Проект 1. Контроллер расходов

# Инфраструктура

## Выбор типа инфраструктуры

Первым делом, нам необходим сервер. Разумеется, речь идёт о «железном» сервере. Варианты с веб-хостингом, облачными провайдерами и т.п. не устраивают, поскольку нам требуется полный контроль над ОС, файлами, процессами, установленными компонентами и т.д.

### Существуют следующие варианты:

* Собственный сервер
  + с размещением в ДЦ (Collocation)
  + с размещением на своей территории
* IaaS (Infrastructure as a service):
  + Dedicated Server (Выделенный Сервер)
  + VDS (Virtual Dedicated Server – Виртуальный Выделенный Сервер, ∃ также название VPS – Virtual Private Server)

### Выбранное решение:

VDS

## Выбор VDS

Инструменты:

* поисковая система Google

Нужно найти несколько вариантов и выбрать из них наиболее подходящий. Для оценки подходящего решения существует несколько методик:

* Вариантные секторы
* Экспертная оценка

Для простоты мы ограничимся экспертной оценкой.

### Виртуализация

Все VDS-гипервизоры делятся на 2 больших категории:

* программная виртуализация (основная ОС выполняет все функции по виртуализации ресурсов; обычно строится на ядре Linux и вследствие позволяет разворачивать серверы только на дистрибутивах Linux)
* аппаратная виртуализация (вся виртуализация выполняется аппаратно, т.е. этот способ максимально приближен к выделенным серверам, в силу чего позволяет разворачивать гетерогенные ОС)

Существует несколько основных VDS-гипервизоров:

* Программная виртуализация
  + OpenVZ – пожалуй, лучшее средство виртуализации на базе Linux
  + FreeBSD Jail – средство виртуализации для FreeBSD
* Аппаратная виртуализация
  + KVM (Kernel-based Virtual Machine) – обеспечивает аппаратную виртуализацию для платформ x86 на базе спец. процессоров Intel и AMD
  + Hyper-V – система аппаратной виртуализации для платформы x64 от корпорации Microsoft
  + Xen – лёгкий гипервизор
  + VMware – система аппаратной виртуализации для платформ x64

### Результаты:

Было проанализировано 20 VDS-провайдеров из топа ответов поисковой системы Google по запросу «Виртуальный выделенный сервер»

1. <http://www.1gb.ru/price_vds_vz.php>
2. <http://firstvds.ru/products/vds_vps_hosting>
3. <http://selectel.ru/services/vds/>
4. <http://fastvps.ru/vds/>
5. <http://infobox.ru/vps/linux/>
6. <http://www.logol.ru/vps/>
7. <http://www.host96.ru/vps-vds>
8. <https://fornex.com/vps/>
9. <http://veeble.org/vps/openvz/>
10. <http://www.globoxhost.com/vds-servers.html>
11. <http://hosting.agava.ru/virt/vps/m_price.shtml>
12. <http://www.io-hosts.ru/vds/>
13. <http://www.r-tele.com/vds_cheap/>
14. <http://www.hyper-v-mart.com/Servers.aspx>
15. <http://masterhost.ru/service/hosting/vps>
16. <http://www.depocloud.ru/cart/>
17. <http://yeshost.ru/vds>
18. <http://www.demos-internet.ru/services.bbc.tarif.html>
19. <http://ispserver.com/ru/products/cloud_vps>
20. <http://gldn.ur.ru/services/hosting/vds/>

### Процесс отбора:

#### Первый этап: анализ сайта провайдера

На первом этапе проводилась эвристическая оценка: отсекались сайты, которые вызывали сомнения, не имели нормальных сравнительных таблиц или просто «не понравились».

