файлы, изображения, медиапотoki или другие данные, которые необходимы клиенту. В ответ веб-сервер передает клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP. Существует множество веб-серверов, написанных на разных языках. Самыми распространенными являются Apache и nginx.

Современные веб-сервера могут выдавать клиенту не только статично хранимые файлы, но и динамически формируемые данные, которые образуются в результате вызова программного кода, написанного на различных языках программирования. Такой код для языка Java называется **сервлетом**. Наиболее распространенный сервер с открытым исходным кодом и поддержкой сервлетов — Tomcat, разрабатываемый Apache Software Foundation. Чтобы подчеркнуть именно особенность этого сервера, выполняется Java-код при обращении клиента, говорят — Tomcat — контейнер сервлетов, реализующий спецификацию сервлетов и спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF). Сам Tomcat тоже написан на языке Java.

Установка Tomcat в системе Windows

Чтобы установить Tomcat в ОС Windows, нужно зайти на сайт <http://tomcat.apache.org/> из раздела *Downloads* выбрать последнюю актуальную версию (на данный момент это Tomcat 9.0), нажать на ссылку *32-bit/64-bit Windows Service Installer* в разделе *Binary Distributions*. Запускаем скаченный установщик. На приветственной странице, следуя инструкциям, нажимаем на кнопку *Next*, в следующем окне принимаем лицензионное соглашение *I Agree*. В третьем окне выбираем элементы (см. рис. 5.11).

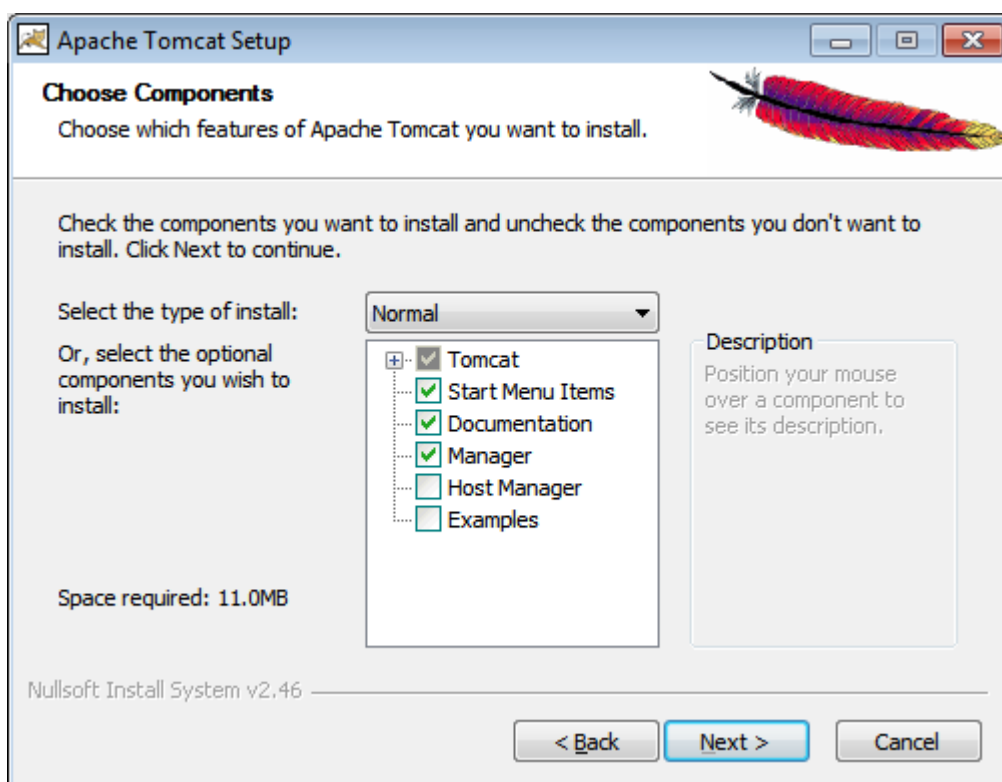


Рис. 5.11.

В большинстве случаев можно просто оставить настройки по умолчанию (см. рис. 5.12).

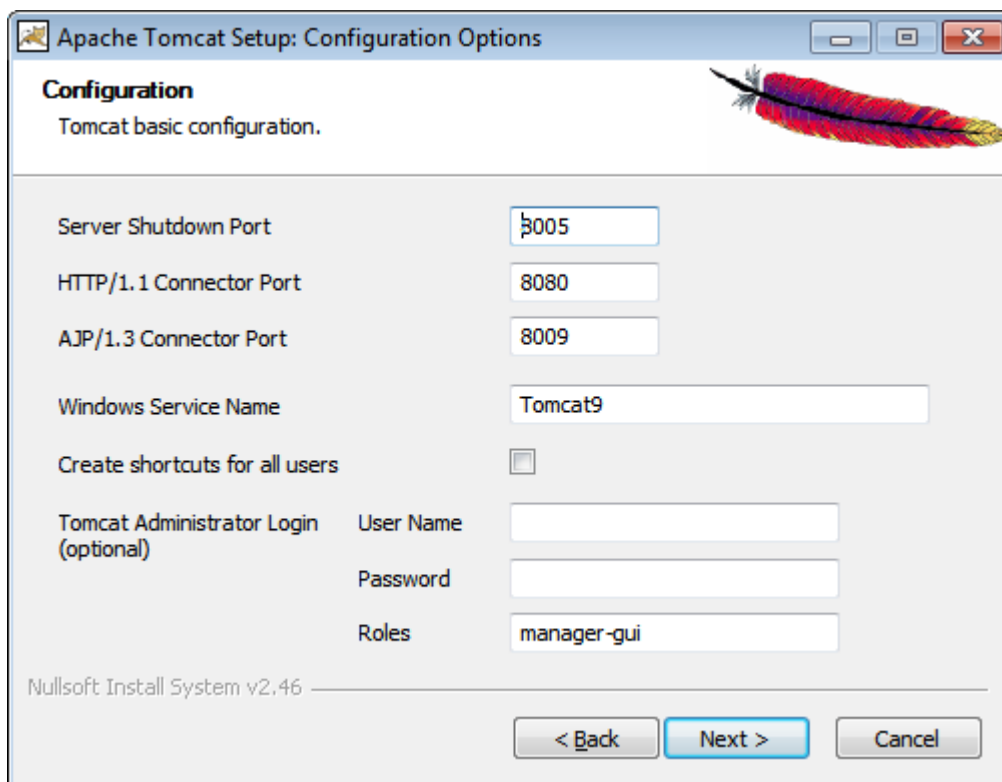


Рис. 5.12.

Здесь необходимо указать путь к установленному в системе JRE (см. рис. 5.13).

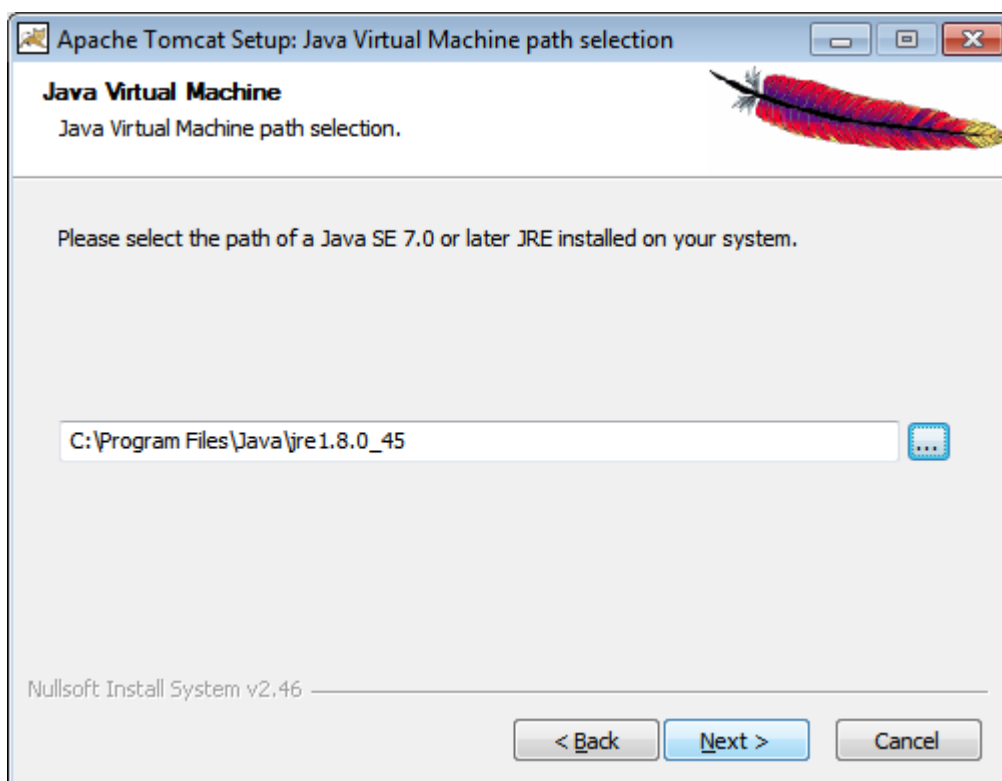


Рис. 5.13.

И наконец, папку установки (см. рис. 5.14).

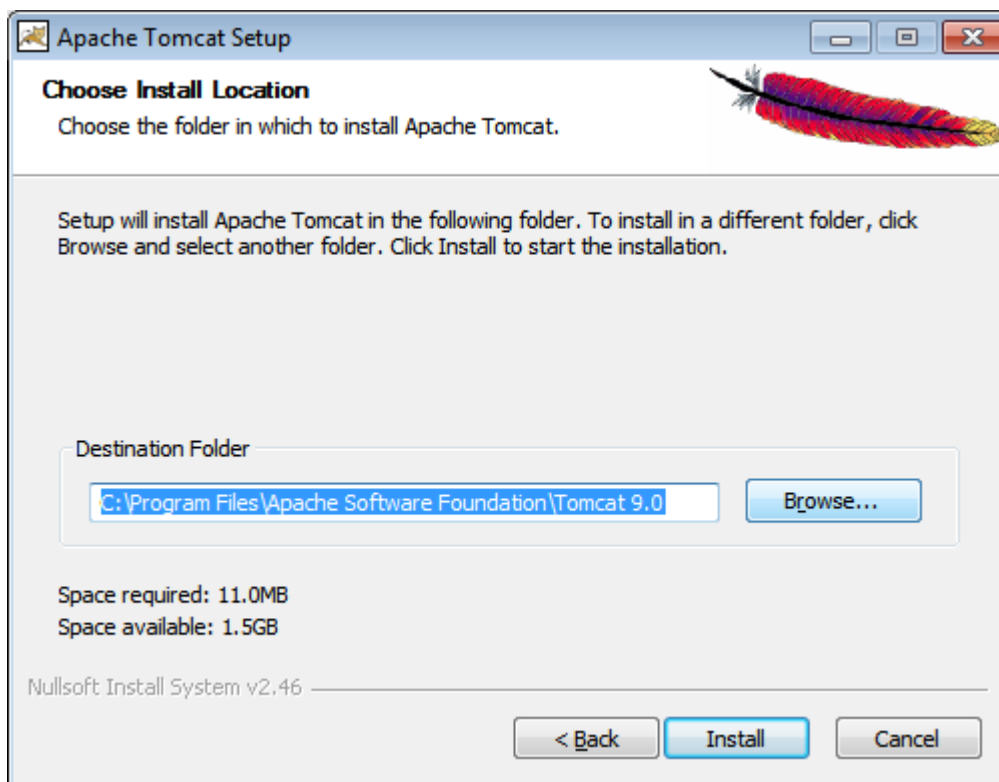


Рис. 5.14.

В последнем окне выбираем опцию Run Apache Tomcat и нажимаем Finish, ждем конца процесса установки. И по окончании установки появится иконка Tomcat.

Установка Tomcat в ОС Linux (Ubuntu)

Чтобы установить Tomcat под Linux нужно запустить терминал и набрать команду: `sudo apt-get install tomcat7`. Далее в браузере в адресной строке запускаем `http://localhost:8080` и видим страницу, которую выдает на ваш запрос сервер Tomcat.

Фреймворк Spring и Eclipse

Преимущества фреймворка общеизвестны, однако подробнее о Spring можно узнать на его сайте <http://spring-projects.ru>. Tomcat также входит в Java-фреймворк Spring. Для разработки веб-приложений используется шаблон Spring MVC. Однако для того, чтобы развернуть такое приложение от программиста, требуется написать конфигурационный код, причем, как правило, для типовых проектов он одинаков. Поэтому у начинающих разработчиков в особенности пользуется популярностью инструмент Spring Boot, который автоматизирует и тем самым облегчает этот процесс.



Обратите внимание, что в плагине STS имеется встроенный сервер Tomcat, который запускается, когда в среде разработки запускается на выполнение проект SpringBoot. Таким образом, если на текущем компьютере уже запущен ранее установленный сервер Tomcat с настройками по умолчанию, то произойдет конфликт при попытке открытия на прослушивание серверного сокета на порту 8080. Решением этой проблемы может быть изменение порта на прослушивание при установке сервера, либо остановка ранее установленного сервера, либо можно просто его не устанавливать (удалить).

Spring Boot в Eclipse можно установить с помощью плагина Spring Tools Suite (STS), который легко подключается через Help -> Eclipse Marketplace. Необходимо набрать в строке поиска STS, далее нажать Install на найденном плагине STS (первый на рис. 5.15).

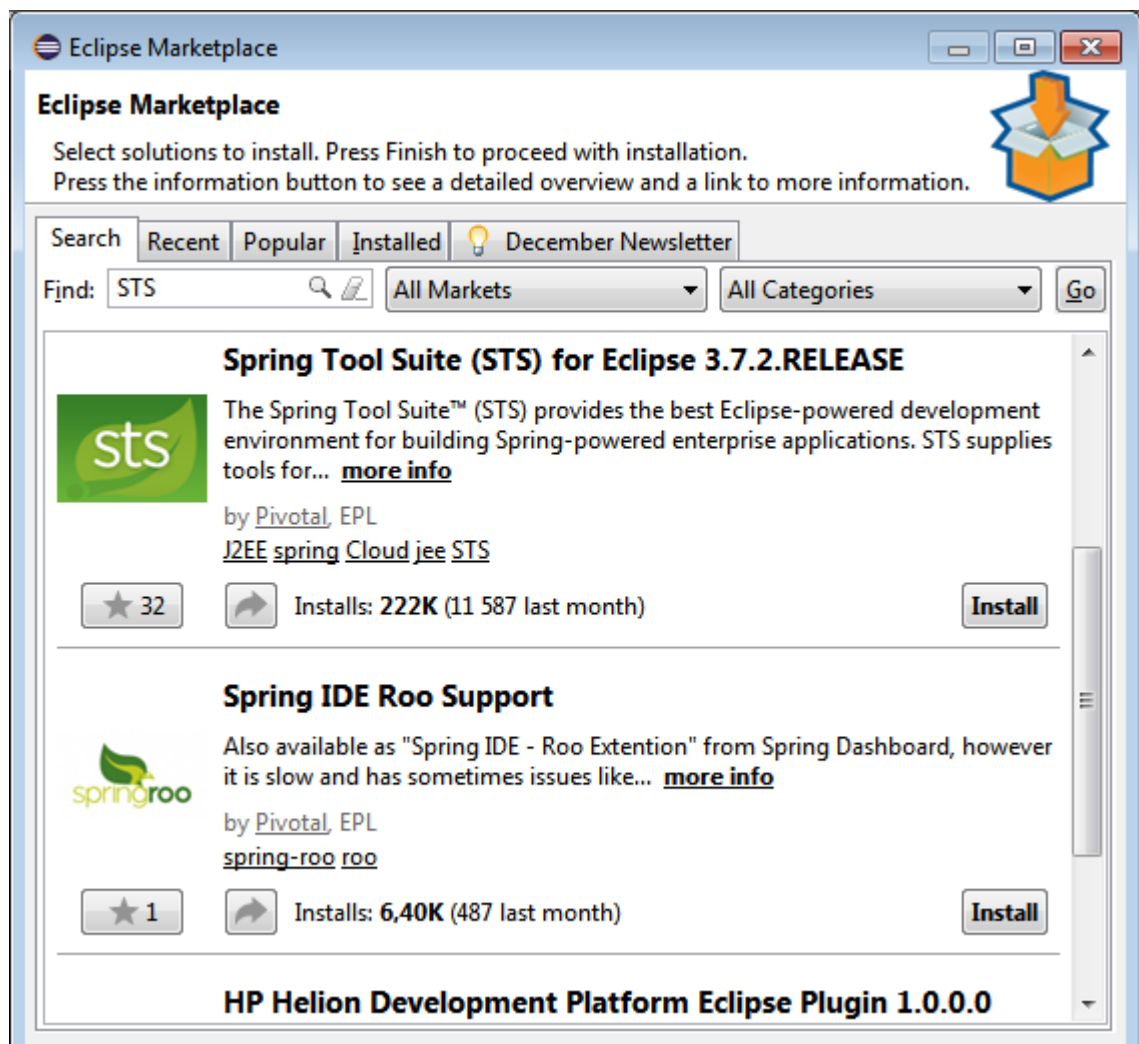


Рис. 5.15.

После поиска всех компонентов в окне Confirm Selected Futures необходимо выбрать все и нажать Confirm. В следующем окне нужно принять лицензионное соглашение и нажимать Finish. Теперь все готово к разработке простого клиент-серверного приложения на базе Spring Boot.

Пример 5.4

В этом примере продемонстрируем взаимодействие браузера (HTML-страницы) и Java-программы в контейнере веб-сервера по протоколу HTTP. Эта работа должна происходить по следующему алгоритму:

1. Простейшая html-форма, которая с помощью браузера будет отправлять значения двух полей Имя и Фамилия на сервер по кнопке «Отправить». Например:

Фамилия:

Petrov

Имя:

Sergey

Отправить

2. Сервер в ответ должен возвратить в браузер одну строку, сформированную из исходных полей, например, «Petrov S.».
3. Эта строка должна появиться в браузере.

Создание html-страницы

Ниже приведен HTML-код простейшей формы (файл 5.4.html):

```
<html>
<form name='test' method='post' action='http://localhost:8080/'>
  <p><b>Фамилия:</b><br>
    <input type='text' name='lastname' size='40'>
  </p>
  <p><b>Имя:</b><br>
    <input type='text' name='firstname' size='40'>
  </p>
  <p>
    <input type='submit' value='Отправить'>
  </p>
</form>
</html>
```

Не вдаваясь в тонкости языка разметки HTML, можно сказать, что тут описана форма с двумя полями ввода и одной кнопкой. В результате нажатия на кнопку «Отправить» произойдет отправка GET-запроса по URL — `http://localhost:8080/`, откуда затем и будет выведен результат работы серверной части приложения.

Создание в Eclipse Spring-проекта

Для выполнения пункта 2 задания необходимо в Eclipse (STS-плагин должен быть установлен) создать проект Spring, воспользовавшись мастером (рис. 5.16).

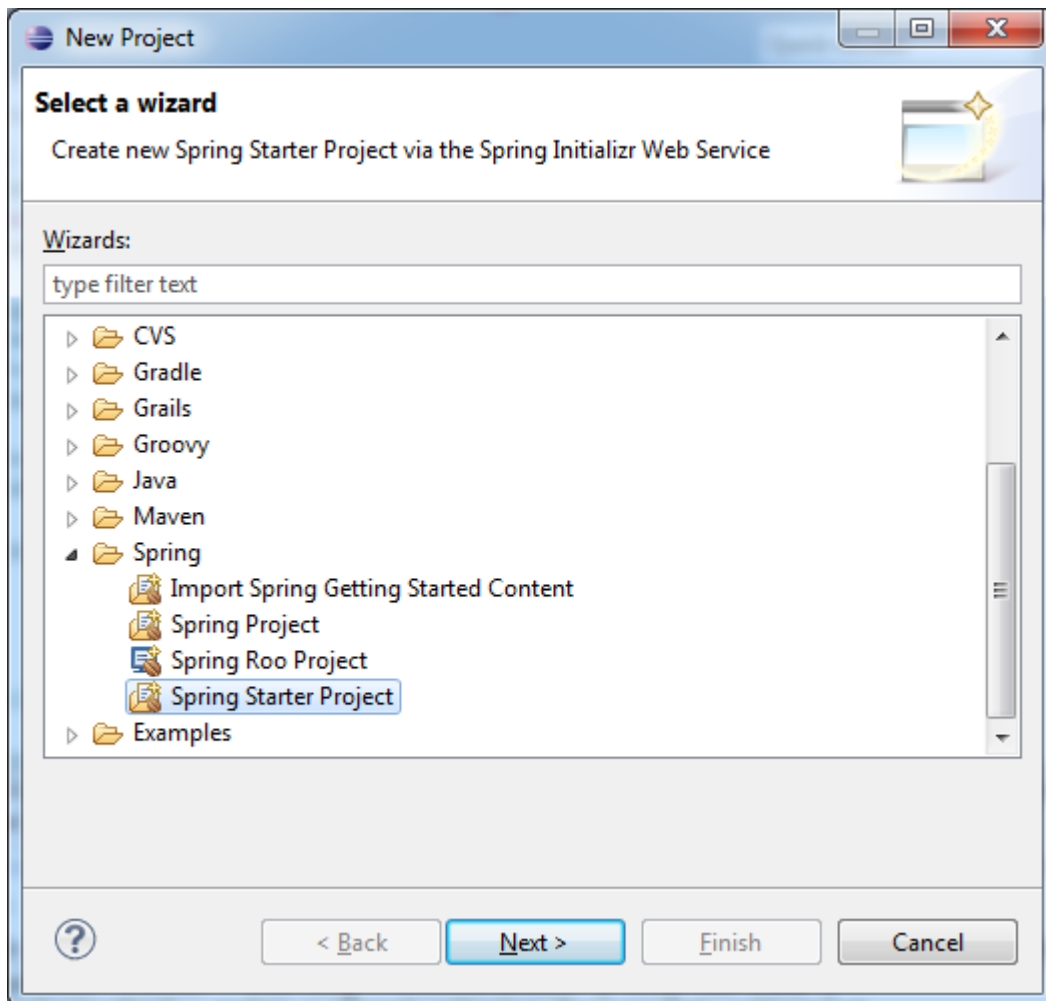


Рис. 5.16.