В результате отбора остались 13 кандидатов:

1. firstvds
2. selectel
3. infobox
4. logol
5. host96
6. fornex
7. veeble
8. globoxhost
9. agava
10. r-telecom
11. 1gb
12. fastvps
13. io-hosts

#### Второй этап отбора: неограниченность трафика

Многие провайдеры накладывают ограничения на трафик:

* определённое соотношение входящего/исходящего трафика
* определённый процент российского/зарубежного трафика

Другие провайдеры не накладывают таких ограничений, однако обычно предъявляют вполне естественные требования на отсутствие «паразитного» трафика (интенсивная рассылка, DOS-атаки, torrent-клиенты и т.д.)

|  |  |
| --- | --- |
| **Ограничивают трафик** | **Не ограничивают трафик** |
| 1gb (исх/вх = 2/1, более 30% российского трафика) | firstvds |
| fastvps (1500 Гб в месяц) | selectel |
| io-hosts (исх/вх = 1/3) | infobox |
|  | logol |
|  | host96 |
|  | fornex |
|  | veeble |
|  | globoxhost |
|  | agava |
|  | r-telecom |

Итак, после второго этапа у нас остаётся 10 кандидатов.

#### Третий этап: цена 1 Мб памяти за простейший тариф

Теперь у каждого кандидата выберем простейший тариф и посчитаем, сколько просит провайдер за 1 Мб памяти.

Примечание: в случае расчёта в долларах тариф принимается равным 33р./$

|  |  |
| --- | --- |
| **Провайдер** | **Цена за 1 Мб памяти (руб.)** |
| globoxhost | 0,16 |
| veeble | 0,35 |
| firstvds | 0,37 |
| infobox | 0,38 |
| r-telecom | 0,56 |
| agava | 0,57 |
| fornex | 0,73 |
| logol | 0,98 |
| host96 | 0,98 |
| selectel | 1,00 |

Итак, на данном этапе была определена тройка лидеров:

* globoxhost
* veeble
* firstvds

#### Четвёртый этап: наличие панели управления

Разумеется, мы будем управлять через Telnet или SSH, но иметь веб-интерфейс для простейших операций тоже неплохо.

Наиболее популярны следующие решения

* ISPmanager Lite
* ISPmanager Pro
* Direct Admin
* cPanel

|  |  |  |
| --- | --- | --- |
| **Провайдер** | **Бесплатный ISPmanager Lite начиная с тарифа (р./мес)** | **Другие панели управления** |
| r-telecom | все тарифы | ISPmanager Pro (289 р./мес)  cPanel (612 р./мес) |
| firstvds | все тарифы |  |
| infobox | все тарифы |  |
| fornex | все тарифы |  |
| host96 | все тарифы |  |
| veeble | – | DirectAdmin (197 р./мес)  WHM/cPanel (495 р./мес)  Kloxo (free)  Webmin (free) |
| logol | 849 | ISPmanager Pro (0-349 р./мес)  cPanel (0-599 р./мес)  Parallels Plesk (0-1999 р./мес) |
| selectel | 1024 |  |
| agava | 1840 | DirectAdmin (245 р./мес)  WHM/cPanel (459 р./мес) |
| globoxhost | – | DirectAdmin (495 р./мес) |

Итак, среди тройки лидеров (globoxhost, veeble, firstvds) предпочтение было отдано последнему варианту, поскольку только «firstvds» имеет бесплатную панель управления на всех тарифах.

## Заключение договора и оплата услуг

Необходимо:

* банковская карта или счёт в платёжной системе (исп. Я.Деньги с привязанной картой Связной.Банк)

## Подключение к серверу

Инструменты:

* клиент для удалённого управления через SSH/Telnet
* клиент для передачи файлов; например:
  + WinSCP
  + SFTP-плагин для TotalCMD
  + FireFTP-плагин для Firefox

Мы использовали связку PuTTY + TotalCMD

# Конфигурирование сервера

## Добавление пользователя

Пароль суперпользователя был выслан в письме через некоторое время после оплаты услуг хостинга.

### Создание пользователя

Процедура добавления пользователя:

useradd mitrakov

passwd mitrakov

mkdir /home/mitrakov

chown mitrakov:mitrakov /home/mitrakov

### Установка Sudo

Далее следует установить программу sudo (от суперпользователя)

apt-get install sudo

Затем добавляем пользователя в sudoers:

sudo adduser mitrakov sudo

Теперь возможно выполнять команды от имени суперпользователя через Sudo.

Ссылка на статью: <http://askubuntu.com/questions/7477/how-can-i-add-a-new-user-as-sudoer-using-the-command-line>

## Установка Midnight Commander

### Инсталляция MC

sudo apt-get install mc

В процессе работы возникла небольшая проблема. При возвращении в консоль через CTRL-O и наборе команды вновь появлялся MC. В этом случае требовалось закрыть MC, и только затем вводить команду (либо вводить прямо внутри MC, что не очень удобно).