В открывшемся мастере из полей необходимым для заполнения является только название проекта (name). Остальные можно оставить как есть (см. рис. 5.17).

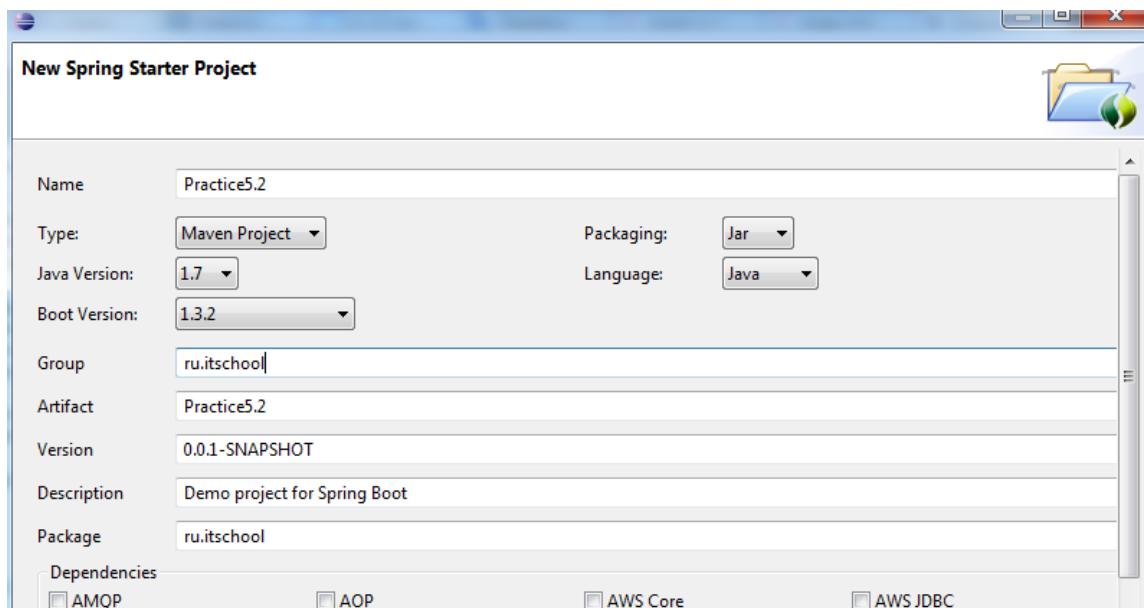


Рис. 5.17.

Однако в нижней части окна Dependencies необходимо установить галочку Web, чтобы в проекте появились необходимые зависимости, подключающие библиотеки для обработки POST и GET-запросов (рис. 5.18).

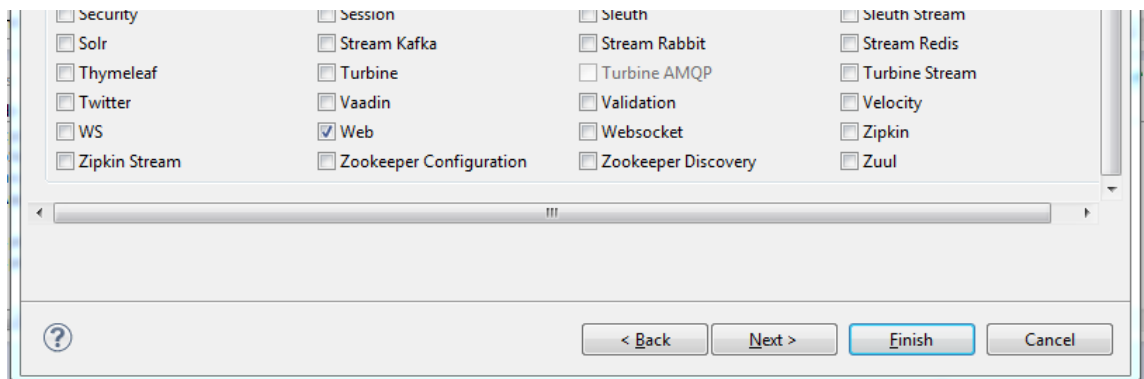


Рис. 5.18.

Далее необходимо нажать кнопку Finish, после чего проект будет полностью сформирован и открыт.

Обработка запросов на сервере

В созданном приложении автоматически был сгенерирован главный класс Application с кодом запуска сервера. В нем аннотация `@SpringBootApplication` позволяет Spring Boot сообщить о необходимости выполнения ряда действий для генерации другого кода, конфигураций и т. д. Для запуска Spring-приложения в имеющемся методе `main` вызывается метод `SpringApplication.run()`, который автоматически формирует веб-страницу из имеющихся в приложении классов:

```
package ru.itschool;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Создадим класс для получения из GET-запроса переданной информации (фамилия, имя) и обратного ответа сервера:

```
package ru.samsung.itschool;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class HttpControllerREST extends HttpServlet {
    @RequestMapping("/")
    public String index(HttpServletRequest request, HttpServletResponse response) {
        if (request.getParameter("lastname") != null || request.getParameter("firstname") != null) {
            if (!request.getParameter("lastname").equals("") && !request.getParameter("firstname").equals("")) {
                String lastname = request.getParameter("lastname");
                String firstname = request.getParameter("firstname");
                firstname = firstname.substring(0, 1); //Первая буква
                return lastname + " " + firstname + ".";
            } else {
                return "No POST data lastname or firstname";
            }
        }
        return "Not POST data ";
    }
}
```

Аннотация `@RestController` указывает на то, что класс готов принимать веб-запросы, используя Spring MVC. `@RequestMapping("/")`, указывает, по какому адресу (URI) будет доступна страница. При запуске приложения (ПКМ -> Run As -> SpringBootApplication) встроенный сервер Tomcat также будет запущен и будет доступен по адресу localhost на порту 8080.

Теперь можно проверить работу примера, открыв веб-форму и заполнив, нажать кнопку «Отправить».

5.2.5.* Реализация сервера на PHP

PHP (англ. *PHP: Hypertext Preprocessor* — «PHP: препроцессор гипертекста»; произносится пи-эйч-пи) — скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений. В настоящее время поддерживается подавляющим большинством хостинг-провайдеров и является одним из лидеров среди языков, применяющихся для создания динамических веб-сайтов.

PHP — язык программирования, исполняемый на стороне веб-сервера, спроектированный Расмусом Лерддорфом (Rasmus Lerdorf). Язык оказался достаточно гибким и мощным, поэтому приобрел большую популярность и используется в проектах любого масштаба: от простого блога до крупнейших веб-приложений в интернете.

Он получил широкое распространение, благодаря множеству преимуществ, начиная с того, что является свободно распространяемым под особой лицензией (PHP license), легок в освоении, имеет развитую поддержку баз данных, библиотеки и расширения языка, может быть развернут почти на любом сервере, на широком спектре аппаратных платформ и операционных систем.

Среди недостатков PHP разработчики отмечают несогласованность его синтаксиса, он не подходит для создания десктопных приложений или системных компонентов и, самое главное, сложилось мнение, что веб-приложения, написанные на PHP, зачастую имеют проблемы с безопасностью. Поэтому во многих крупных компаниях, банках существует политика запрета на использование PHP.

Однако, это не мешает его широчайшему распространению на миллионах веб-серверов. К крупнейшим сайтам, использующим PHP, относятся Facebook, Wikipedia и др.

Наиболее популярный PHP сервер — это HTTP сервер Apache. Apache — свободно распространяемый кроссплатформенный сервер, полностью разрабатываемый Apache Software Foundation. Его основными достоинствами считаются надежность и гибкость конфигурации, поддерживает IPv6. Помимо PHP Apache поддерживает языки программирования: Python, Ruby, Perl, ASP, Tcl. По некоторым данным Apache установлен на более чем 60% серверах в мире.

Как установить Apache и PHP5?

В ОС Windows 7 этот процесс подробно описан в статье здесь:
<http://www.q2w3.ru/2011/01/12/3093/>

Для установки Apache и PHP5 в ОС Linux (Ubuntu) можно воспользоваться руководством отсюда: <http://help.ubuntu.ru/wiki/apachemysqlphp>

Функциональность, аналогичная ранее рассмотренной на Java-сервере Tomcat, может быть реализована с помощью кода:

```
<?php
if (isset($_POST["lastname"]) || isset($_POST["firstname"]))
    if ($_POST["lastname"] != "" && $_POST["firstname"] != "") {
        $lastname = $_POST["lastname"];
        $firstname = $_POST["firstname"];
        $firstname = iconv('UTF-8', 'windows-1251', $firstname); // перевод кодировки нужен для
того, чтобы корректно русские буквы отображались
        $firstname = substr($firstname, 0, 1);
        $firstname = iconv('windows-1251', 'UTF-8', $firstname); // перевод кодировки нужен для
того, чтобы корректно русские буквы отображались
        echo $lastname . " " . $firstname . ".";
    } else
        echo "No POST data lastname or firstname";
    else
        echo "Not POST data";
?>
```

5.3. Клиент-серверная архитектура мобильных приложений

Сайт: IT Академия SAMSUNG

Курс: MDev @ IT Академия Samsung

Книга: 5.3. Клиент-серверная архитектура мобильных приложений

Напечатано:: Егор Беляев

Дата: Суббота, 18 Апрель 2020, 19:34

Оглавление

5.3.1. Общие понятия

5.3.2. Отправка запросов из Android-приложений

5.3.3. Форматы JSON и XML. Сериализация

5.3.4. Библиотека Retrofit

5.3.1. Общие понятия

Архитектура клиент-сервер предполагает наличие трех компонент:

- клиентской части (пользователь);
- серверной части (удаленный сервис);
- среды коммуникации.

В мобильных приложениях клиентская часть располагается на мобильном устройстве и, как правило, предназначена для взаимодействия с пользователем устройства. Серверная часть располагается либо на отдельном компьютере, либо на другом мобильном устройстве. Сервер предоставляет клиенту некий сервис и напрямую с пользователем не взаимодействует. Для связи между клиентской и серверной частью используется коммуникационная среда, в случае мобильных приложений, чаще всего, обеспечиваемая протоколами интернет.

Одним из популярных интернет-протоколов является Hyper Text Transmission Protocol (HTTP). Если серверная часть приложения базируется на системе обработки HTTP, то говорят о мобильном HTTP-приложении. В этом случае все коммуникации между клиентом и сервером организуются с использованием стандартных HTTP-запросов и ответов, рассмотренных в главе 5.2 (см. рис. 5.19).

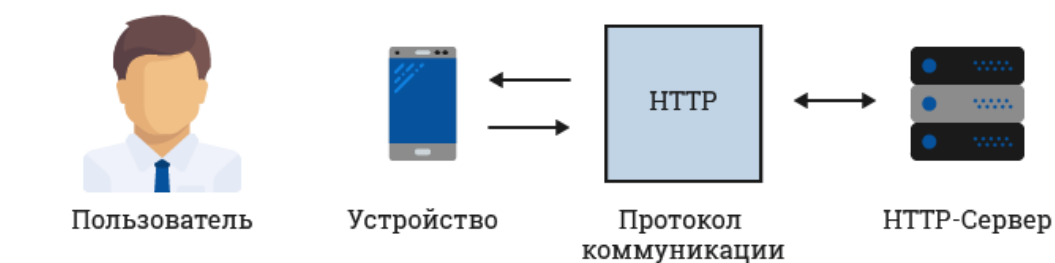


Рис. 5.19.

Запросы от мобильного клиента к серверу могут быть как синхронными, так и асинхронными. Понятия синхронных и асинхронных сообщений по своей сути сходны с понятиями синхронных и асинхронных процессов, рассмотренных в модуле 3. При использовании синхронных сообщений работа приложения блокируется до тех пор, пока не получен ответ от сервера. Если используются асинхронные сообщения, то клиент продолжает работу в штатном режиме, не дожидаясь ответа.

5.3.2. Отправка запросов из Android-приложений

Чтобы отправлять запросы через коммуникационную среду интернет, Android-приложению должен быть разрешен доступ в интернет. Для этого необходимо добавить в манифест строку:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Для написания HTTP-приложений под Android существует ряд классов, которые позволяют быстро реализовывать базовую функциональность. Они содержатся в пакете `org.apache.http` и его подпакетах. Интерфейс клиента называется `HttpClient`. Экземпляр класса с этим интерфейсом легче всего создать, используя класс `DefaultHttpClient`. В случае вызова `new DefaultHttpClient()` создается стандартный клиент.



Apache HTTP клиент был встроен в Android API до версии 22. Начиная с версии 23 (Marshmallow) эта библиотека была полностью удалена. Таким образом, для работы с HTTP нужно воспользоваться одним из следующих способов:

- при создании проекта указать API 22, или
- если вы указываете SDK выше чем 22, то можно разрешить использование отключенной библиотеки указав строку `useLibrary 'org.apache.http.legacy'` в файле `build.gradle (app)` в разделе `android`, или
- отказаться от использования указанной библиотеки и использовать вместо нее другую, например `OkHTTP`

```
HttpClient httpClient = new DefaultHttpClient();
```

Чтобы отправлять и получать данные, клиент использует объекты, имплементирующие интерфейс `HttpRequest`. Для Android разработаны классы, приспособленные для частных видов запросов. Например, для отправки данных на сервер используется `HttpPost`, для получения данных с сервера — `HttpGet`.

Объект `HttpPost` можно создать следующим образом:

```
HttpPost httpPost = new HttpPost("http://192.168.72.3:8080");
```

В конструктор передается URI-сервера (в виде объекта класса `URI` или класса `String`). Это может быть как IP-адрес, с указанием порта или без него, так и символьное доменное имя.

Существуют и другие виды HTTP-запросов, для которых разработаны специальные классы - `HttpDelete`, `HttpPut`, `HttpOptions` и т. д. Их особенности и варианты использования описаны в документации к пакету `org.apache.http.client.methods`.

Выполнение запроса осуществляется с помощью метода `execute()` интерфейса `HttpClient`. Существуют различные имплементации метода с различными входными параметрами. Если в качестве аргумента используется объект одного из классов HTTP-запросов. Метод может бросать исключения `IOException` и `ClientProtocolException`. Первое вырабатывается в случае, если возникли проблемы с соединением, второе — если вернулась ошибка HTTP-протокола.

Метод возвращает ответ сервера в виде объекта, имплементирующего интерфейс `HttpResponse`. Поэтому конструкция по отсылке запроса на сервер и получения ответа должна быть такой:

```
HttpResponse response = httpClient.execute(httppost);
```

Для извлечения полезной информации из объекта, имплементирующего `HttpResponse`, используются интерфейс `HttpEntity` и класс `EntityUtils`. Первый из них описывает блок пользовательских данных, содержащийся в HTTP-сообщении. Используя метод интерфейса `HttpResponse.getEntity()`, можно получить эту часть сообщения.

```
HttpEntity entity = response.getEntity();
```

Класс `EntityUtils` — это небольшая библиотека статических методов для обработки блоков пользовательских данных. Там есть функции преобразования данных в строки и массив байтов. Вызов метода из этого класса таков:

```
String answerHTTP =EntityUtils.toString(entity);
```

Пример 5.5

Разработаем программу для Android, которая реализует задание, аналогичное показанному в примере 5.4. При реализации асинхронного клиент-серверного приложения важно все коммуникационные операции инкапсулировать в класс, расширяющий класс `AsyncTask`, как это было показано в модуле 3. В данном упражнении достаточно переопределить методы `DoInBackground()` и `onPostExecute()`, но для решения других задач можно использовать весь арсенал средств, предоставляемых `AsyncTask`.

Итак, имплементация клиентской части требуемого приложения будет выглядеть следующим образом:

```

package com.samsung.itschool.app5_3_1;

import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
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.HttpClient;
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.apache.http.util.EntityUtils;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {
    TextView lastnameF;
    String answerHTTP;
    String lastnameS, firstnameS;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lastnameF = (TextView) findViewById(R.id.lastnameF);
    }

    public void sendPOST(View view) {
        EditText lastname = (EditText) findViewById(R.id.lastname);
        EditText firstname = (EditText) findViewById(R.id.firstname);
        lastnameS = lastname.getText().toString();
        firstnameS = firstname.getText().toString();
        new MyAsyncTask().execute("");
    }

    class MyAsyncTask extends AsyncTask<String, String, String> {

        @Override
        protected String doInBackground(String... params) {
            // Создаем HttpClient и Post Header
            HttpClient httpclient = new DefaultHttpClient();
            HttpPost httppost = new HttpPost("http://192.168.72.3:8080");
            try {
                List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
                nameValuePairs.add(new BasicNameValuePair("lastname", lastnameS));
                nameValuePairs.add(new BasicNameValuePair("firstname", firstnameS));
            }
        }
    }
}

```



```

        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs, "UTF-8"));
        // Выполняем HTTP Post запрос
        HttpResponse response = httpClient.execute(httppost);
        if (response.getStatusLine().getStatusCode() == 200) {
            HttpEntity entity = response.getEntity();
            answerHTTP = EntityUtils.toString(entity);
        }
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
    } catch (IOException e) {
        // TODO Auto-generated catch block
    }
    return null;
}

@Override
protected void onPostExecute(String result) {
    super.onPostExecute(result);
    lastnameF.setText(answerHTTP);
}
}
}

```

Пользовательский интерфейс должен включать:

- два поля текстового редактирования lastname, firstname;
- кнопку «Отправить» с обработчиком под именем sendPOST.

Для корректной работы серверная часть должна быть несколько иной, чем в примере 5.4. Класс `HttpControllerREST` изменится следующим образом:

```

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HttpControllerREST extends HttpServlet {

    @RequestMapping("/")
    public String index(HttpServletRequest request, HttpServletResponse response) {
        if (request.getParameter("lastname") != null || request.getParameter("firstname") != null)
            if (!request.getParameter("lastname").equals("") && !request.getParameter("firstname").equals("")) {
                String lastname = request.getParameter("lastname");
                String firstname = request.getParameter("firstname");
                firstname = firstname.substring(0, 1);
                return lastname + " " + firstname + ".";
            } else
                return "No POST data lastname or firstname";
        return "<h1>Greetings from Spring Boot!</h1>"
            + "<form name='test' method='post' action=''>"
            + "<p><b>Фамилия:</b><br>"
            + "<input type='text' name='lastname' size='40'></p>"
            + "<p><b>Имя:</b><br>"
            + "<input type='text' name='firstname' size='40'></p>"
            + "<p><input type='submit' value='Отправить'></p>" + " </form>";
    }
}

```

Обратите внимание, что посылаемая через интернет пользовательская информация разделена на пары «ключ — значение». Ключи в упражнении — `firstname` и `lastname`. Они одинаковы как для клиентской части приложения (хранится в `nameValuePairs`), так и для серверной части. По этим ключам происходит присвоение и извлечение соответствующих значений, которые в дальнейшем используются в программе.

Класс `Application` останется прежним:

```

import java.util.Arrays;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

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

```

*Аналогичная реализация сервера на PHP приведена ниже:

```
<?php
if (isset($_POST["lastname"]) || isset($_POST["firstname"]))
    if ($_POST["lastname"] != "" && $_POST["firstname"] != "") {
        $lastname = $_POST["lastname"];
        $firstname = $_POST["firstname"];
        $firstname = iconv('UTF-8', 'windows-1251', $firstname); // перевод кодировки нужен
        для того, чтобы корректно русские буквы отображались
        $firstname = substr($firstname, 0, 1);
        $firstname = iconv('windows-1251', 'UTF-8', $firstname); // перевод кодировки нужен
        для того, чтобы корректно русские буквы отображались
        echo $lastname . " " . $firstname . ".";
    } else
        echo "No POST data lastname or firstname";
else
    echo "
    <form name='test' method='post' action=''>
    <p><b>Фамилия:</b><br>
    <input type='text' name='lastname' size='40'></p>
    <p><b>Имя:</b><br>
    <input type='text' name='firstname' size='40'></p>
    <p><input type='submit' value='Отправить'></p>
    </form>";
?>
```

5.3.3. Форматы JSON и XML. Сериализация

Для передачи структуры данных по коммуникационным протоколам необходимо перед передачей перевести структуру в последовательность битов. Такое преобразование называется *сериализацией*. Для того чтобы принимающая сторона могла эффективно обработать эти данные, существует обратный процесс, называемый *десериализацией*. В процессе десериализации происходит восстановление структуры данных в первоначальный вид.

Распространенным видом сериализации является перевод объектов программы в файл. В этом случае объект заполняется нужными данными, потом вызывается функция сериализации, которая преобразует его в файл некоторого формата. Файл передается через коммуникационную среду. Принимающая сторона отправляет файл в функцию десериализации. После обработки получатель располагает исходным объектом, заполненным нужными данными.

Любой из схем сериализации присуще то, что кодирование данных происходит последовательно по определению, и поэтому необходимо считать весь файл и только потом воссоздать исходную структуру данных.

Формат JSON

В качестве формата файла сериализации часто используют XML. Однако возможно использование и других форматов. В частности, при разработке мобильных приложений очень популярен JSON (JavaScript Object Notation) — специальный текстовый формат обмена данными, основанный на JavaScript. Несмотря на то что синтаксис описания данных напоминает JS, формат является независимым от языка и может использоваться практически с любым языком программирования.

За счет лаконичности синтаксиса по сравнению с XML, формат JSON является более подходящим для сериализации сложных структур. Далее представлен пример сериализованного объекта «Пользователь» в обоих форматах.

JSON:

```
{
  "firstname": "Александр",
  "lastname": "Викторов",
  "from": {
    "region": "Нижегородская область",
    "town": "Дзержинск",
    "school": 17
  },
  "phone": [
    "312 123-1234",
    "312 123-4567"
  ]
}
```

XML:

```
<user>
  <firstname>Александр</firstname>
  <lastname>Викторов</lastname>
  <from>
    <region> Нижегородская область</region>
    <town>Дзержинск</town>
    <school>17</school>
  </from>
  <phone>
    <phonenumber>312 123-1234</phonenumber>
    <phonenumber>312 123-4567</phonenumber>
  </phone>
</user>
```

В JSON-формате предусмотрено использование двух структур:

- набор пар «ключ — значение». Ключом может быть только строка, значением — любая запись. В разных языках программирования ей соответствуют объект, запись, структура, словарь, хеш, именованный список или ассоциативный массив;
- упорядоченный набор значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Названные структуры данных удобны: большая часть алгоритмических языков программирования высокого уровня поддерживает их в той или иной форме. Поэтому даже если клиентская и серверная части приложения написаны на разных языках, организовать их взаимодействие не представляет труда.

В качестве значений в JSON-формате возможно использование следующих записей:

- объект — это неупорядоченное множество пар «ключ — значение», заключенное в фигурные скобки { }. Ключ и значение разделяется символом «:». Пары «ключ — значение» разделяются запятыми;
- одномерный массив — это упорядоченное множество значений. Массив заключается в квадратные скобки []. Значения ячеек массива разделяются запятыми;
- простое значение может быть строкой в двойных кавычках, числом или одним из литералов: true, false, null.

Записи могут быть вложены друг в друга. В приведенном ранее примере по ключу from находится объект, состоящий из трех пар «ключ — значение».