Решение было найдено на форуме: <http://www.bsdportal.ru/viewtopic.php?t=7300>. В статье говорится, что проблема не в MC, а в Shell. Поскольку MC был собран в /bin/sh, то он и возвращает консоль /bin/sh.

### Диагностика:

Для проверки текущей конфигурации окружения вводим команду:

env

Вывод:

$ env

TERM=xterm

SHELL=/bin/sh

SSH\_CLIENT=46.146.231.100 15802 22

SSH\_TTY=/dev/pts/0

USER=mitrakov

MAIL=/var/mail/mitrakov

PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games

PWD=/home/mitrakov

LANG=en\_US.UTF-8

SHLVL=1

HOME=/home/mitrakov

LOGNAME=mitrakov

SSH\_CONNECTION=46.146.231.100 15802 82.146.37.95 22

\_=/usr/bin/env

Очевидно, текущий Shell равен /bin/sh

### Изменение Shell

Для изменения текущего Shell вводим:

sudo chsh -s /bin/bash root

sudo chsh -s /bin/bash mitrakov

Далее обязательно следует перезайти в свою учётную запись и убедиться, что текущий Shell равен /bin/bash.

### Переустановка MC

Теперь нужно в конфигурации MC указать, какую Subshell использовать по команде CTRL-O. Однако есть решение «для чайников»: просто переустановить MC из нормального Shell. Введите от суперпользователя:

sudo apt-get autoremove mc

sudo apt-get install mc

Теперь сочетание клавиш CTRL-O должно работать нормально

### Изменение редактора в MC

Так оказалось, что по клавише F4 MC по умолчанию использует Nano. Для изменения параметров следует зайти в Options → Configuration и поставить флажки:

Use internal edit

Use internal view

Ссылка на статью: <http://askubuntu.com/questions/16776/how-to-switch-the-editor-in-mc-midnight-commander-from-nano-to-mcedit>

## Установка MySQL

### Инсталляция

Для закрепления тех. навыков было принято решение для 2-х проектов выбрать 2 популярные Open-Source СУБД:

* MySQL (Oracle Corporation, Sun Microsystems)
* PostgreSQL (EDB Group)

Конкретно для данного проекта была выбрана СУБД MySQL (насчёт принадлежности этого продукта к разряду Open-Source после покупки корпорацией Oracle спорить не будем).

MySQL была предустановлена в системе, однако было принято решение установить более новую версию:

sudo apt-get autoremove mysql-server-5.0

sudo apt-get autoremove mysql-client-5.0

sudo apt-get install mysql-server mysql-client libmysqlclient-dev

Во время установки был указан пароль для пользователя root.

Проверка версии клиента:

mysql --version

Вывод:

mitrakov@mitrakov-artem:~$ mysql --version

mysql Ver 14.14 Distrib 5.5.35, for debian-linux-gnu (x86\_64) using readline 6.2

Проверка версии сервера:

mysql –p

SELECT VERSION();

Вывод:

mysql> SELECT VERSION();

+------------------+

| VERSION() |

+------------------+

| 5.5.35-0+wheezy1 |

+------------------+

1 row in set (0.00 sec)

### Создание нового пользователя в MySQL

Входим (из-под суперпользователя):

mysql –u root –p

Теперь добавляем нового пользователя:

CREATE USER 'mitrakov'@'localhost' IDENTIFIED BY 'mypass';

CREATE USER 'mitrakov'@'%' IDENTIFIED BY 'mypass';

Первая строка создаёт пользователя для локального подключения, вторая – для удалённого (с любого хоста). В целях безопасности, вместо знака «%» рекомендуется ставить конкретный IP-адрес.