```
"from": {
  "region": "Нижегородская область",
  "town": "Дзержинск",
  "school": 17
}
```

Одномерный массив строк задан ключом «phone».

```
"phone": [
  "312 123-1234",
  "312 123-4567"
]
```

Строка в JSON — это строка, состоящая из символов юникода, заключенного в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\», или записаны в кодировке UTF-8. Для записи чисел используется только десятичный формат. Пробелы могут быть вставлены между любыми двумя синтаксическими элементами. Помимо описанного среди основных отличий от XML:

- JSON не поддерживает многострочные тексты — однако можно в рамках одной строки использовать escape-последовательность «\n» вместо переводов строк. Например, { «greeting» : «Поздравляю с новым годом!\nЖелаю счастья!» };
- Если текст в JSON содержит спецсимвол, то его необходимо вручную экранировать символом escape-последовательности: { «title» : «\«Rock&roll» = life» };
- В XML нет синтаксического представления массивов и списков, тогда как в JSON есть. Это, конечно, не означает, что в XML нельзя сериализовать список — конечно, можно. В этом случае программист просто имеет в виду, что группа узлов с одинаковым названием и есть элементы списка.

Сериализация с помощью класса Gson

Сериализация в JSON возможна разными способами. Рассмотрим один из простейших способов — с помощью класса Gson из пакета *com.google.gson*. Предположим, мы хотим сериализовать объект класса User.

```
public class User {  
    public String firstname; // имя  
    public String lastname; // фамилия  
    public int school; // номер школы  
}
```

Следующий код сериализует объект класса User в JSON (код для Android).

```
User student = new User();  
student.firstname = "Александр";  
student.lastname = "Викторов";  
student.school = 17;  
Gson gson = new Gson();  
Log.i("GSON", gson.toJson(student));
```

После исполнения программы в логах появится строка:

```
{"fistname":"Александр","lastname": "Викторов","school":17}
```

Чтобы производить десериализацию в Gson есть другая функция. Предположим, из JSON-строки jsonText необходимо построить объект для работы в приложении. Тогда код будет выглядеть так:

```
// строка JSON - {"fistname":"Александр","lastname": "Викторов","school":17}  
String jsonText = "{\"fistname\":\"Alexandr\",\"lastname\":\"Viktorov\",\"school\":17}";  
Gson gson = new Gson();  
User user = gson.fromJson(jsonText, User.class);  
Log.i("GSON", "Student: " + user.firstname + " " + user.lastname + " , school N " + user.school);
```

После выполнения кода в логах появится строка со значениями из структуры данных User.

Таким образом происходит сериализация/десериализация Java-объектов в JSON-строки. Примерно аналогичным образом выполняется и сериализация/десериализация Java-объектов и в XML-строки.



Однако рассмотренный способ сериализации — это лишь частный случай более общего процесса сериализации. Следует помнить, что указанные библиотеки — это надстройки над механизмом сериализации в Java. В общем случае сериализация в Java — это возможность представления любого объекта в виде простой последовательности байт («серии»). В этом случае появляется возможность передачи объектов через любые потоки. Больше можно прочесть здесь:

- <https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html>
- <http://info.javarush.ru/translation/2013/09/03/Как-работает-сериализация-в-Java.html>
- <http://www.javable.com/tutorials/fesunov/lesson17/>

5.3.4. Библиотека Retrofit

Для реализации клиент-серверных приложений, коммуницирующих по протоколу HTTP, удобно использовать специально разработанные для таких целей пакеты классов, или как их часто называют библиотеки. Одной из наиболее популярных библиотек является Retrofit. К ее достоинствам относят:

- нет нужды делать запросы к HTTP API в отдельном потоке;
- сокращается длина кода и, соответственно, ускоряется разработка;
- возможно подключение стандартных пакетов для конвертации JSON в объекты и обратно (например, пакета Gson);
- динамическое построение запросов;
- обработка ошибок;
- упрощенная передача файлов.

Логика работы библиотеки основывается на аннотациях. Благодаря их использованию можно описывать динамические запросы на сервер.

Для описания запросов к серверу необходимо объявить интерфейс, который впоследствии будет использоваться при генерации запросов. Перед каждым методом интерфейса должна стоять аннотация, основываясь на которой, Retrofit определяет, какого типа запрос обрабатывается данным методом. 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 UserService {
    @GET("/greeting/user")
    Response fetchUser();
}
```

Адрес сайта, которым отправляется запрос, не указывается. Он будет передан на этапе создания соединения клиент-сервер. Аннотация содержит лишь путь к РНР-файлу на сервере. В классе Response содержится информация о статусе запроса и ответ от сервера.

POST-запрос выглядит схоже с GET:

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

public interface UserService{
    @POST("/greeting/registration")
    Response registerUser();
}
```

Можно изменять статический маппинг URI на динамический:

```
@GET("/greeting/{firstName}/{lastName}")
Response fetchUser(@Path("firstName") String firstName, @Path("lastName") String lastName);
```


В этом случае Retrofit заменит {firstName} и {lastName} на фактические части URL, которые и будут переданы методу при вызове. Сами аргументы метода должны быть аннотированы Path, а в скобках должно заключаться обозначение конкретной части URI.

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

```
@GET("/greeting/user")
Response fetchUser(@Query("name") String name);
```

Создание соединения клиент-сервер выглядит, как серия вызовов методов библиотеки Retrofit. Например, так:

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("http://192.168.72.3")
    .addConverterFactory(GsonConverterFactory.create())
    .build();
UserService service = retrofit.create(UserService.class);
```

Через параметр метода baseUrl() передается адрес сервера. GsonConverterFactory — это класс для конвертации объектов в JSON. В нем используется уже рассмотренный нами класс Gson. UserService — интерфейс с запросами HTTP API.

После того, как соединение создано, можно его эксплуатировать, вызывая методы, привязанные к HTTP API.

```
Response resp = service.fetchUser(firstname, lastname);
```

Если ожидается ответ в виде специально созданной структуры данных, то используется конструкция Call<T>, где T — тип возвращаемой структуры данных. Например, интерфейс и структура могут выглядеть так:

```
public interface UserService {
    @GET("/greeting/user")
    Call<User> fetchUser();
}
public class User {
    public String firstName;
    public String lastName;
    public String fullName;
}
```

Call — это класс, который знает, что такое десериализация и может работать с HTTP-запросами. Один экземпляр этого класса — это одна пара «запрос — ответ». Экземпляр может быть использован лишь единожды. Чтобы использовать ту же пару повторно, необходимо использовать метод clone().

Вот как выглядит запрос, в ответ на который ожидается получить специальную структуру данных.

```

    Call<User> call = service.fetchUser(firstname, lastname);
    try {
        Response<User> userResponse = call.execute();
        userFromServer = userResponse.body();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Как видно, сначала создается вызов запроса, и лишь потом он выполняется методом `execute()`. Это дает возможность передавать созданный запрос в другие классы и выполнять его в любом месте программы.

Если попытаться выполнить вызов запроса дважды, то будет брошено исключение `IllegalStateException`.

```

    userResponse = call.execute();

    // это вызовет IllegalStateException:
    userResponse = call.execute();
    Call<User> call2 = call.clone();

    // А это нет:
    userResponse = call2.execute();

```

Если вызов клонировать и выполнить уже объект-клон, то программа работает без ошибок.

Пример 5.6

Модифицируем программу из примера 5.5 так, чтобы она использовала JSON и библиотеку Retrofit. Полученный код клиентской части приведен ниже. Также можно скачать подобный пример (клиент и сервер).

Импорт используемых классов:

```

import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import com.samsung.itschool.app5_3_2.R;
import com.samsung.itschool.app5_3_2.api.UserService;
import com.samsung.itschool.app5_3_2.model.User;
import java.io.IOException;
import retrofit.Call;
import retrofit.GsonConverterFactory;
import retrofit.Response;
import retrofit.Retrofit;

```

Базовая активность приложения:

```

public class MainActivity extends AppCompatActivity {
    private static String LOG_TAG = "MainActivity";
    private final String baseUrl = "http://192.168.72.3:8080";
    private TextView lastnameF;
    private String answerHTTP;
    private String lastnameS, firstnameS;
    private User userFromServer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lastnameF = (TextView) findViewById(R.id.lastnameF);
    }

    public void sendPOST(View view) {
        EditText lastname = (EditText) findViewById(R.id.lastname);
        EditText firstname = (EditText) findViewById(R.id.firstname);
        lastnameS = lastname.getText().toString();
        firstnameS = firstname.getText().toString();
        new MyAsyncTask().execute("");
    }
}

```

Асинхронный поток, осуществляющий клиент-серверный обмен информацией:

```

class MyAsyncTask extends AsyncTask<String, String, String> {

    @Override
    protected String doInBackground(String... params) {
        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(baseUrl)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
        UserService service = retrofit.create(UserService.class);
        Call<User> call = service.fetchUser(firstnameS, lastnameS);
        try {
            Response<User> userResponse = call.execute();
            userFromServer = userResponse.body();
            Log.d(LOG_TAG, userFromServer.fullName);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);
        lastnameF.setText(userFromServer.fullName);
    }
}

```

Интерфейс клиент-серверного взаимодействия UserService:

```

import com.samsung.itschool.app5_3_2.model.User;
import retrofit.Call;
import retrofit.http.GET;
import retrofit.http.Path;

public interface UserService {
    @GET("/greeting/{firstName}/{lastName}")
    Call<User> fetchUser(@Path("firstName") String firstName, @Path("lastName") String lastName);
}

```

Ниже приведен класс передаваемой структуры данных User. Обратите внимание, что для таких сериализуемых объектов правилом хорошего тона является get-теров и set-теров, также должен быть конструктор по умолчанию.

```

public class User {
    public String firstName;
    public String lastName;
    public String fullName;
    public String getFirstName() { return firstName; }
    public String getLastName() { return lastName; }
    public String getFullName() { return fullName; }
    public void setFirstName(String firstName) { this.firstName=firstName; }
    public void setLastName(String lastName) { this.lastName=lastName; }
    public void setFullName(String fullName) { this.fullName=fullName; }
}

```

Рассмотрим серверную часть программы.

На сервере получаем путь запроса с параметрами GET и возвращаем сериализованный объект User:

```

package ru.itschool.controller;
import ru.itschool.model.User;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.*;

@RestController
@EnableAutoConfiguration
public class UserController {
    @RequestMapping(path="/greeting/{firstName}/{lastName}",RequestMethod.GET)
    public User greeting(@PathVariable("firstName") String firstName, @PathVariable("lastName") String lastName) {
        return new User(firstName, lastName);
    }
}

```

Класс User на Java-сервере:

```

package ru.itschool.model;

public class User {
    public String firstName;
    public String lastName;

    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFullName() {
        return firstName + " " + lastName;
    }
}

```

Аналогичная функциональность на PHP-сервере будет выглядеть следующим образом:

```

<?php

class User
{
    public $lastname = "";
    public $firstname = "";

    function __construct($firstname, $lastname)
    {
        $this->firstname = $this->getShortFirstname($firstname);
        $this->lastname = $lastname;
    }

    public function getShortFirstname($firstname)
    {
        $firstname = iconv('UTF-8', 'windows-1251', $firstname); // для корректного отображения
русских букв
        $firstname = substr($firstname, 0, 1);
        $firstname = iconv('windows-1251', 'UTF-8', $firstname); // для корректного отображения
русских букв
        return $firstname;
    }
}

if (isset($_GET["lastname"]) || isset($_GET["firstname"]))
    if ($_GET["lastname"] != "" && $_GET["firstname"] != "") {
        $user = new User($_GET["firstname"], $_GET["lastname"]);
        echo json_encode($user);
    } else
        echo "No GET data lastname or firstname";
else
    echo "No data";
?>

```


5.4. Облачные платформы. REST-взаимодействие

Сайт: IT Академия SAMSUNG
Курс: MDev @ IT Академия Samsung
Книга: 5.4. Облачные платформы. REST-взаимодействие
Напечатано:: Егор Беляев
Дата: Суббота, 18 Апрель 2020, 19:34

Оглавление

5.4.1. Облачные технологии

5.4.2. Модели развертывания

5.4.3. Модели обслуживания

5.4.4. Платформа как услуга

5.4.5. REST-взаимодействие

5.4.6. REST-аутентификация и OAuth-авторизация

5.4.1. Облачные технологии

Облачные технологии — сегодня это уже не только информационная технология, но и привычная концепция повседневной жизни современного человека. По сути, это перенос в публичную сеть всех видов ресурсов пользователя — от файлов до приложений и сервисов. Доступ к ним может осуществляться через интернет как по отдельности, так и с возможностью их взаимодействия.

Еще совсем недавно пользователь, чтобы просмотреть офисный документ, пересланный по почте, должен был открыть приложение электронной почты, получить письмо, скачать файл из вложения, открыть его при помощи офисного приложения. При использовании облачных технологий пользователь может открыть браузер, зайти на веб-почту, открыть письмо и тут же в браузере просмотреть файл, используя офисное веб-приложение.

Таким образом, можно видеть, что в облачные технологии предполагают перенос файлов (данных) и приложений (вычислений) с компьютера пользователя на интернет-облачные ресурсы.

Достоинства облачных технологий:

- доступность: чтобы получить доступ к своим данным, пользователю достаточно иметь лишь доступ в интернет;
- мобильность: пользователь не привязан к своему рабочему месту, он может работать из любой точки мира, с любого устройства;
- экономичность: пользователю нет необходимости закупать программное обеспечение либо компьютерное оборудование, достаточно иметь любое устройство, вплоть до мобильного, с браузером. При этом он не несет нагрузки по обслуживанию оборудования, ПО и лицензий;
- арендность: пользователь платит за продукт только в тот момент, когда он ему нужен, и только в том объеме, в котором хочет;
- гибкость: все необходимые ресурсы предоставляются провайдером автоматически;
- высокая технологичность: инфраструктура, на которой работает облако, поддерживается оператором в самом актуальном состоянии. Это означает, что ПО будет самых последних версий, оборудование использует самые мощные и современные решения;
- надежность, которую обеспечивают облачные платформы, обычно гораздо выше, чем у оборудования пользователя, ведь далеко не всякий пользователь может позволить приобрести отказоустойчивый ЦОД.

Недостатки:

- необходимость постоянного соединения к интернет;
- ограничения ПО. ПО, которое разворачивается в облаке, может иметь ограничения, не позволяющие использовать некоторые функции, доступные в локальных версиях аналогичного ПО. Однако следует помнить, что в облачном ПО в то же самое время могут быть и функции, превосходящие возможности локального ПО;
- конфиденциальность: конфиденциальность данных, хранимых в публичных «облаках», часто вызывает беспокойство у пользователей. Однако грамотный подход к вопросам шифрования данных и настроек доступа могут минимизировать эти риски;
- безопасность: облачная платформа сама по себе достаточно безопасна, однако в случае проникновения злоумышленника в сервисы пользователя, он получает доступ сразу к очень большому объему данных. Такое проникновение возможно, например, из-за

использования слабых паролей или пользования небезопасной сетью;

- дороговизна оборудования: создание собственного облака требует довольно крупных материальных вложений, в связи с чем это не могут себе позволить небольшие компании или стартапы;
- дальнейшая монетизация ресурса. Вполне возможно, что компании-операторы в дальнейшем решат брать плату с пользователей за предоставляемые услуги, которые в данный момент доступны бесплатно.

5.4.2. Модели развертывания

Частное облако

Частное облако (англ. *private cloud*) — это облачная ИТ-инфраструктура, созданная частной организацией. Обычно доступ в это облако имеют сотрудники организации и иногда ее клиенты. Таким образом, эта инфраструктура предназначена для пользования закрытыми корпоративными сообществами и не содержащая публичных сервисов.



Общественное облако

Общественное облако (англ. *community cloud*) — это вид инфраструктуры, предназначенный для использования сообществом пользователей, вне зависимости от принадлежности пользователя к какой-либо компании. Общественное облако обычно находится в собственности коммерческой организации, и она несет затраты по его обслуживанию.

Гибридное облако

Гибридное облако (англ. *hybrid cloud*) — это комбинация из двух или более различных облачных инфраструктур (частных, публичных или общественных), остающихся независимыми информационными структурами, но связанных технологиями передачи данных и приложений.

5.4.3. Модели обслуживания

Модель обслуживания определяет, какой вид услуг предоставляет облако.

Программное обеспечение как услуга

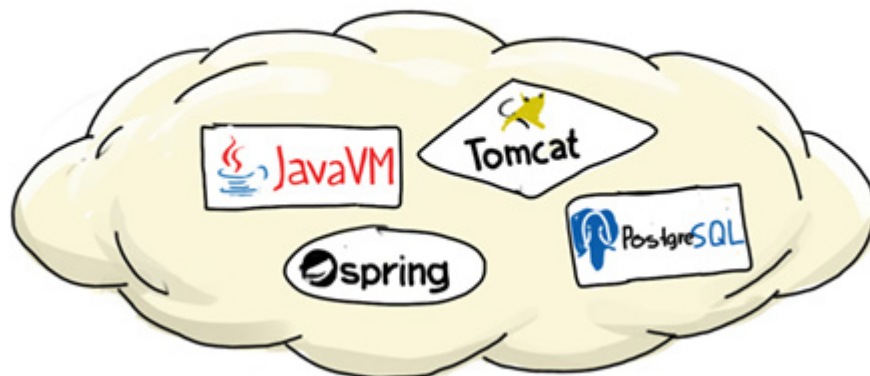
SaaS



Программное обеспечение как услуга (SaaS, англ. Software-as-a-Service) — модель, в которой услугой, предоставляемой пользователю, является доступ к прикладному программному продукту, работающему в облаке. Доступ к предоставляемой программе может осуществляться из браузера. Пример SaaS — веб-почта, Google Docs, Google Calendar и подобные онлайн-программы. Администрирование и управление физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, систем хранения, осуществляется облачным провайдером. Доступ пользователю может предоставляться как на платной, так и на бесплатной основе.

Платформа как услуга

PaaS



Платформа как услуга (PaaS, англ. Platform-as-a-Service) — модель, когда пользователю предоставляется возможность запускать свое программное обеспечение на работающем в облаке ПО среднего уровня (MiddleWare) и прочего серверного/служебного ПО и фреймворков. Обычно в состав ПО платформы входят:

- сервера СУБД, например, PostgreSQL, MySQL, db2, Oracle DB, MsSql, nosql MongoDB и т. д.;
- сервера приложений/контейнеры, например, Tomcat, Jboss и т. д.;

- среды исполнения, например, Java JVM, GO vm и т. д.;
- фреймворки/библиотеки, например, JDBC, Gson, Spring и т. д.

На этой платформе пользователь может устанавливать как разработанные им приложения, так и существующие ранее. Развертывание, администрирование и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, систем хранения, а также промежуточного ПО осуществляется облачным провайдером. Иными словами, PAAS — это облачная платформа для разработки, тестирования, развертывания и поддержки собственного ПО пользователя. К 2020 году мировой рынок PaaS предположительно оценивается в \$235 млрд. Десятка крупнейших провайдеров:

- 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.

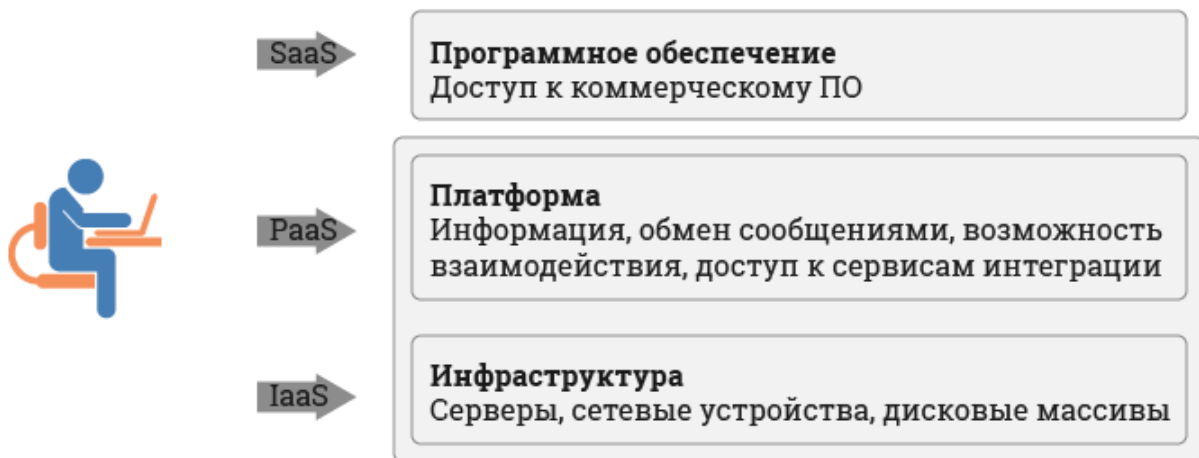
Инфраструктура как услуга

IaaS



Инфраструктура как услуга (IaaS, англ. Infrastructure-as-a-Service) предоставляет потребителю виртуальный компьютер (именно как аппаратную часть), с сетью и выходом в интернет. Используя эти облачные аппаратные ресурсы, пользователь может сам установить операционную систему, и любое ПО, которое ему необходимо для его деятельности. Таким образом, предоставляемой услугой фактически является аренда в облаке аппаратной IT-инфраструктуры. Потребитель услуги может устанавливать и запускать произвольное программное обеспечение, которое может включать в себя операционные системы, платформенное и прикладное программное обеспечение. Администрирование 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 веб-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 веб-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 веб-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: WindowsTM, Apple Mac OS X и Linux[®] для операционных систем; Google Android, Windows Mobile[®] и Apple iOS для мобильных вычислений; Adobe[®] AIRTM 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.



Рис. 5.20.

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:

1. Какие инфраструктуры и языки поддерживаются?
2. Сколько приложений можно создать?
3. 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.

4. Базы данных какого типа поддерживаются?
5. 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.

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.

OAuth PUT
JSON GET POST XML
REST
Basic-Auth
DELETE

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. То есть 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 (рис. 5.21).

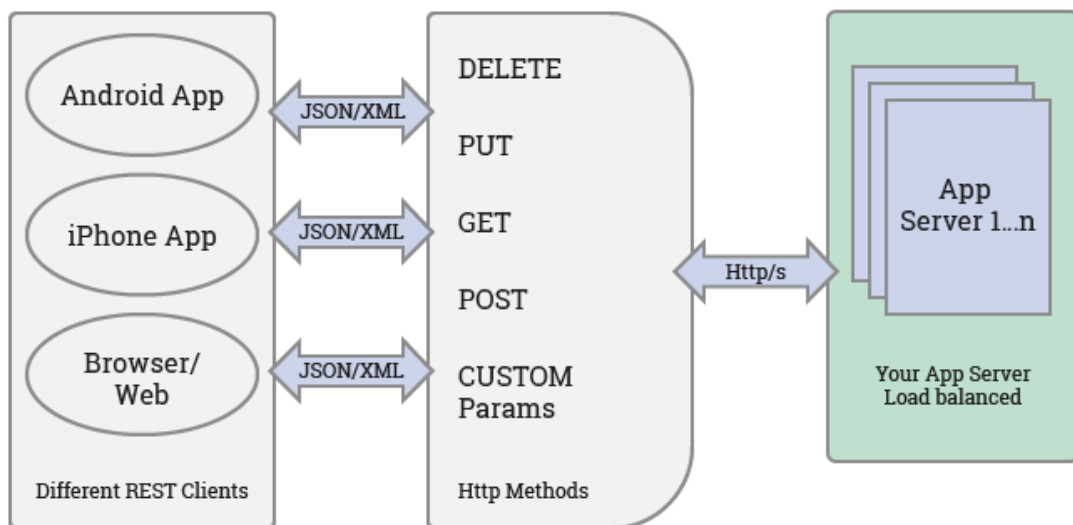


Рис. 5.21.

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. То есть, 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 четырьмя 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 остается 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 некаэшируемые для предотвращения повторного использования клиентами в последующих запросах сохраненной 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 отправлять 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 медиатипу).
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, то есть преобразовать Java-объект в JSON;
- выполнить запрос, воспользовавшись HTTP-клиентом;
- распарсить результаты запроса, то есть 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 аннотацией Path и в скобках нужно указать строку с ключевым словом.

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);
```

Пример 5.7

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
```

Пример запроса 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"]
```

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

Табл. 5.3. Коды ошибок

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
```

Пример запроса JSON-интерфейс:

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

Параметр	Тип	Описание
Обязательные		
key	string	API-ключ
lang	string	Язык текста (например, «en»). Список языков можно получить с помощью метода <code>getLangs</code>

Параметр	Тип	Описание
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-четырёх слов
Необязательные		
limit	int	Максимальное количество возвращаемых строк (по умолчанию 1)

Табл. 5.4. Список входных параметров.

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 (см. табл. 5.5).

Элемент	Описание
CompleteResponse	Корневой элемент, содержит элемент text. Атрибуты: endOfWord — признак конца слова (true/false). Метод может одновременно вернуть продолжение запроса и признак endOfWord = true (например, для текста «здравствуй» возвращается продолжение «здравствуйте», а также признак конца слова). Pos — позиция в слове, для которого возвращается продолжение. Позиция отсчитывается от последнего символа в запросе, переданном в элементе q. 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»)
text	Содержит элементы string с наиболее вероятными вариантами продолжения заданного текста. Если метод не может «продолжить» текст, то элемент не возвращается

Табл. 5.5.

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

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

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)

Код	Значение	Описание
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

Табл. 5.6. Коды ошибок

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



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>



Для подключения библиотеки Retrofit2 необходимо добавить следующие строки в файл *build.gradle* проекта:

```
dependencies {
    compile 'com.squareup.retrofit2:retrofit:2.0.0-beta4'
    compile 'com.squareup.retrofit2:converter-gson:2.0.0-beta4'
}
```

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.cb27d8058bc81d29.10b405aa732895274b7d14c9f7a55a116a832d93";

    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();
}
}

```

Создадим класс для объекта, который соответствует результату, полученному от Яндекса:

```

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

```

Interface, описывающий запрос:

```

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. В нем можно увидеть, какие слова предиктор предлагает в ответ на введенные начала слов (см. рис. 5.22).

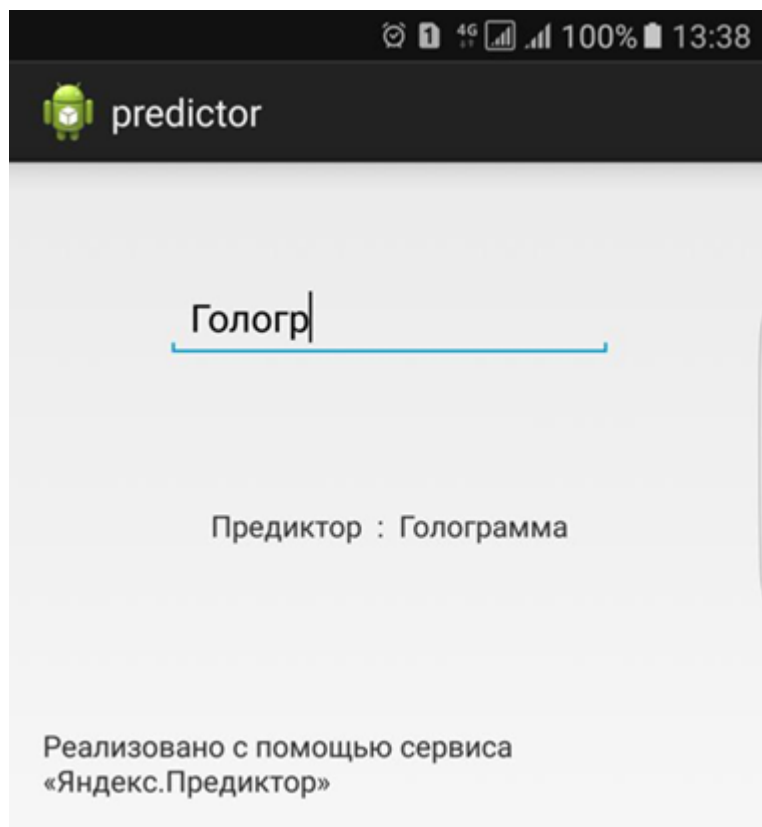


Рис. 5.22.

5.4.6. REST-аутентификация и OAuth-авторизация

Рассмотрим различные способы REST-аутентификации.

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 — 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, может содержать expiration time (время хранения объекта) и роли пользователя, а также 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 — 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.

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

Аутентификация: ученик указывает свой логин и пароль, система находит пользователя с таким логином, сверяет хеш введенного пароля с хранимым в системе. Если они совпадают, то система считает, что пользователь подлинный.

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

Новый протокол OAuth 2.0 стал популярным в последнее время. Он позволяет предоставлять доступ к защищенному ресурсу пользователю, который выполнил авторизацию в какой-либо крупной социальной сети. Таким образом сам ресурс избавлен от обязанности аутентификации, то есть не хранит информацию о логине и пароле пользователя.

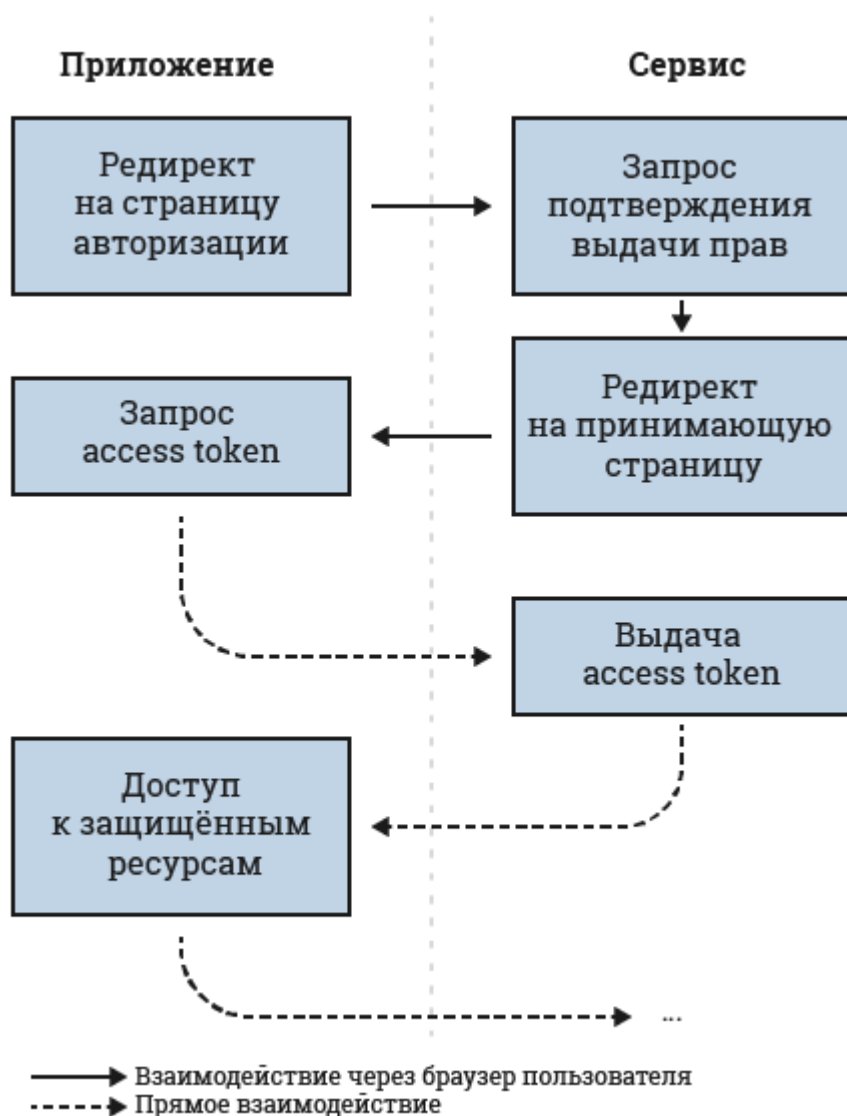


Рис. 5.23.

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 стандартная для регистрации информация 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.

Пример 5.8*

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

1. Приложение реализует OAuth-авторизацию через социальную сеть ВКонтакте.
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, *RНР.

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 примера 5.6. 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, необходимо использовать 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 {
    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 (см. табл. 5.7).

Параметр	Описание
Client_id	Обязательно Идентификатор вашего приложения

Параметр	Описание
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: 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. Укажите code, чтобы осуществлять запросы со стороннего сервера
v	Версия используемого API. Актуальная версия: 5.50
state	Произвольная строка, которая будет возвращена вместе с результатом авторизации

Табл. 5.7.

Получим необходимые регистрационные данные приложения на сайте <http://vk.com/editapp?act=create>. Заполняем поля формы (см. рис. 5.24).

Создание приложения

Название:

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

Адрес сайта:

Базовый домен:

Рис. 5.24.

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 (рис. 5.25).



Если будет стоять задача создать другое приложение с авторизацией ВКонтакте, то для него необходимо получить собственные регистрационные данные, аналогично тому, как было показано для данного упражнения. Публикация ID и ключа приложения в открытом доступе запрещена.

OAuth 5.4

Информация

Настройки

Хранимые процедуры

Статистика

Руководство

Помощь

ID приложения: **5400655**

Защищенный ключ:

Состояние: **Приложение включено и видно всем**

Первый запрос к API:

Установка приложения: **Не требуется**

Open API: **Включён**

Open API:

Адрес сайта:

Тематика сайта: **Мобильная связь и интернет**

Базовый домен:

Доверенный redirect URI:

[Добавить ещё](#)

Сохранить изменения

Рис. 5.25.

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. Следует обратить внимание, что если сервер запускается на собственном компьютере, то в переменной `OUR_SERVER` необходимо указать его 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 = "wwtRxNOBD6yfkclsSE";
    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();
    pDialog = new ProgressDialog(MainActivity.this);
    pDialog.setMessage("Connecting VK ...");
    pDialog.setIndeterminate(false);
    pDialog.setCancelable(true);
    Code = pref.getString("Code", "");
    pDialog.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) {
        pDialog.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();
            pDialog.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>

```

strngs.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 jsonObj = 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 {
        jsonObj = new JSONObject(jsonT);
    } catch (JSONException e) {
        Log.e("JSON Parser", "Error parsing data " + e.toString());
    }
    // Return JSON String
    return jsonObj;
}
}

```

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 {
        8);
        BufferedReader reader = new BufferedReader(new InputStreamReader(is, "utf-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 уже обсуждалось, разрабатываемый сервер, получив JSON-сообщение, возвращает искомые 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;
    }
}

```

```
}  
}
```

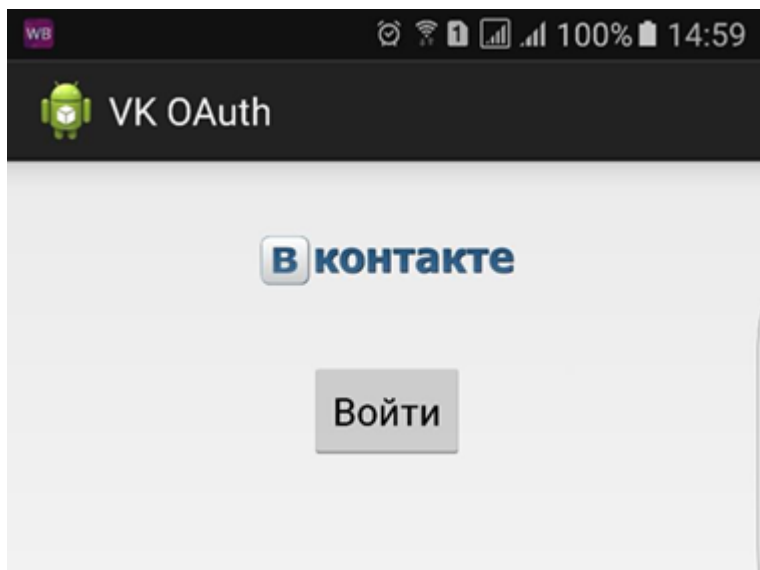


Рис. 5.26 Стартовое окно приложения.

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 уже выполнен, 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 ИТ ШКОЛЫ SAMSUNG

<https://my.webinar.ru/record/621951/> [<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/> (пример 5.7) и создадим новое приложение (рис. 5.27).

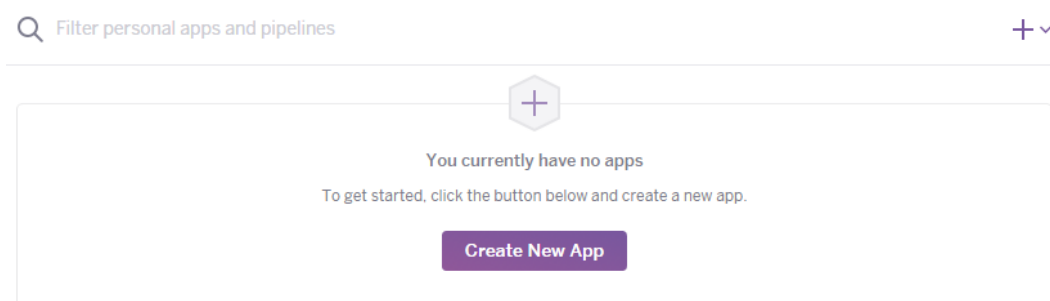


Рис. 5.27.

Для примера назовем создаваемый проект «server-54» (см. рис. 5.28). 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

Рис. 5.28.

В качестве способа развертывания сервера (Deployment method) выберем Heroku Git (см. рис. 5.29).

Deployment method

Heroku Git
Use Heroku Toolbelt

GitHub
Connect to GitHub

Dropbox
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
```

Рис. 5.29.

- 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
```

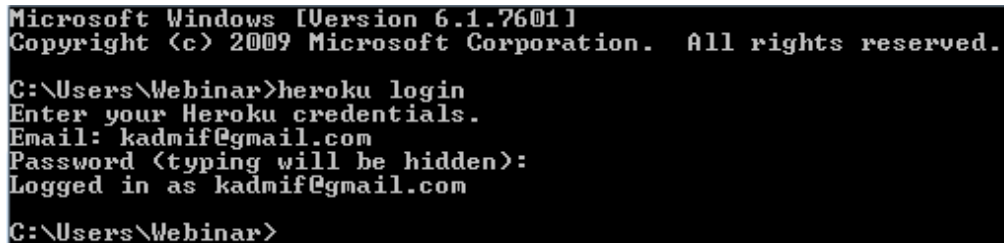
Данный файл будет указывать компилятору Heroku, что необходимо создать веб-проект.

- Также необходимо 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. Далее нужно разместить проект, запустив командную строку (терминал), и выполнить все описанные в подсказке Heroku команды.

```
heroku login
```



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

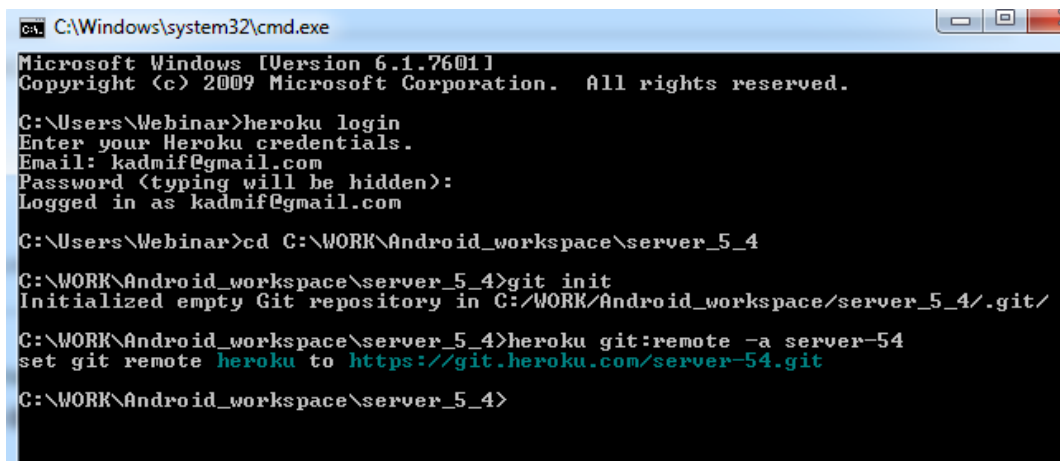
C:\Users\Webinar>heroku login
Enter your Heroku credentials.
Email: kadmif@gmail.com
Password (typing will be hidden):
Logged in as kadmif@gmail.com

C:\Users\Webinar>
```

Рис. 5.30.

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 нужно перейти в папку с проектом, инициализировать его для git и указать, 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>
```

Рис. 5.31.

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

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

В папке проекта появится скрытая папка.git. И последняя команда закинет файлы на сервер, где он сам их соберет в проект: `git push heroku master`.

5. Далее нужно указать адрес сервера в клиентском приложении. В личном кабинете на HEROKU, в панели Dashboard выберите свой проект, и в меню Settings в разделе Domains будет показан адрес сервера, который необходимо указать в переменной OUR_SERVER класса MainActivity. В рассматриваемом случае измененная строка будет выглядеть так:

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

Проект полностью готов.

5.5. Серверные СУБД

Сайт: IT Академия SAMSUNG
Курс: MDev @ IT Академия Samsung
Книга: 5.5. Серверные СУБД
Напечатано: Егор Беляев
Дата: Суббота, 18 Апрель 2020, 19:34

Оглавление

- 5.5.1. Клиент-серверные архитектуры. Серверные СУБД
- 5.5.2. Настройка PostgreSQL и подключение к БД
- 5.5.3. Реализация back end части приложения на языке Java
- 5.5.4. Реализация back end части приложения на языке PHP

5.5.1. Клиент-серверные архитектуры. Серверные СУБД

Клиент-серверная архитектура частично обсуждалась в главе 5.3. Кроме того, была рассмотрена локальная система управления базами данных SQLite. Также было изучено и рассмотрено на примерах, как Android-приложение работает в качестве клиента для веб-сервиса.