Далее присваиваем права пользователю:

GRANT ALL ON \*.\* TO 'mitrakov'@'localhost';

GRANT ALL ON \*.\* TO 'mitrakov'@'%';

Теперь просим MySQL сбросить привилегии (и заново перезагрузить конфигурацию пользователей):

FLUSH PRIVILEGES;

Дело сделано! Проверить текущую конфигурацию можно через запрос:

SELECT User, Host FROM mysql.user;

Вывод:

mysql> SELECT User, Host FROM mysql.user;

+------------------+-----------+

| User | Host |

+------------------+-----------+

| mitrakov | % |

| root | 127.0.0.1 |

| root | ::1 |

| debian-sys-maint | localhost |

| mitrakov | localhost |

| root | localhost |

+------------------+-----------+

6 rows in set (0.00 sec)

«127.0.0.1» и «::1» – это адреса локального хоста (для IPv4 и IPv6 соответственно). Удалить запись можно через запрос:

DROP USER 'mitrakov'@'localhost';

После всех изменений следует не забыть сбросить привилегии.

### Настройка bind-адреса

Если требуется тестирование приложения с другого хоста, то требуется настроить bind-address (текущий адрес равен ‘localhost’).

Измените bind-address в файле **/etc/mysql/my.cnf** (от суперпользователя):

# Instead of skip-networking the default is now to listen only on

# localhost which is more compatible and is not less secure.

bind-address = 82.146.37.95

Также в целях безопасности там же можно поменять порт подключения. Далее следует перезагрузить mysqld:

sudo service mysql restart

Внимание! Настройка bind-адреса не влияет на обычные подключения к БД с удалённого хоста. Это требуется, только если само приложение будет запускаться с удалённого хоста. Поскольку в нашем проекте приложение будет подключаться локально, данная настройка не была изменена.

## Установка Tomcat 7.0

### Инсталляция

Текущая версия Tomcat имеет номер 8.03, однако мы для простоты установим версию 7.0 из центрального репозитория.

sudo apt-get install tomcat7

### Изменение порта

По умолчанию Tomcat использует порт 8080. Для изменения порта изменяем файл **/etc/tomcat7/server.xml**:

<Connector port="80" protocol="HTTP/1.1"

Далее находим файл **/etc/default/tomcat7**. Там находим строку:

#AUTHBIND=no

И меняем её на

AUTHBIND=yes

Последнее условие требуется для того, чтобы разрешить приложению Tomcat привязку к системным портам (номера которых 0-1023).

Ссылка на статью: <http://stackoverflow.com/questions/4756039/how-to-change-the-port-of-tomcat-from-8080-to-80>

Теперь можно зайти на хост через браузер:

http://82.146.37.95/

Вывод:

It works!

This is the default web page for this server.

The web server software is running but no content has been added, yet.

Нетрудно заметить[[1]](#footnote-1), что вывод файла не соответствует стартовой HTML-странице Томката, расположенной по адресу **/var/lib/tomcat7/webapps/ROOT.index.html**. Дело в том, что порт уже занят установленным по умолчанию веб-сервером Apache2.

### Удаление Apache2 Server

Сначала останавливаем сервер:

sudo service apache2 stop

Теперь удаляем сам сервер. Этот шаг необязательный, но всё же лучше его выполнить, чтобы, например, при перезагрузке машины он не стартовал заново:

sudo apt-get autoremove apache2.2

Перезапускаем Tomcat:

sudo service tomcat7 restart

И заходим на стартовую страницу:

http://82.146.37.95/

Вывод

It works !

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

This is the default Tomcat home page. It can be found on the local filesystem at: /var/lib/tomcat7/webapps/ROOT/index.html

Tomcat7 veterans might be pleased to learn that this system instance of Tomcat is installed with CATALINA\_HOME in /usr/share/tomcat7 and CATALINA\_BASE in /var/lib/tomcat7, following the rules from /usr/share/doc/tomcat7-common/RUNNING.txt.gz.

You might consider installing the following packages, if you haven't already done so:

tomcat7-docs: This package installs a web application that allows to browse the Tomcat 7 documentation locally. Once installed, you can access it by clicking here.

tomcat7-examples: This package installs a web application that allows to access the Tomcat 7 Servlet and JSP examples. Once installed, you can access it by clicking here.

tomcat7-admin: This package installs two web applications that can help managing this Tomcat instance. Once installed, you can access the manager webapp and the host-manager webapp.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager-gui". The host-manager webapp is restricted to users with role "admin-gui". Users are defined in /etc/tomcat7/tomcat-users.xml.