Преимущества использования баз данных абсолютно понятно — это централизованность (данные хранятся в одном месте) и структурированность (данные логически связаны между собой, образуя определенную структуру, которая позволяет хранить информацию о некоторой предметной области). Рассмотрим различные варианты архитектур построения приложений (рис. 5.32).

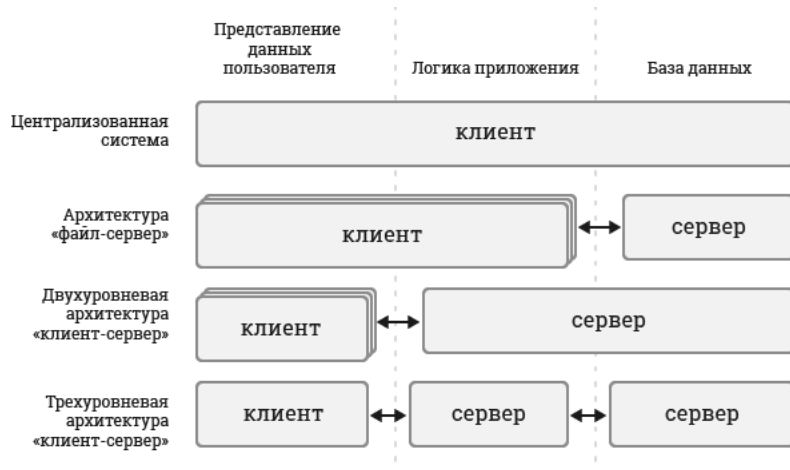


Рис. 5.32.

Приложения с централизованной архитектурой уже рассматривались — это приложения со встроенным SQLite. В чем недостатки таких систем? Представим Android-приложение, работающее с базой данных, в которой заранее вложена необходимая информация. Это значит, что, когда потребуется изменить данные в этой базе, ее возможно изменить, только обновив все приложение. К тому же, вся логика работы с БД, а следовательно, и вычислительная нагрузка, ложатся на смартфон пользователя. Конечно, это не очень хорошая практика — заставлять пользователя постоянно скачивать такие мало полезные обновления и нагружать его телефон.

Файл-серверная архитектура не получила распространения среди мобильных приложений по причине того, что работа с сервером баз данных требует довольно мощный и надежный канал связи. Обычно связь на мобильных устройствах ни первому, ни второму критерию не соответствует.

Двухуровневая клиент-серверная архитектура предполагает использование клиент-серверной СУБД. Такая СУБД использует технологию «клиент-сервер», размещается на сервере и позволяет обмениваться клиенту и серверу минимально необходимыми объемами информации. При этом основная вычислительная нагрузка ложится на сервер. Клиент может выполнять функции предварительной обработки перед передачей информации серверу, но в основном его функции заключаются в организации доступа пользователя к серверу. Сервер баз данных дает возможность отказаться от пересылки по сети файлов данных и передавать только ту выборку из базы данных, которая удовлетворяет запросу пользователя. Логика, необходимая для работы с базой данных, остается все еще на клиентских устройствах пользователей, что в случае смартфона является большим недостатком, кроме того, велика вероятность возникновения ошибок и других проблем. Здесь же стоит упомянуть, что, если необходимо клиентские приложения под различные платформы, то придется продублировать много одинакового по своей логике кода.

Трехуровневая (трехзвенная) архитектура решает эту проблему путем переноса всей работы с СУБД в отдельный слой. В этом случае клиентское приложение будет являться лишь средством ввода и отображения информации, а вся «Бизнес-логика» системы будут располагаться на сервере.



Рис. 5.33.

Рисунок 5.33 в полной мере описывает работу трехзвенной архитектуры приложения, для каждого слоя слева располагается поясняющая информация о каждом уровне. С развитием сетей и интернета данная трехуровневая архитектура построения приложений приобретала все более популярный характер, что повлекло за собой появление различных новых программных комплексов, качественно реализующих ее. Однако в сложных

приложениях используют более масштабируемую многоуровневую архитектуру.

На рынке серверного ПО существует достаточно много СУБД. Наиболее распространенными являются MySQL, PostgreSQL, Oracle, SQL Server.

На предыдущих занятиях были представлены ознакомительные примеры на облачной платформе Heroku. Она как раз является одним из многих решений для размещения серверных частей приложений. Это достаточно мощная платформа, которая поддерживает около семи языков программирования, сервера приложений (в примерах книги использовался веб-сервер Tomcat) и может работать с пятью СУБД, где основной является СУБД PostgreSQL. Данная платформа выбрана для создания учебных приложений, потому что поддерживает необходимое ПО и дает возможность бесплатного использования.

5.5.2. Настройка PostgreSQL и подключение к БД

Как уже было сказано в конце предыдущей главы, в качестве серверной платформы выбрана <https://www.heroku.com/>. В контексте данной главы одной из задач является создание базы данных в СУБД PostgreSQL Heroku и подключение к ней для последующей работы. Для начала потребуется:

1. Успешно зарегистрированный аккаунт на <https://www.heroku.com/>.
2. Успешно установленная СУБД <http://www.postgresql.org/download/>.

Установка в Heroku СУБД PostgreSQL

Если все вышеперечисленное имеется, можно переходить к созданию базы данных. К слову сказать, этот процесс будет несложным, потому что для этого необходимо всего четыре клика мышкой. Следующие четыре рисунка демонстрируют этот процесс.

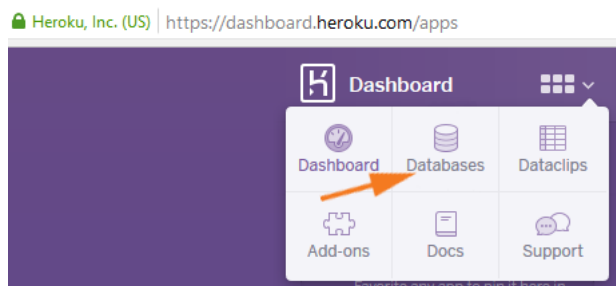


Рис. 5.34. Переход в раздел Databases (базы данных).

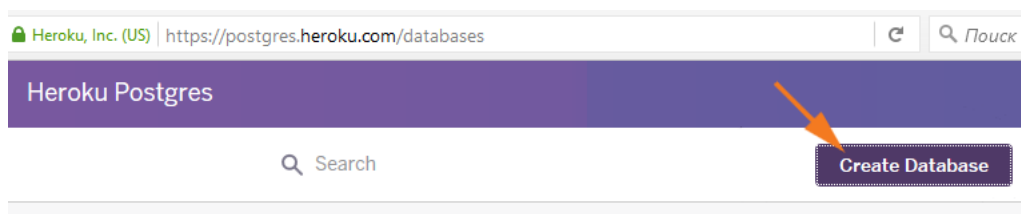


Рис. 5.35. Нажатие на кнопку «Create Database» (создать базу данных).

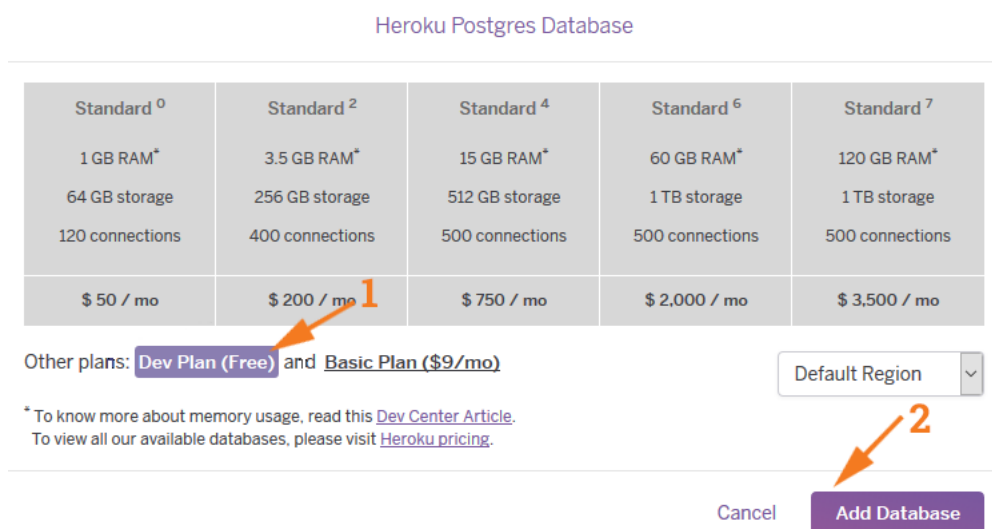


Рис. 5.36. Выбор Dev Plan (Free) (бесплатного плана разработчика).

После проделанных выше действий необходимо подождать некоторое время, приблизительно 30–60 секунд для того, чтобы СУБД успешно установилось. Подтверждением этого будет являться ее статус Available (доступно), как показано на рисунке 5.37.



Рис. 5.37. Завершение создания БД, статус Available (доступно).

Установка интерфейса к СУБД pgAdmin

Итак, база данных была создана успешно. Однако, к большому сожалению, на платформе Heroku нет полноценного веб-интерфейса для управления созданной БД. Поэтому приходится пользоваться дополнительным инструментом, программой pgAdmin, свободно распространяемой программой для администрирования СУБД PostgreSQL, которую необходимо установить на компьютер (скачать здесь: <http://www.pgadmin.org/download/>). Запустим pgAdmin и создадим подключение к нашему серверу на Heroku. Для этого в запущенном pgAdmin выполним следующие действия: выберем пункт в главном меню «Файл», а затем «Добавить сервер», после чего откроется окно «Новая регистрация сервера», изображенное на рисунке 5.38.

Рис. 5.38. Регистрация нового сервера.

Теперь необходимо заполнить недостающие параметры подключения. Однако откуда взять данные для заполнения? Эта информация доступна на сервисе heroku, если перейти в подробное описание базы данных, кликнув на ее название.

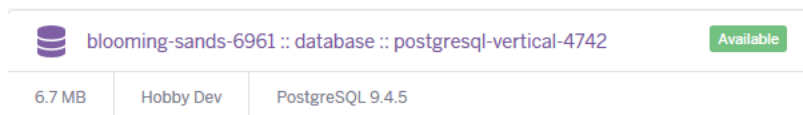


Рис. 5.39. Клик по названию базы данных

Перенесем необходимые параметры из раздела Connection Settings (настройки соединения) в окно pgAdmin «Новая регистрация сервера» следующим образом:

- имя задаем произвольное, например, Heroku;
- Host — Хост;
- Database — Обслуживание DB;
- User — Имя пользователя;
- Port — Порт;
- Password — Пароль.

Пустым остается лишь пункт в pgAdmin «Новая регистрация сервера» — «Служба». Результат вышеописанных действий изображен на рисунке 5.40.

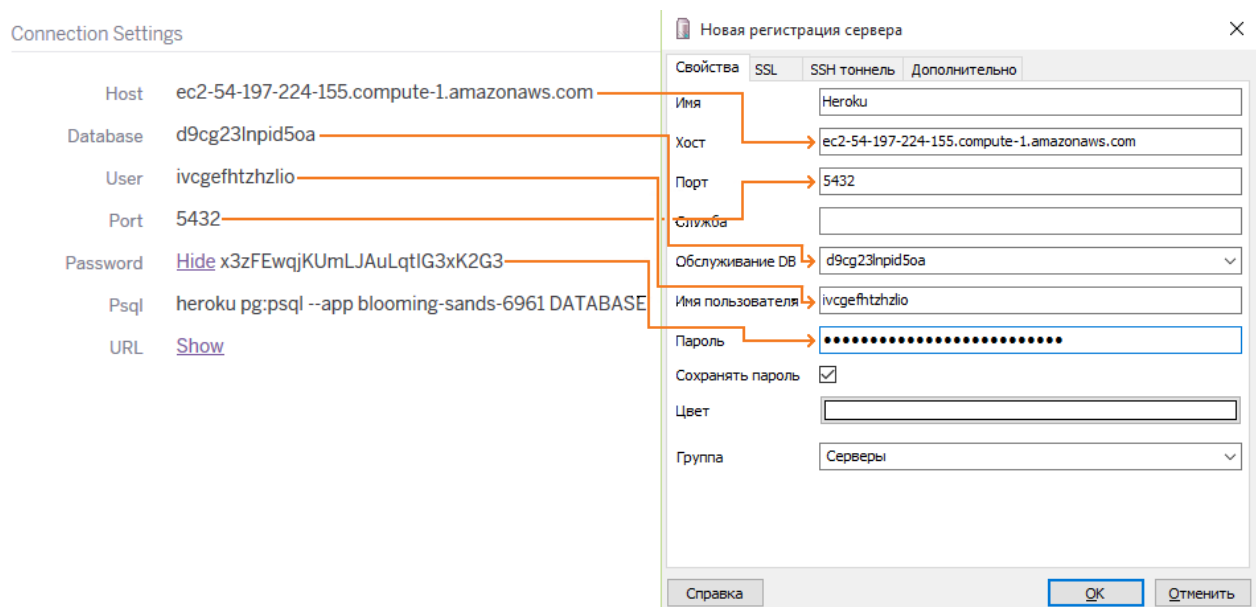


Рис. 5.40. Перенос настроек, новая регистрация сервера.

После того как все настройки перенесены, нужно всего лишь нажать кнопку «Ок», и Heroku появится в списке серверов, расположенном в браузере объектов (см. рис. 5.41).

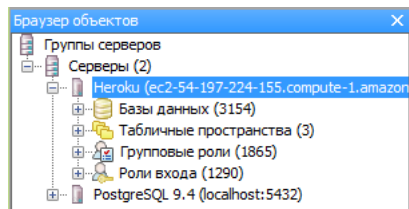


Рис. 5.41. PgAdmin браузер объектов, отображается сервер Heroku

В pgAdmin в разделе «Базы данных» отображается очень много объектов. Откуда они? На самом деле к создаваемому проекту принадлежит, конечно, только одна база данных, которая была создана в процессе этой лекции, а остальные — это БД Heroku других проектов.

Найдем нужную БД по названию, которое было указано на предыдущем рисунке под пунктом 2. На этом настройка PostgreSQL и подключение к БД закончены. Следующим шагом нужно создать таблицы в этой базе данных, после чего можно создавать серверную программную часть для работы с этой БД.

Создание таблиц в БД

В качестве предметной области для создания таблиц в базе данных рассмотрим телефонный справочник. Это интересный и простой для понимания пример работы с БД.

В самом простом случае, который взят за основу, будет две таблицы: люди (которым принадлежит какой-то номер) и номер телефона. Таблицы будут соединены связью один ко многим (1:M), так как одному человеку может принадлежать несколько номеров, но в свою очередь за одним номером закрепляется только один человек. На рисунке 5.42 представлена модель данных.

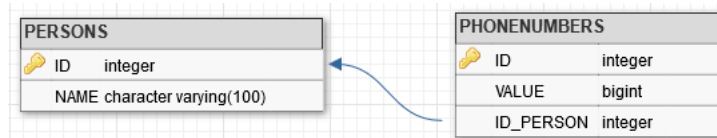


Рис. 5.42. Модель данных телефонного справочника

Для того чтобы создать эти таблицы, необходимо в pgAdmin в браузере объектов найти свою базу данных и нажать слева от ее названия на значок плюса, затем таким же способом раскрыть объекты «Схема» и «public», в результате чего будут показаны много возможностей, которые можно использовать с выбранной схемой. Выберем пункт «Таблицы» и возможность создания новой, для этого правой кнопкой мыши вызываем контекстное меню пункта «Таблицы», затем «Новый объект» — «Новая таблица» (рис. 5.43).

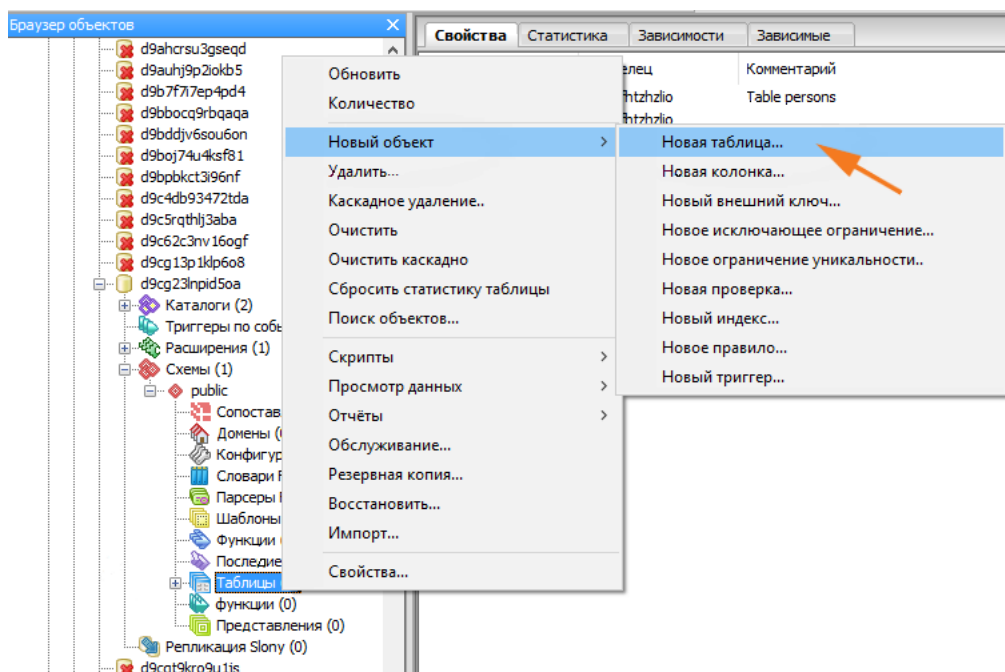


Рис. 5.43. Выбор пункта «Новая таблица»

В результате выполненных действий откроется окно «Новая таблица». Чтобы было видно все верхние вкладки, данное окно необходимо расширить, либо переключать их специальными стрелочками, расположенными в правом верхнем углу. Заполним важные поля на примере создания таблицы PERSONS. На рисунке 5.44 отображена вкладка «Свойства» и написано имя таблицы. Также выделены важные для настройки пункты.

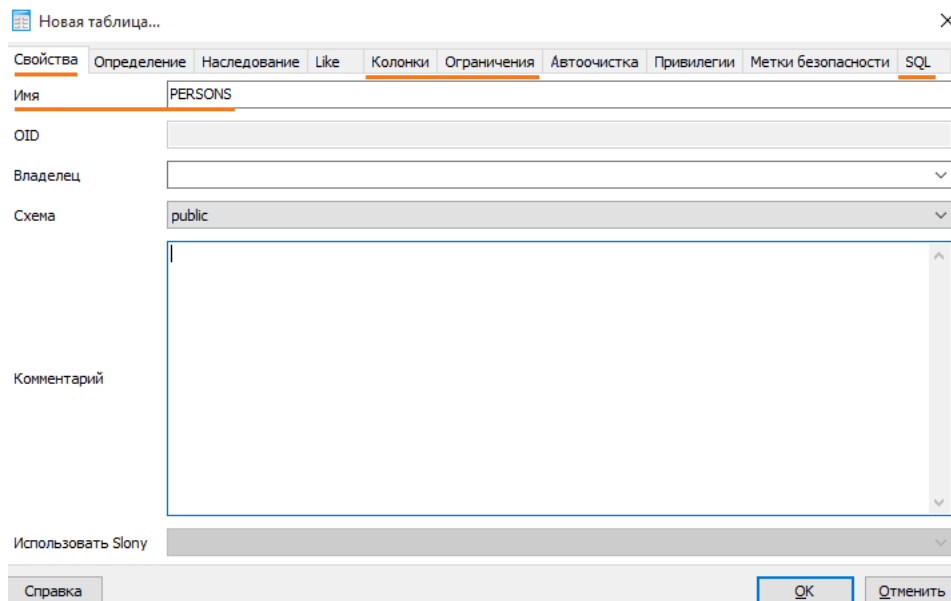


Рис. 5.44. Окно «Новая таблица», вкладка «Свойства»

Если перейти во вкладку «Колонки» и в правом нижнем углу нажать на кнопку «Добавить», то в результате откроется окно «Новая колонка» для добавления полей в таблицу (рис. 5.45). Важными в рассматриваемом примере являются два поля: «Имя» и «Тип данных», а также две вкладки: «Свойства» и «Определение». На вкладке «Определение» можно поставить галочку на разрешение Null-значения, либо задать значение по умолчанию. После нажатия на кнопку «Ок» произойдет закрытие данного окна и возврат во вкладку «Колонки», в которой уже будет присутствовать только что добавленное поле.

Далее переходим во вкладку «Ограничения». Она позволяет создавать первичные и внешние ключи. Разберем их создание на примере установки поля ID таблицы PERSONS первичным ключом. Выбираем в нижнем выпадающем списке «Первичный ключ» и нажимаем кнопку «Добавить», откроется окно «Новый первичный ключ». Задаем ему имя, например, ID_PERSONS, и переходим во вкладку «Колонки», затем в выпадающем списке выбираем поле «ID» и нажимаем «Добавить», а потом на кнопку «Ок». В результате данных манипуляций поле «ID» станет первичным ключом.

Последняя вкладка SQL позволяет просмотреть получающийся скрипт.

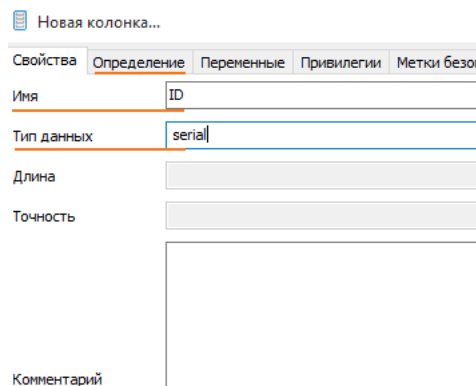


Рис. 5.45. Окно «Новая колонка»

Обратите внимание, что у идентификатора устанавливается тип данных «serial» для автоматической инкрементации описываемого поля. Итогом создания двух таблиц должна получиться структура в браузере объектов, идентичная изображенной на рисунке 5.46.

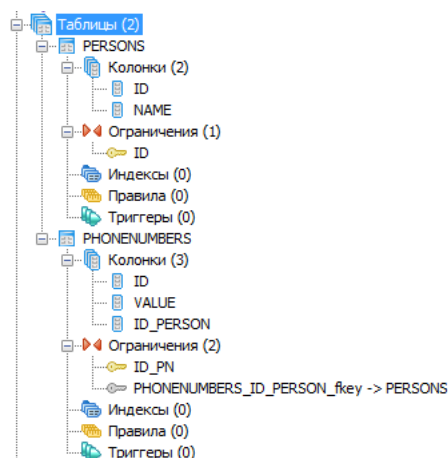


Рис. 5.46. Финальное дерево объектов с созданными таблицами и ограничениями.

Командная строка PostgreSQL

Тот же результат по созданию БД и таблиц предыдущего раздела можно получить с помощью командной строки PostgreSQL. Для тех, кто хорошо владеет SQL, это более удобный способ, потому что не требует установки дополнительного приложения. Но и для начинающих разработчиков это полезно, к примеру, в случае создания пустой копии существующей уже БД, потому что необходимо просто запустить уже сгенеренный ранее SQL-скрипт.

Для того чтобы создать таблицы в базе данных heroku через командную строку, необходимо к ней подключиться. Для этого в консоли запускаем команду:

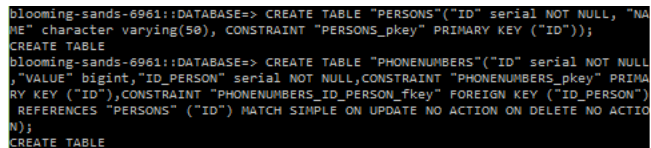
```
heroku pg:psql --app НАЗВАНИЕ_БАЗЫ_ДАННЫХ DATABASE
```

Данную команду можно скопировать из веб-интерфейса Heroku Postgres. Она доступна при просмотре подробных сведений о базе данных в строке с названием «psql». При выполнении команды осуществляется переход в раздел управления базой данных. Теперь нужно всего лишь запустить sql-команды на выполнение:

```
CREATE TABLE "PERSONS"("ID" serial NOT NULL, "NAME" CHARACTER VARYING(50), CONSTRAINT
"PERSONS_pkey" PRIMARY KEY ("ID"));

CREATE TABLE "PHONENUMBERS"("ID" serial NOT NULL,"VALUE" BIGINT,"ID_PERSON" serial NOT
NULL, CONSTRAINT "PHONENUMBERS_pkey" PRIMARY KEY ("ID"), CONSTRAINT
"PHONENUMBERS_ID_PERSON_fkey" FOREIGN KEY ("ID_PERSON") REFERENCES "PERSONS" ("ID")
MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION);
```

При успешном исполнении sql-команд в консоли выводится надпись «CREATE TABLE». Пример результата представлен на рисунке 5.47.



```
blooming-sands-69611:DATABASE=> CREATE TABLE "PERSONS"("ID" serial NOT NULL, "NA
ME" character varying(50), CONSTRAINT "PERSONS_pkey" PRIMARY KEY ("ID"));
CREATE TABLE
blooming-sands-69611:DATABASE=> CREATE TABLE "PHONENUMBERS"("ID" serial NOT NULL
,"VALUE" bigint,"ID_PERSON" serial NOT NULL,CONSTRAINT "PHONENUMBERS_pkey" PRIMA
RY KEY ("ID"),CONSTRAINT "PHONENUMBERS_ID_PERSON_fkey" FOREIGN KEY ("ID_PERSON")
REFERENCES "PERSONS" ("ID") MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTIO
N);
CREATE TABLE
```

Рис. 5.47. Исполнение sql-скриптов в командной строке.

Данные классы лучше расположить в отдельном пакете ru.myitschool.entity. Код классов сущностей представлен ниже, сначала Persons, затем PhoneNumbers.

```
package ru.myitschool.entity;

public class Persons {
    private Integer id;
    private String name;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

package ru.myitschool.entity;
public class PhoneNumbers {
    private Integer id;
    private Long value;
    private Integer idPerson;
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public Long getValue() {
        return value;
    }
    public void setValue(Long value) {
        this.value = value;
    }
    public Integer getIdPerson() {
        return idPerson;
    }
    public void setIdPerson(Integer idPerson) {
        this.idPerson = idPerson;
    }
}
```

Реализовать репозитории для сущностей

Конечно, полностью реализовывать классический паттерн «репозиторий» нет необходимости. Создадим классы с методами, позволяющими выполнять SQL-запросы к базе данных. Так же создадим объект класса JdbcTemplate, для того чтобы не прописывать ручным способом работу с соединением и источником данных. Код класса PersonRepository представлен ниже.

```

package ru.myitschool.repositories;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import ru.myitschool.entity.Persons;

@Component
public class PersonsRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public int createPerson(String name){
        return jdbcTemplate.update("INSERT INTO \"PERSONS\" (\"NAME\") VALUES (?)", name);
    }

    public int updatePerson(Persons person){
        return jdbcTemplate.update("UPDATE \"PERSONS\" SET \"NAME\" = ? WHERE
                                   \"ID\" = ?", person.getName(), person.getId());
    }

    public int deletePerson(Integer id){
        return jdbcTemplate.update("DELETE FROM \"PERSONS\" WHERE \"ID\" = ?",id);
    }

    public Persons getPerson(Integer id){
        return jdbcTemplate.queryForObject("SELECT * FROM \"PERSONS\" WHERE
                                           \"ID\"=?", new PersonsMapper(), id);
    }

    public List<Persons> getPersons(){
        return jdbcTemplate.query("SELECT * FROM \"PERSONS\"", new PersonsMapper());
    }
}

```

В методах `createPerson`, `updatePerson` и `deletePerson` нет ничего примечательно. У объекта `jdbcTemplate` просто вызывается метод `update()`, принимающий в качестве параметров SQL-запрос и переменные для этого запроса, значение которых подставляется вместо вопросительного знака. Наибольший интерес вызывают методы получения одной персоны и списка персон. В первом случае для выполнения запроса используется метод `queryForObject()`, во втором `query()`. В отличии от метода `update`, примечательно в них то, что они принимают еще один дополнительный объект типа `RowMapper<T>`, функцией которого является преобразование ответа от базы данных в требуемый для разработчика вид. На самом деле `RowMapper<T>` — это интерфейс, поэтому, для того чтобы создать на его основе объект, сначала создадим класс, который его имплементирует. Код класса `PersonsMapper` представлен ниже.

```

package ru.myitschool.repositories;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
import ru.myitschool.entity.Persons;

public class PersonsMapper implements RowMapper<Persons> {

    public Persons mapRow(ResultSet rs, int rowNum) throws SQLException {
        Persons person = new Persons();
        person.setId(rs.getInt("id"));
        person.setName(rs.getString("name"));
        return person;
    }
}

```

В результате имплементации интерфейса `RowMapper<T>` необходимо реализовать метод `mapRow()`. У него имеется два параметра: `ResultSet` (результатирующий набор, сюда приходит результат выполнения SQL-запроса) и целочисленная переменная `rowNum` (хранит номер строки). Создаем объект класса `Persons` и присваиваем значения его атрибутам.

Рассмотрим класс `PhoneNumberRepository`. У него есть интересный метод `getPhoneBook()`, позволяющий получить все записи (люди и их телефонные номера) из базы данных и преобразовать в вид JSON-строки. Для этого сначала создается два JSON-объекта. Объект класса `JSONObject` позволит поместить каждую конкретную запись, а `JSONArray` запакует все в массив. Далее создадим объекты классов `Connection` и `Statement`, напишем SQL-запрос и выполним его при помощи метода `executeQuery` объекта `stmt`, результат которого присвоим объекту класса `ResultSet`. При помощи цикла `while` делается обход всего ответа и методами `put()` он запаковывается в JSON. Ниже приведен код данного класса.

```

package ru.myitschool.repositories;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import ru.myitschool.entity.PhoneNumbers;

@Component
public class PhoneNumbersRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public int createPhoneNumber(Long value, Integer idPerson){
        return jdbcTemplate.update("INSERT INTO \"PHONENUMBERS\" (\"VALUE\",
            \"ID_PERSON\") VALUES (?,?)", value, idPerson);
    }

    public int updatePhoneNumber(PhoneNumbers phoneNumbers){
        return jdbcTemplate.update("UPDATE \"PHONENUMBERS\" SET \"VALUE\" = ?
            WHERE \"ID\" = ?", phoneNumbers.getValue(), phoneNumbers.getId());
    }

    public int deletePhoneNumbers(Integer id){
        return jdbcTemplate.update("DELETE FROM \"PHONENUMBERS\" WHERE \"ID\"= ?", id);
    }

    public JSONArray getPhoneBook(){
        JSONObject json;
        JSONArray jsonArr = new JSONArray();
        try {
            Connection conn=jdbcTemplate.getDataSource().getConnection();
            Statement stmt = conn.createStatement();
            String sql = "SELECT \"PHONENUMBERS\".\"ID\" AS \"ID\",
                \"PHONENUMBERS\".\"VALUE\" AS \"NUMBER\", \"PERSONS\".\"NAME\"
                AS \"NAMEPERSON\" FROM \"PHONENUMBERS\" LEFT JOIN \"PERSONS\"
                ON \"PERSONS\".\"ID\" = \"PHONENUMBERS\".\"ID_PERSON\";";
            ResultSet rs = stmt.executeQuery(sql);
            while(rs.next()){
                json = new JSONObject();
                json.put("name", rs.getString("NAMEPERSON"));
                json.put("value", rs.getString("NUMBER"));
                jsonArr.put(json);
            }
        } catch (SQLException e) {
            e.getLocalizedMessage();
            return null;
        } catch (JSONException e) {
            e.getLocalizedMessage();
            return null;
        }
        return jsonArr;
    }
}

```

Написать CRUD-контроллеры

Создадим еще один пакет с названием `ru.myitschool.controllers` и два класса: `PersonsController` и `PhoneNumberController`. Код обоих классов представлен ниже.

```

package ru.myitschool.controllers;
import java.util.List;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import ru.myitschool.entity.Persons;
import ru.myitschool.repositories.PersonsRepository;

@RestController
@RequestMapping("person")
public class PersonsController {

    @Autowired
    private PersonsRepository person;

    @RequestMapping(value="/create", method=RequestMethod.PUT, consumes="text/plain")
    public int createPerson(@RequestBody String param){
        String name = null;
        try{
            JSONObject json = new JSONObject(param);
            name = json.getString("name");
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return person.createPerson(name);
    }

    @RequestMapping(value="update",method=RequestMethod.POST,consumes="text/plain")
    public int updatePerson(@RequestBody String param){
        Persons p = new Persons();
        try{
            JSONObject json = new JSONObject(param);
            p.setId(json.getInt("id"));
            p.setName(json.getString("name"));
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return person.updatePerson(p);
    }

    @RequestMapping(value="{id}", method=RequestMethod.DELETE)
    public int deletePerson(@PathVariable Integer id){
        return person.deletePerson(id);
    }

    @RequestMapping(value = "/getperson", method=RequestMethod.GET)
    public Persons getPerson(@RequestParam("id") Integer id){
        return person.getPerson(id);
    }

    @RequestMapping(value = "/getpersons", method=RequestMethod.GET)
    public List<Persons> getPersons(){
        return person.getPersons();
    }
}

```

Код класса PhoneNumbersController:

```

package ru.myitschool.controllers;

import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import ru.myitschool.entity.PhoneNumbers;
import ru.myitschool.repositories.PhoneNumbersRepository;

@RestController
@RequestMapping("pn")
public class PhoneNumberController {

    @Autowired
    private PhoneNumbersRepository phoneNumbersRepository;

    @RequestMapping(value="/create",method=RequestMethod.PUT,consumes="text/plain")
    public int createPhoneNumber(@RequestBody String param){
        Long value = null;
        Integer id = null;
        try{
            JSONObject json = new JSONObject(param);
            value = json.getLong("value");
            id = json.getInt("idperson");
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return phoneNumbersRepository.createPhoneNumber(value, id);
    }

    @RequestMapping(value="update",method=RequestMethod.POST,consumes="text/plain")
    public int updatePhoneNumber(@RequestBody String param){
        PhoneNumbers pn = new PhoneNumbers();
        try{
            JSONObject json = new JSONObject(param);
            pn.setId(json.getInt("id"));
            pn.setValue(json.getLong("value"));
            pn.setIdPerson(json.getInt("idperson"));
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return phoneNumbersRepository.updatePhoneNumber(pn);
    }

    @RequestMapping(value="{id}", method=RequestMethod.DELETE)
    public int deletePhoneNumber(@PathVariable Integer id){
        return phoneNumbersRepository.deletePhoneNumbers(id);
    }

    @RequestMapping(value = "/getpb", method=RequestMethod.GET)
    public String getItemPhoneBook(){
        return phoneNumbersRepository.getPhoneBook().toString();
    }
}

```

Поскольку оба класса идентичны, рассмотрим последний из них. Для класса `PhoneNumbersController` указано две аннотации. `@RestController` указывает на то, что данный класс является контроллером по архитектуре REST. `@RequestMapping` («pn») означает, что для доступа к методам данного класса URL-адрес будет иметь вид: `http://localhost:8080/pn/`. Затем идет определение объекта репозитория, на базе описанного ранее класса. У данного объекта имеется аннотация `@Autowired`, позволяющая Spring автоматически устанавливать значение полей. Далее определены четыре метода-контроллера. У каждого из них присутствует аннотация `@RequestMapping`, позволяющая указать: путь (по которому данный метод будет доступен), тип запроса по архитектуре REST (GET, POST, PUT, DELETE и т. д.). У двух методов указан еще один параметр для данной аннотации — `consumes`, со значением «text/plain», сигнализирующий о том, что входящим параметром является простая строка (текст).

Суть написанных методов достаточно проста. Они должны принять какую-либо информацию через параметры, возможно, немного ее подготовить, а затем вызвать метод(ы), например, репозитория, чтобы получить от них ответ и отправить его клиенту. Если обратить внимание на параметры данных методов, то можно увидеть, что там также используются аннотации:

- `@RequestBody` — информация передается в теле запроса;
- `@PathVariable` — значение является частью пути (`http://localhost:8080/pn/8`);
- `@RequestParam` — передача значения через переменную HTTP (`http://localhost:8080/pn/get?id=8`).

Использование аннотаций — одно из условий, которое ставит SpringBoot для обеспечения простоты написания кода, на основе данного фреймворка.

5.5.3. Реализация back end части приложения на языке Java

На профессиональном сленге разработчиков клиент-серверных программ часть серверной программы, которая занимается сохранением/получением данных в/из СУБД называется **back end**. Написание именно такой программы и будет рассмотрено в текущем примере.

Создадим проект *Spring Starter Project* для написания back end функционала. Для этого выполним следующие действия: File ⇒ New ⇒ Project. Затем в списке выберем раздел *Spring* и далее пункт *Spring Starter Project*. Появится окно создания нового стартового проекта Spring. Заполним необходимые поля (действительно необходимые только первое и последнее) и нажмем кнопку Finish:

- Name (название проекта) — PhoneBook;
- Type (система сборки проекта) — Maven Project;
- Java Version — 1.8;
- Boot Version (версия фреймворка Spring Boot) — 1.3.1;
- Packaging — jar;
- Language — Java;
- Group (группа, команда, можно указать сайт разработчика) — ru.myitschool;
- Artifact (задаем такой же, как название приложения) — PhoneBook;
- Version — 0.0.1-SNAPSHOT;
- Description — Phone book project for Spring Boot;
- Package — ru.myitschool;
- Dependencies — Web.

Завершая создание проекта кнопкой Finish, в Package Explorer появится наш проект (рис. 5.48).

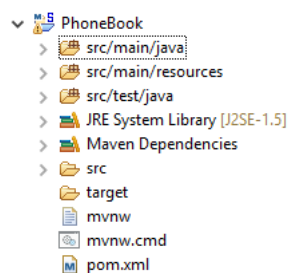


Рис. 5.48. Структура проекта PhoneBook

Теперь необходимо убедиться в том, что проект корректно запускается, но если это сделать прямо сейчас, то при обращении в браузере отобразится страничка 404. Поэтому раскроем самую первую ветку (*src/main/java*), затем пакет (*ru.myitschool*), и создадим в нем класс *HomeController*. Напишем метод *index*, возвращающий строковое значение «Привет, мир!». Ниже приведен код.

```
package ru.myitschool;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HomeController {

    @RequestMapping("/")
    public String index() {
        return "Привет, мир!";
    }
}
```

Для запуска приложения нажимаем правой кнопкой мыши на название проекта, Run As ⇒ Spring Boot App. В консоль выводится процесс запуска приложения. Если запуск прошел успешно, в последней строке в консоли будет написана фраза «Started PhoneBookApplication». В адресной строке браузера вводим <http://localhost:8080/>. В результате чего должна загрузиться страница с надписью «Привет, мир!». Даже в таком простом примере, с единственным методом вывода строкового значения, использовались важные особенности фреймворка Spring, а именно аннотации (*@RestController*, *@RequestMapping*(«/»)). Они помогают простым и удобным способом обозначить назначение какого-либо элемента программы. Например, аннотация «*@RestController*» говорит о том, что класс *HomeController* является контроллером, а «*@RequestMapping*(«/»» позволяет задавать пути запросов.

Последним подготовительным шагом для успешной дальнейшей работы является добавление дополнительных библиотек. Так как используется система сборки проекта Maven, сделать это достаточно просто, нужно лишь добавить необходимые зависимости в файле *pom.xml* (находится в корне проекта). Ниже указаны используемые зависимости.

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>9.4-1203-jdbc42</version>
</dependency>
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20151123</version>
</dependency>

```

Все подготовительные шаги выполнены и можно приступить к написанию «главных» функций сервера. Для этого создается базовый CRUD (Create, Read, Update, Delete) функционал для представленных ранее сущностей базы данных, а именно Persons и PhoneNumbers. Посмотрим на те шаги, которые нужно выполнить для решения данной задачи:

- создать подключение к базе данных, располагающейся на сервисе heroku;
- представить сущности Persons и PhoneNumbers в виде java-классов;
- реализовать на их основе репозитории;
- написать CRUD-контроллеры.

Создать подключение к базе данных, располагающейся на сервисе heroku

Для этого в среде разработки раскроем ветку «src/main/resource» «application.properties». Затем добавим следующие строки:

```

spring.datasource.url=jdbc:postgresql://host:port/database?sslmode=require
spring.datasource.username=
spring.datasource.password=
spring.datasource.driver-class-name=org.postgresql.Driver

```

В первой строке, указывая URL (строку подключения), перенесем данные со страницы параметров базы данных Heroku, как это было сделано при подключении к серверу через pgAdmin. В результате получится строка подключения, аналогично как на рисунке 5.49.



Рис. 5.49. Строка подключения (spring.datasource.url).

Перенесем данные об имени пользователя и пароле аналогичным образом. Особое внимание нужно уделить наличию параметра «sslmode=require», так как при его отсутствии подключение к базе данных будет невозможно (это специфическая особенность Heroku, да и вообще многих других облачных сервисов). В последней строке указывается название драйвера базы данных.

2. Представить сущности Persons и PhoneNumbers в виде java-классов

Данные классы лучше расположить в отдельном пакете `ru.myitschool.entity`. Код сущностей представлен ниже, сначала Persons, затем PhoneNumbers.

```

package ru.myitschool.entity;

public class Persons {
    private Integer id;
    private String name;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

package ru.myitschool.entity;

public class PhoneNumbers {
    private Integer id;
    private Long value;
    private Integer idPerson;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Long getValue() {
        return value;
    }

    public void setValue(Long value) {
        this.value = value;
    }

    public Integer getIdPerson() {
        return idPerson;
    }

    public void setIdPerson(Integer idPerson) {
        this.idPerson = idPerson;
    }
}

```

3. Реализовать репозитории для сущностей

В данном случае, для упрощения, не будем реализовывать паттерн «репозиторий» в привычном понимании. Создадим классы с методами, позволяющими выполнять SQL-запросы к базе данных. Так же создадим объект класса `JdbcTemplate`, для того что бы не прописывать ручным способом работу с соединением и источником данных. Код класса `PersonRepository` представлен ниже.


```

package ru.myitschool.repositories;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import ru.myitschool.entity.Persons;

@Component
public class PersonsRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public int createPerson(String name){
        return jdbcTemplate.update("INSERT INTO \"PERSONS\" (\"NAME\") VALUES(?)", name);
    }

    public int updatePerson(Persons person){
        return jdbcTemplate.update("UPDATE \"PERSONS\" SET \"NAME\" = ? WHERE \"ID\" = ?", person.getName(), person.getId());
    }

    public int deletePerson(Integer id){
        return jdbcTemplate.update("DELETE FROM \"PERSONS\" WHERE \"ID\" = ?", id);
    }

    public Persons getPerson(Integer id){
        return jdbcTemplate.queryForObject("SELECT * FROM \"PERSONS\" WHERE \"ID\"=?", new PersonsMapper(), id);
    }

    public List<Persons> getPersons(){
        return jdbcTemplate.query("SELECT * FROM \"PERSONS\"", new PersonsMapper());
    }
}

```

В методах `createPerson`, `updatePerson` и `deletePerson` нет ничего примечательно. У объекта `jdbcTemplate` просто вызывается метод `update()`, принимающий в качестве параметров SQL-запрос и переменные для этого запроса, значение которых подставляется вместо вопросительного знака. Наибольший интерес вызывают методы получения одной персоны и списка персон. В первом случае для выполнения запроса используется метод `queryForObject()`, во втором `query()`. В отличие от метода `update()` примечательно в них то, что они принимают еще один дополнительный объект типа `RowMapper<T>`, функцией которого является преобразование ответа от базы данных в требуемый для разработчика вид. На самом деле `RowMapper<T>` — это интерфейс, поэтому для того чтобы создать на его основе объект, сначала создадим класс, который его имплементирует. Код класса `PersonsMapper` представлен ниже.

```

package ru.myitschool.repositories;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
import ru.myitschool.entity.Persons;

public class PersonsMapper implements RowMapper<Persons> {

    public Persons mapRow(ResultSet rs, int rowNum) throws SQLException {
        Persons person = new Persons();
        person.setId(rs.getInt("id"));
        person.setName(rs.getString("name"));
        return person;
    }
}

```

В результате имплементации интерфейса `RowMapper<T>` необходимо реализовать метод `mapRow()`. У него имеется два параметра:

`ResultSet` (результатирующий набор, сюда приходит результат выполнения SQL-запроса) и целочисленная переменная `rowNum` (хранит номер строки). Создаем объект класса

Рассмотрим класс `PhoneNumberRepository`. У него есть интересный метод `getPhoneBook()`, позволяющий получить все записи (люди и их телефонные номера) из базы данных и преобразовать в вид json-строки. Для этого сначала создает два json-объекта. Объект класса `JSONObject` позволит поместить каждую конкретную запись, а `JSONArray` запакует все в массив. Далее создадим объекты классов `Connection` и `Statement`, напишем SQL-запрос и выполним его при помощи метода `executeQuery` объекта `stmt`, результат которого присвоим объекту класса `ResultSet`. При помощи цикла `while` делается обход всего ответа и методами `put()` он запаковывается в json. Ниже приведен код данного класса.

```

package ru.myitschool.repositories;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;
import ru.myitschool.entity.PhoneNumbers;

@Component
public class PhoneNumbersRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public int createPhoneNumber(Long value, Integer idPerson){
        return jdbcTemplate.update("INSERT INTO \"PHONENUMBERS\" (\"VALUE\", \"ID_PERSON\") VALUES (?,?)", value, idPerson);
    }

    public int updatePhoneNumber(PhoneNumbers phoneNumbers){
        return jdbcTemplate.update("UPDATE \"PHONENUMBERS\" SET \"VALUE\" = ? WHERE \"ID\" = ?", phoneNumbers.getValue(), phoneNumbers.getId());
    }

    public int deletePhoneNumbers(Integer id){
        return jdbcTemplate.update("DELETE FROM \"PHONENUMBERS\" WHERE \"ID\" = ?", id);
    }

    public JSONArray getPhoneBook(){
        JSONObject json;
        JSONArray jsonArr = new JSONArray();
        try {
            Connection conn=jdbcTemplate.getDataSource().getConnection();
            Statement stmt = conn.createStatement();
            String sql = "SELECT \"PHONENUMBERS\".\"ID\" AS \"ID\", \"PHONENUMBERS\".\"VALUE\" AS \"NUMBER\", \"PERSONS\".\"NAME\" AS \"NAMEPERSON\" FROM \"PHONENUMBERS\" LEFT JOIN \"PERSONS\" ON \"PERSONS\".\"ID\" = \"PHONENUMBERS\".\"ID_PERSON\"";
            ResultSet rs = stmt.executeQuery(sql);
            while(rs.next()){
                json = new JSONObject();
                json.put("name", rs.getString("NAMEPERSON"));
                json.put("value", rs.getString("NUMBER"));
                jsonArr.put(json);
            }
        } catch (SQLException e) {
            e.getLocalizedMessage();
            return null;
        } catch (JSONException e) {
            e.getLocalizedMessage();
            return null;
        }
        return jsonArr;
    }
}

```

4. Написать CRUD контроллеры

Создадим еще один пакет с названием `ru.myitschool.controllers` и два класса: `PersonsController` и `PhoneNumberController`. Код обоих классов представлен ниже.

```

package ru.myitschool.controllers;

import java.util.List;
import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import ru.myitschool.entity.Persons;
import ru.myitschool.repositories.PersonsRepository;

@RestController
@RequestMapping("person")
public class PersonsController {

    @Autowired
    private PersonsRepository person;

    @RequestMapping(value = "/create", method=RequestMethod.PUT, consumes="text/plain")
    public int createPerson(@RequestBody String param){
        String name = null;
        try{
            JSONObject json = new JSONObject(param);
            name = json.getString("name");
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return person.createPerson(name);
    }

    @RequestMapping(value = "update", method=RequestMethod.POST, consumes="text/plain")
    public int updatePerson(@RequestBody String param){
        Persons p = new Persons();
        try{
            JSONObject json = new JSONObject(param);
            p.setId(json.getInt("id"));
            p.setName(json.getString("name"));
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return person.updatePerson(p);
    }

    @RequestMapping(value="{id}", method=RequestMethod.DELETE)
    public int deletePerson(@PathVariable Integer id){
        return person.deletePerson(id);
    }

    @RequestMapping(value = "/getperson", method=RequestMethod.GET)
    public Persons getPerson(@RequestParam("id") Integer id){
        return person.getPerson(id);
    }

    @RequestMapping(value = "/getpersons", method=RequestMethod.GET)
    public List<Persons> getPersons(){
        return person.getPersons();
    }
}

```

Код класса PhoneNumbersController:

```

package ru.myitschool.controllers;

import org.json.JSONException;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import ru.myitschool.entity.PhoneNumbers;
import ru.myitschool.repositories.PhoneNumbersRepository;

@RestController
@RequestMapping("pn")
public class PhoneNumberController {

    @Autowired
    private PhoneNumbersRepository phoneNumbersRepository;

    @RequestMapping(value = "/create", method=RequestMethod.PUT, consumes="text/plain")
    public int createPhoneNumber(@RequestBody String param){
        Long value = null;
        Integer id = null;
        try{
            JSONObject json = new JSONObject(param);
            value = json.getLong("value");
            id = json.getInt("idperson");
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return phoneNumbersRepository.createPhoneNumber(value, id);
    }

    @RequestMapping(value = "update", method=RequestMethod.POST, consumes="text/plain")
    public int updatePhoneNumber(@RequestBody String param){
        PhoneNumbers pn = new PhoneNumbers();
        try{
            JSONObject json = new JSONObject(param);
            pn.setId(json.getInt("id"));
            pn.setValue(json.getLong("value"));
            pn.setIdPerson(json.getInt("idperson"));
        }catch(JSONException e){
            e.getLocalizedMessage();
            return 0;
        }
        return phoneNumbersRepository.updatePhoneNumber(pn);
    }

    @RequestMapping(value="{id}", method=RequestMethod.DELETE)
    public int deletePhoneNumber(@PathVariable Integer id){
        return phoneNumbersRepository.deletePhoneNumbers(id);
    }

    @RequestMapping(value = "/getpb", method=RequestMethod.GET)
    public String getItemPhoneBook(){
        return phoneNumbersRepository.getPhoneBook().toString();
    }
}

```

Так как оба класса похожи друг на друга, рассмотрим последний из них. Для класса `PhoneNumbersController` указано две аннотации. `@RestController` указывает на то, что данный класс является контроллером по архитектуре REST. `@RequestMapping("pn")`, означает, что для доступа к методам данного класса URL адрес будет иметь вид: <http://localhost:8080/pn/>. Затем идет определение объекта репозитория, на базе созданного ранее класса. У данного объекта имеется аннотация `@Autowired`, позволяющая автоматически устанавливать значение поля. Далее определены четыре метода-контроллера. У каждого из них присутствует аннотация `@RequestMapping`, позволяющая указать: путь (по которому данный метод будет доступен), тип запроса по архитектуре REST (GET, POST, PUT, DELETE и т.д.). У двух методов указан еще один параметр для данной аннотации — `consumes`, со значением «text/plain», сигнализирующий о том, что входящим параметром является простая строка (текст).

Суть написанных методов достаточно проста. Они должны принять какую-либо информацию через параметры, возможно немного ее подготовить, а затем вызвать метод(ы), например, репозитория, чтобы получить от них ответ и отправить его клиенту. Если обратить внимание на параметры данных методов, то можно увидеть, что там также используются аннотации:

- `@RequestBody` — информация передается в теле запроса;
- `@PathVariable` — значение является частью пути (<http://localhost:8080/pn/8/>);
- `@RequestParam` — передача значения через переменную пути (<http://localhost:8080/pn/get?id=8>)

Использование аннотаций — одно из условий, которое ставит SpringBoot для обеспечения простоты написания кода, на основе данного фреймворка. В качестве дополнительного материала рекомендуется почитать статьи на официальном сайте данного фреймворка. А здесь можно посмотреть, как писать на SpringBoot без использования строк запросов на чистом SQL.

5.5.4. Реализация back end части приложения на языке PHP

В этой главе рассмотрим, как реализовать серверную часть приложения, выполненную ранее на языке Java, при помощи скриптового языка программирования PHP.

Для написания кода серверной части есть несколько устоявшихся подходов, один из которых использовался в предыдущем разделе, называется *MVC* — Model View Controller. Этот подход служит для разделения кода работы с базой данных от кода непосредственно серверной части и от интерфейса пользователя. Согласно этому подходу весь код, реализующий CRUD-функционал, размещается в моделях, код необходимый для организации работы сервера размещается в контроллерах, и соответственно интерфейс размещается в представлениях. *Модель* — это класс, представляющий собой описание сущности базы данных (например, таблица). В рассматриваемом случае будет две модели — персоны и телефонные номера. Также добавим еще одну служебную модель — dbConfig, в которой будем описывать методы для работы непосредственно с базой данных (вызов функций, подключение к базе и т. д.).

Первая модель будет называться «persons», она описывает сущность «Персоны». Приведем часть кода этой модели:

```
include "dbFunc.php";
class persons {
    var $id;
    var $name;
    var $result;
    var $objDb;

    function persons() {
        //конструктор
        $this->objDb = new dbConfig();
        if($this->objDb->getResult()!=1) {
            echo "Database connection error.";
            exit;
        }
    }
}
```

Здесь в первой строке подключается служебная модель dbConfig, которая находится в файле *dbFunc.php*. Для подключения сторонних файлов в скрипт в PHP есть функция «include». Далее объявляем класс persons и объявляем его атрибуты. После чего описываем конструктор класса, в котором создаем подключение к базе данных и, при необходимости, сообщаем об ошибке в случае неудачного соединения с базой.

Также этот класс имеет методы для создания, изменения, удаления и чтения данных. Все эти методы имеет одинаковую структуру и отличаются только названием и кодом для работы с базой. Приведем для примера код метода добавления новой записи в таблицу.

```
function newPersons($name) {
    //метод для добавления новой персоны
    $textQuery = "INSERT INTO `PERSONS` (`NAME`) VALUES ('".$name."')";
    $this->objDb->executeQuery($textQuery);
    $this->result = $this->objDb->getResult();
}
```

Этот метод принимает переменную \$name, в которой содержится имя добавляемой персоны. Далее в переменную \$textQuery записывается запрос на языке SQL для вставки данных в таблицу. Потом этот запрос передается в функцию executeQuery служебной модели dbConfig. Именно эта функция посылает запрос в базу и возвращает результат. Результат операции получаем с помощью функции getResult() служебной модели. Таким образом, меняя код запроса, можно реализовать весь CRUD-функционал.

Модель phoneNumber выглядит точно также, как и модель persons, только с атрибутами для сущности телефонных номеров. Интересной для рассмотрения является модель dbConfig. Рассмотрим ее подробнее и приведем весь ее код.

```

<?php
class dbConfig {
//класс-конфиг для работы с базой данных
    var $result;
    var $db;

    function dbConfig() {
        $this->result=0;
        $connectionString="host=ec2-54-217-231-152.eu-west-.compute.amazonaws.com
                                port=5432
                                dbname=d80fg54admdmct
                                user=usyzfnmscfzbmv
                                password=gtRV3ci01pbD_Zqqjlc-I6FrRe
                                sslmode=require";

        # Установка соединения с базой данных
        $this->db = pg_connect($connectionString);
        if (!$this->db) {
            $this->result=0;
            exit;
        }
        else {
            $this->result=1;
        }
    }

    function executeQuery($textQuery) {
        $resultQuery = pg_query($this->db, $textQuery);
        if(!$resultQuery) {
            $this->result=0;
            exit;
        }
        else {
            $this->result=1;
            return $resultQuery;
        }
    }

    function getResult() {
        return $this->result;
    }
}
?>

```

Все стандартно начинается с объявления класса dbConfig, далее описываются атрибуты и конструктор. В конструкторе задается строка подключения к базе данных, расположенной в сервисе Heroku, и производится попытка соединения с ней. Функция executeQuery принимает текст SQL-запроса и с помощью функции pg_query, которая работает с базой данных отправляет этот запрос на сервер.

На этом рассмотрение моделей можно закончить и перейти к контроллерам. Контроллеры в рассматриваемом серверном приложении будут являться чем-то вроде шлюзов, их задачей будет принять запрос от клиента и перенаправить его модели, а после получения ответа от модели вернуть ответ клиенту. В примере присутствует две сущности — Персоны и Телефонные номера, значит необходимо сделать два контроллера. Их структура абсолютно одинакова, различаются только имена переменных и функций. Приведем начало контроллера для работы с сущностью «Персоны».

```

<?php
include "personsModel.php";
$action = $_GET["action"];
$objPerson = new persons();
switch($action) {
    case 'get':
        $masPerson = $objPerson->listPersons();
        echo "List data: ".json_encode($masPerson);
        break;
    case 'getDetails':
        $itemPerson = $objPerson->getPerson($_GET["id"]);
        echo "Current item: ".json_encode($itemPerson);
        break;
    case 'post':
        $objPerson->newPersons($_POST["json"]);
        echo ($objPerson->getResult() != 1) ? "Error" : "Success";
        break;
}
}
?>

```

В приведенном программном коде сначала подключаем файл с моделью persons уже известной командой include. Далее считывает из служебной переменной \$_GET с указанием ключа, идентификатор действия, которое контроллер должен выполнить. Стоит привести пример запроса, который будет отправляться на сервер: ~/personController.php?action=ДЕЙСТВИЕ. То есть здесь в контроллер передается переменная action с определенным значением. Согласно правилам построения клиент-серверных приложений определено четыре типа действий:

- post — создание объекта;
- put — изменение объекта;
- delete — удаление объекта;

- `get` — получение объекта.

Эти четыре действия позволяют организовать REST подобное API. Однако, язык PHP в силу своих особенностей не позволяет в полной мере реализовать API, поэтому добавим некоторые свои типы действий, например, действие `getDetails` для получения сведений о конкретном объекте. После получения типа действия запускаем условный оператор `switch`, в котором проверяем значение переменной `$action` и выполняем требуемое действие. Таким образом, реализуя контроллеры для сущностей, обеспечиваем выполнение запланированных функций по работе с базой данных и изменения данных в ней.

5.6.* Дизайн программного обеспечения и приложений Material Design

Сайт: IT Академия SAMSUNG

Курс: MDev @ IT Академия Samsung

Книга: 5.6.* Дизайн программного обеспечения и приложений Material Design

Напечатано:: Егор Беляев

Дата: Суббота, 18 Апрель 2020, 19:35

Оглавление

1. 5.6.1. Введение в Material Design
2. 5.6.2. Создание проекта Material Design
3. 5.6.3. Пример приложения с использованием элементов MD

1. 5.6.1. Введение в Material Design

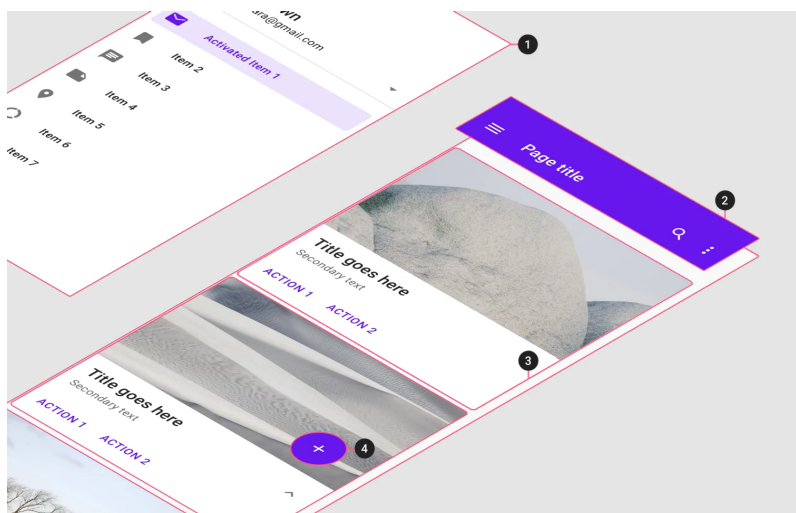
Принципы построения графического интерфейса пользователя для мобильных приложений в общем и целом общеизвестны и интуитивно понятны. Однако они формулировались зачастую в виде отдельных разрозненных best practices или статей в специализированных ресурсах.

Однако в 2014г на конференции Google I/O представила свою, достаточно всеобъемлющую концепцию построения интерфейсов. Google создал всеобъемлющую модель интерфейса, отвечающего всем современным запросам пользователей как физической системы. Т.е. каждый элемент интерфейса в этой модели обладает набором некоторых физических свойств, как то скоростью движения расположением в трехмерном пространстве. При взаимодействии с пользователем или с другим и элементами UI он начинает изменяться - перемещаться по определенным законам как по горизонтали так и по оси Z (в вертикальной стопке). Кроме того движение элементов меняет и положение теней и доступность других элементов, а иногда и их форму.

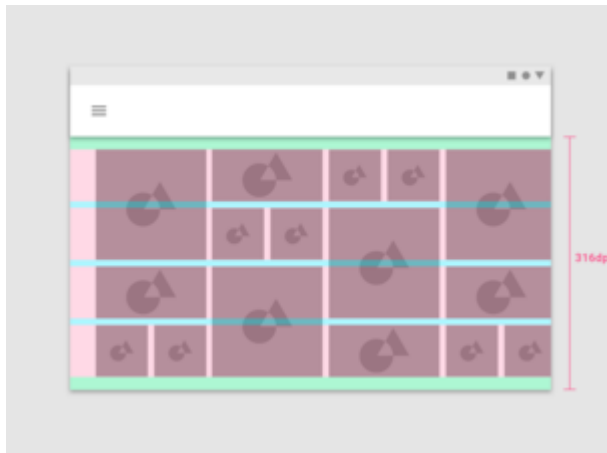
На основе этой модели, Google сформировал сборник рекомендаций по правильному построения интерфейсов. Следование этим требованиям позволяет получить удобный, красивый и интуитивно понятный пользовательский интерфейс, отвечающий современным требованиям и ожиданиям пользователей. Кроме рекомендаций был разработан фреймворк с новыми экранными элементами отвечающими требованиям материального дизайна. Также представлены библиотеки обратной совместимости для более ранних версий Android.

Описание модели и список рекомендаций можно посмотреть здесь: <https://material.io/design/> . Разделы рекомендаций включают следующее:

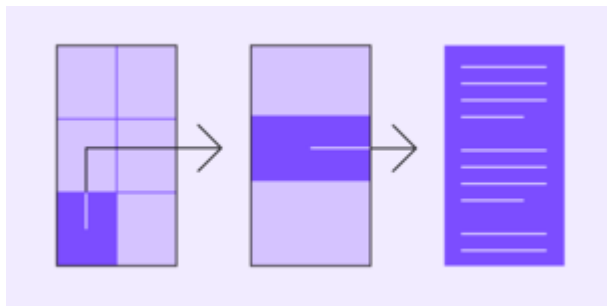
- Environment - описание физической и геометрической модели UI , В частности расположение основных поверхностей, модели их движения и модели теней,



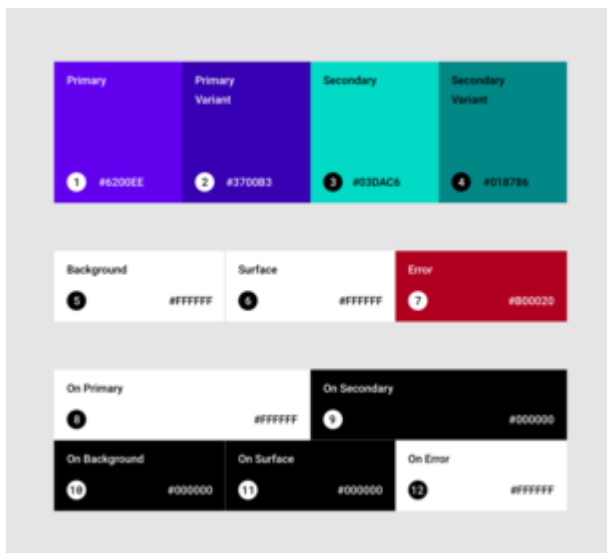
- Layout - рекомендации по построению макета - разрешение и пиксельная плотность, колонки столбцы и отступы,



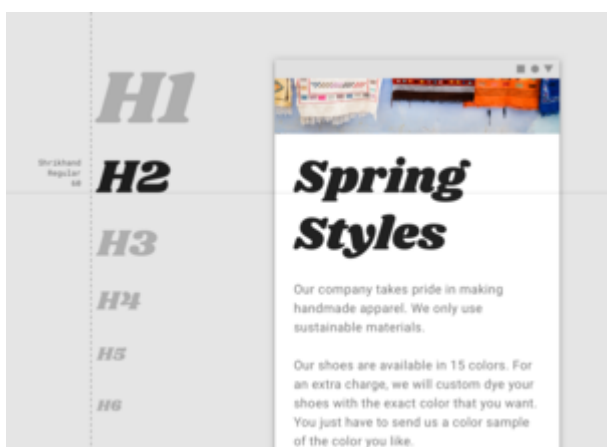
- Navigation - рекомендации по навигации в приложении между экранными формами. Как прямая навигация, так и "назад", "вверх" и более сложные варианты,



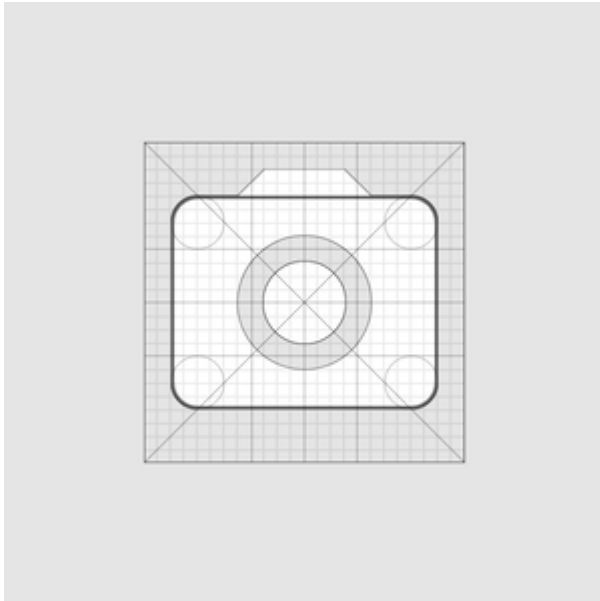
- Color - рекомендации по созданию тем, использованию палитры, использованию цветов шрифтов,



- Typography - рекомендации по использованию типографики для наиболее четкого и эффективного представления дизайна и контента, использования шрифтов, стилей, размеров, а также дизайна элементов текста,



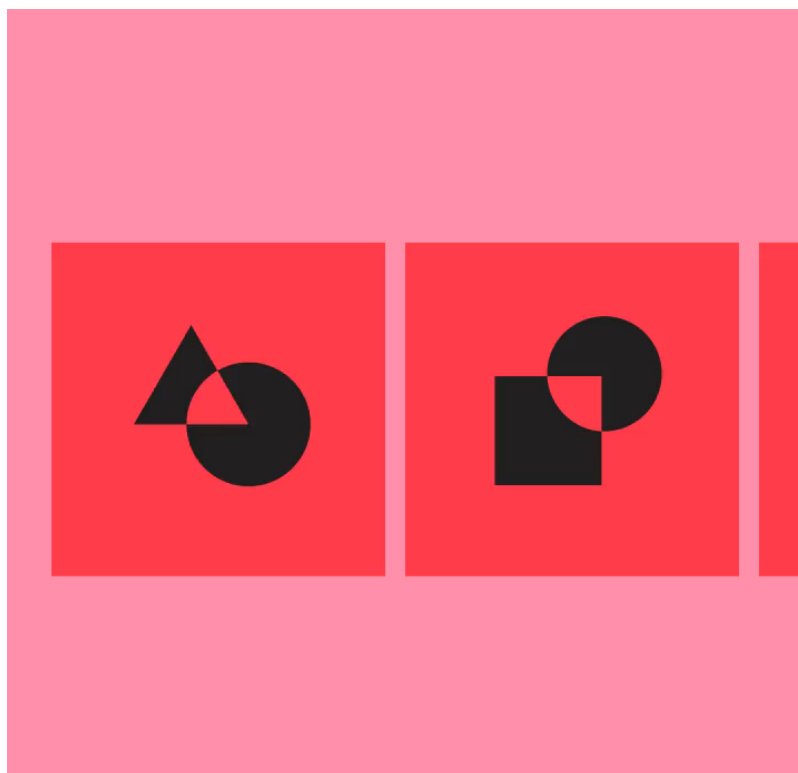
- Iconography - рекомендации по созданию иконок - свет, цвет, объем, метрики, анимации,



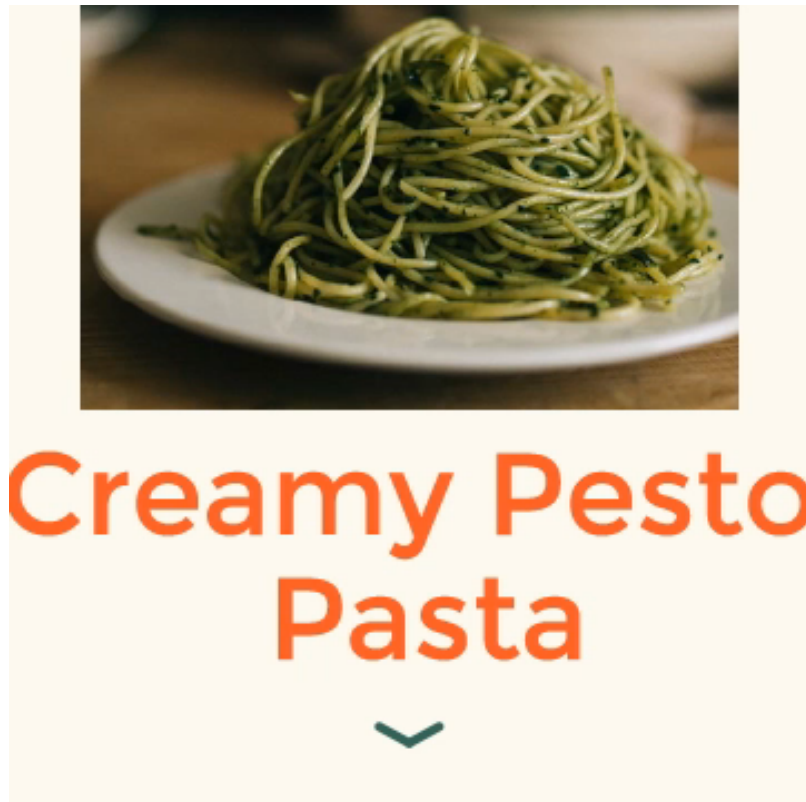
- Shape - рекомендации по формам поверхностей материального дизайна, в том числе поверхностей и краев элементов, углы, динамика форм,



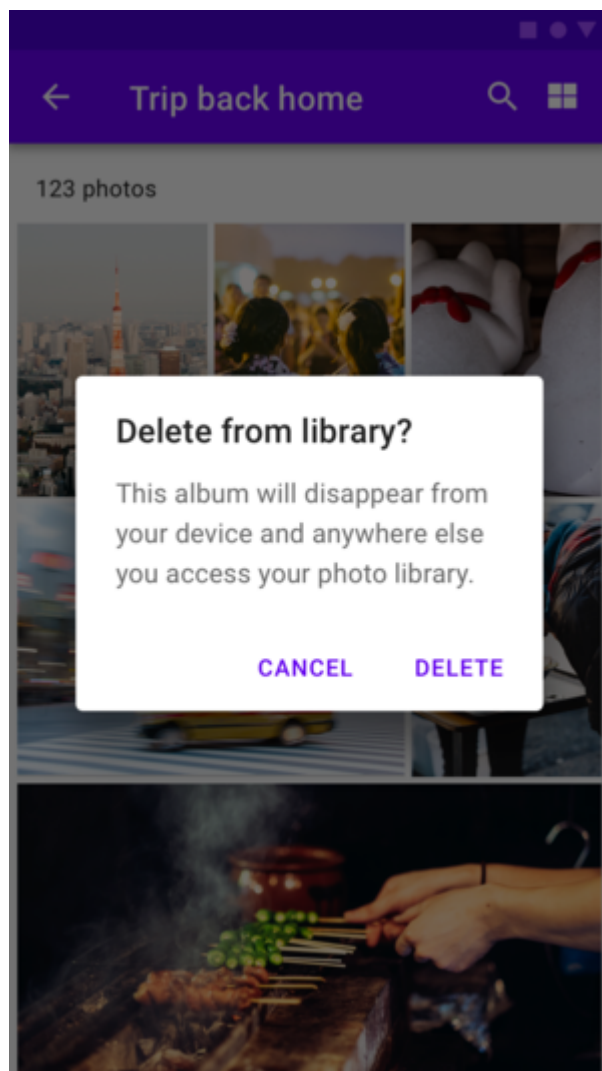
- Motion - рекомендации по принципам движения элементов материального дизайна, которое подчеркивает связь элементов, в том числе такие характеристики, как скорость, композиция и т.д.



- Interaction - рекомендацию по дизайну взаимодействия с элементами путем тача, в т.ч. жесты (gestures), выборы (selections), состояния (states),



- Communication - рекомендации по разработке алгоритмов взаимодействия пользователя с приложением, в т.ч. подтверждения, экранные форматы данных, обратная связь (feedback) и т.д.



2. 5.6.2. Создание проекта Material Design

Для создания проекта желательно иметь:

- обновленную версию вашей среды разработки, например Android Studio 1.5+ и JDK 7+
- Android устройство с API 4.1+ . Могут быть использованы устройства и с более младшим API - Android 2.3.3 (Gingerbread, API Level 10) или выше, однако, при этом часть эффектов (например, круговая анимация) может не воспроизводиться.

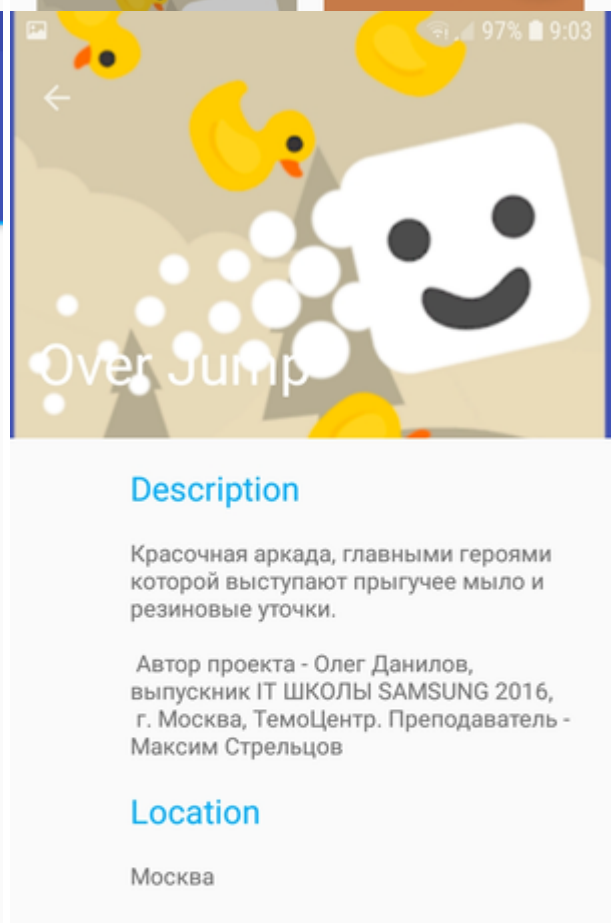
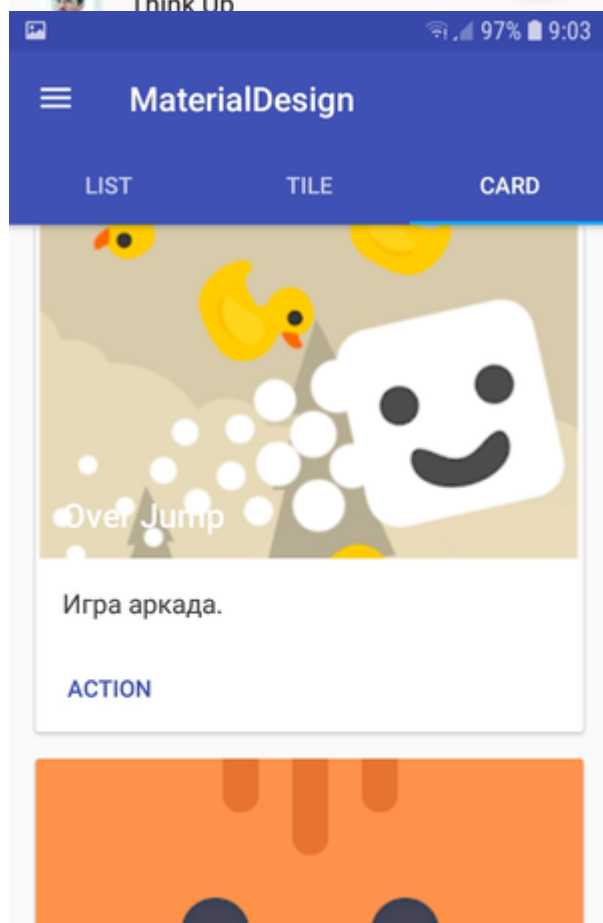
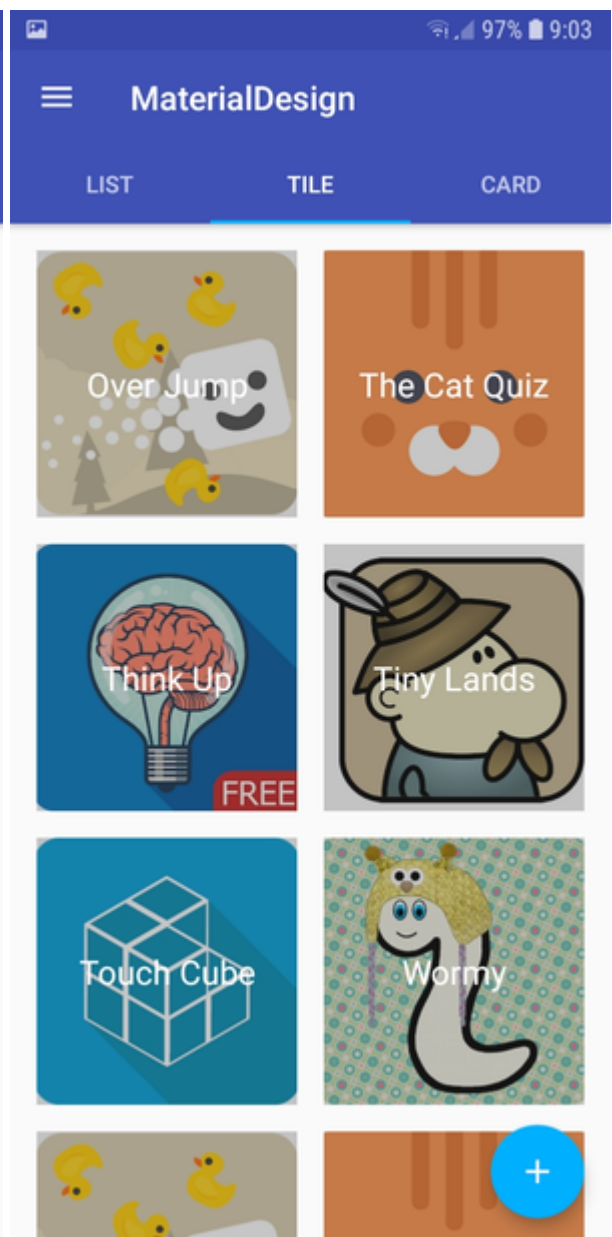
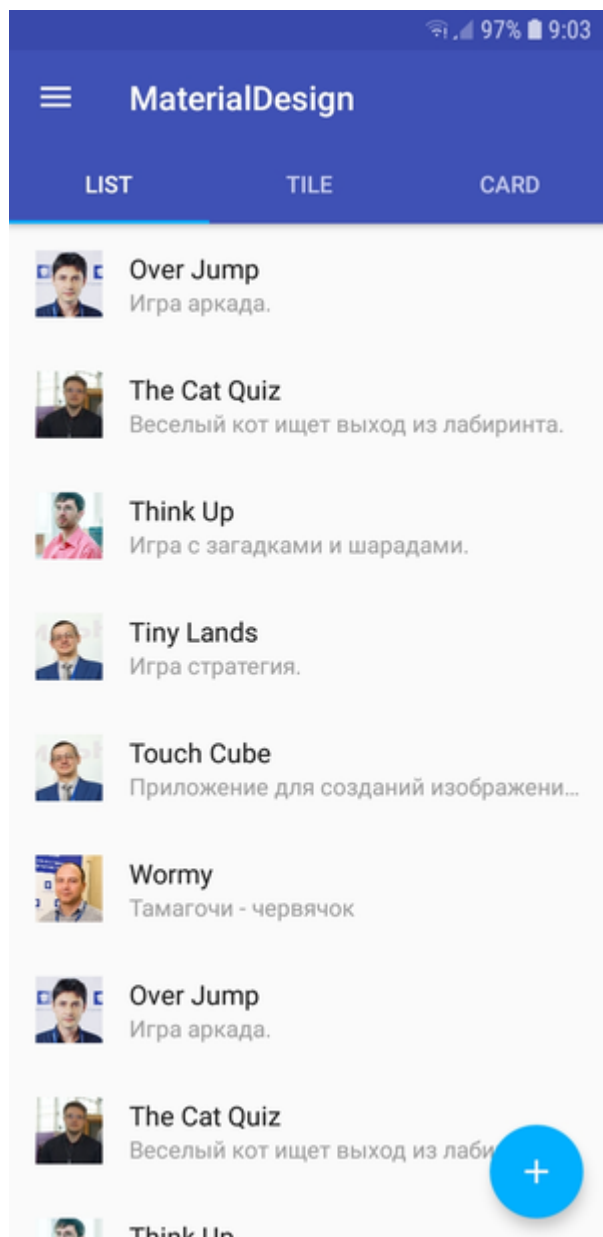
Создадим проект с Empty Activity.

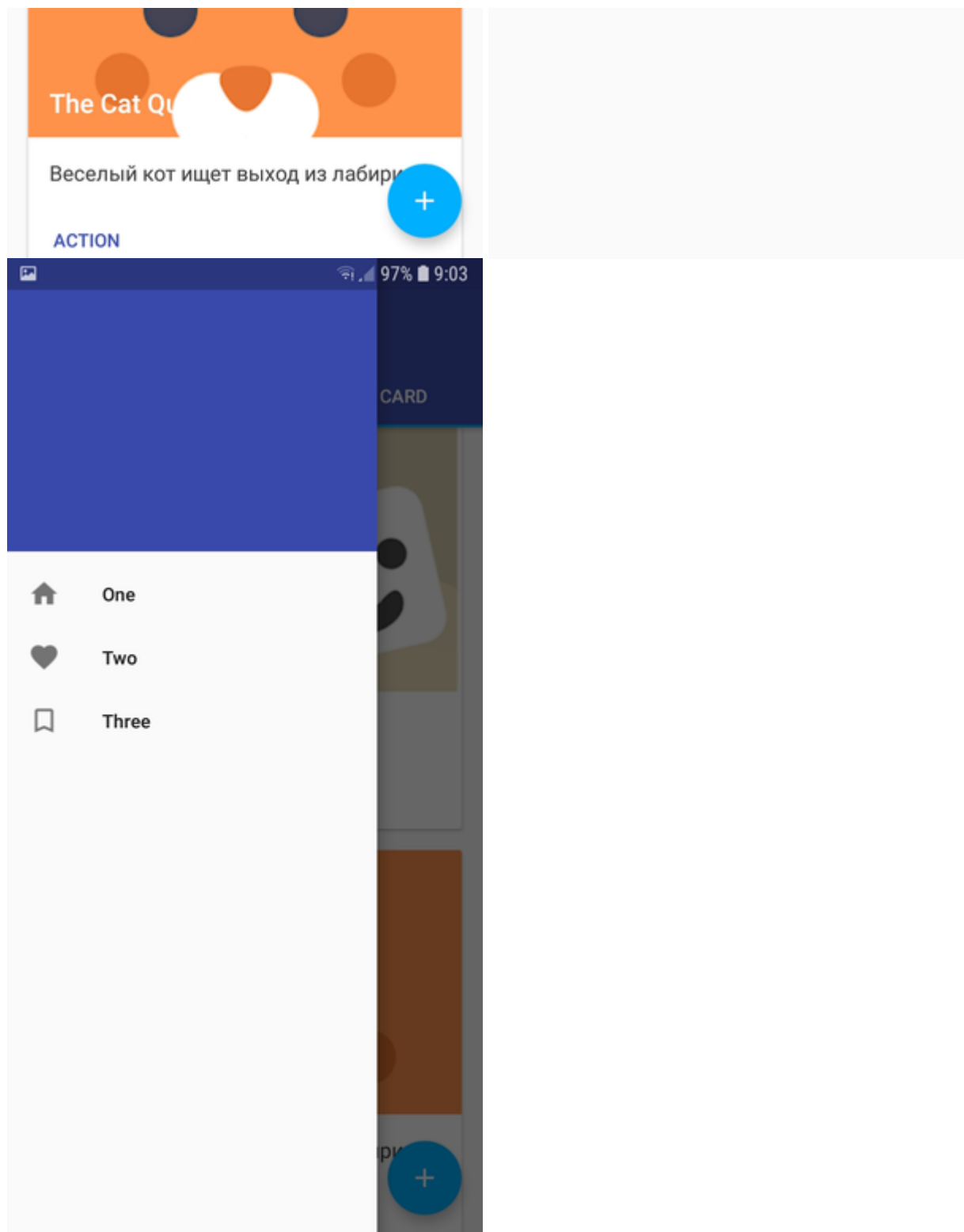
Для того чтобы подключить библиотеки элементов Material Design необходимо добавить в файл build.gradle (app) в раздел `dependencies` добавить следующие строки

```
implementation 'com.android.support:design:27.1.1'
implementation 'com.android.support:support-v4:27.1.1'
implementation 'com.android.support:support-annotations:27.1.1'
implementation 'com.android.support:cardview-v7:27.1.1'
implementation 'com.android.support:recyclerview-v7:27.1.1'
```

где 27 - в этом примере - `compileSdkVersion` . После синхронизации gradle файла проект готов к работе.

Используя библиотечные компоненты и рекомендации по построения интерфейса необходимо получить такое приложение.





Также можно скачать готовый пример реализации приложения с использованием элементов библиотек MD.

3. 5.6.3. Пример приложения с использованием элементов MD

Рассмотрим пример приложения с использованием библиотеки Material Design. Данный пример был реализован по мотивам статьи на CodeLabs. В нем представлены следующие разделы:

- цвета
- слои
- стили
- новые элементы GUI