## Установка Redis и Curl

Для установки СУБД Redis достаточно набрать команду:

sudo apt-get install redis-server

Для установки утилиты Curl достаточно набрать команду:

sudo apt-get install curl

# Проектирование сервера

## Проектирование БД

Инструменты:

* MySQL Workbench (6.0.8.11354 Community Edition)

База данных была создана локально:

mysql –u mitrakov –p

...

CREATE DATABASE varlam DEFAULT CHARACTER SET utf8 ;

MySQL Workbench позволяет легко подключиться к удалённой БД, используя имя хоста, номер порта, имя пользователя и пароль.

Схема была создана с помощью визуального интерфейса, ниже представлен полный её скрипт:

-- MySQL dump 10.13 Distrib 5.5.35, for debian-linux-gnu (x86\_64)

--

-- Host: localhost Database: varlam

-- ------------------------------------------------------

-- Server version 5.5.35-0+wheezy1

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `category`

--

DROP TABLE IF EXISTS `category`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `category` (

`category\_id` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` bigint(20) unsigned NOT NULL COMMENT 'Reference to a user',

`parent\_id` bigint(20) unsigned DEFAULT NULL COMMENT 'Reference to itself (subcategory); all root nodes must be NULL',

`name` varchar(64) NOT NULL COMMENT 'UNIQUE name of a category',

PRIMARY KEY (`category\_id`),

UNIQUE KEY `name\_UNIQUE` (`user\_id`,`name`),

KEY `fk\_category\_this\_idx` (`parent\_id`),

KEY `ind\_user\_parent` (`user\_id`,`parent\_id`) COMMENT 'Index to search all root categories by user\_id',

CONSTRAINT `fk\_category\_this` FOREIGN KEY (`parent\_id`) REFERENCES `category` (`category\_id`) ON UPDATE CASCADE,

CONSTRAINT `fk\_category\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`user\_id`) ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='A group of related items (food, entertainment, transport etc.)';

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `item`

--

DROP TABLE IF EXISTS `item`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `item` (

`item\_id` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` bigint(20) unsigned NOT NULL COMMENT 'Reference to a user',

`category\_id` bigint(20) unsigned NOT NULL COMMENT 'Reference to a category (food, fines, housing & utilities)',

`name` varchar(64) NOT NULL COMMENT 'Name of an item (must be UNIQUE)',

PRIMARY KEY (`item\_id`),

UNIQUE KEY `name\_UNIQUE` (`user\_id`,`name`),

KEY `fk\_item\_category\_idx` (`category\_id`),

CONSTRAINT `fk\_item\_category` FOREIGN KEY (`category\_id`) REFERENCES `category` (`category\_id`) ON UPDATE CASCADE,

CONSTRAINT `fk\_item\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`user\_id`) ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='Items (goods/services and so on)';

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `operation`

--

DROP TABLE IF EXISTS `operation`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `operation` (

`operation\_id` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` bigint(20) unsigned NOT NULL COMMENT 'Reference to a user',

`time` timestamp DEFAULT CURRENT\_TIMESTAMP,

`item\_id` bigint(20) unsigned NOT NULL COMMENT 'Reference to an item (goods/services, salary, debts and so on)',

`person\_id` bigint(20) unsigned DEFAULT NULL COMMENT 'Reference to a person that corresponds to an operation (e.g. debts). May be NULL',

`summa` int(11) NOT NULL COMMENT 'Operation summa (negative values for charges and positive ones for income)',

PRIMARY KEY (`operation\_id`),

KEY `fk\_operation\_item\_idx` (`item\_id`),

KEY `fk\_operation\_person\_idx` (`person\_id`),

KEY `fk\_operation\_user\_idx` (`user\_id`),

KEY `operation\_timestamp` (`time`),

CONSTRAINT `fk\_operation\_item` FOREIGN KEY (`item\_id`) REFERENCES `item` (`item\_id`) ON UPDATE CASCADE,

CONSTRAINT `fk\_operation\_person` FOREIGN KEY (`person\_id`) REFERENCES `person` (`person\_id`) ON UPDATE CASCADE,

CONSTRAINT `fk\_operation\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`user\_id`) ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=2 DEFAULT CHARSET=utf8 COMMENT='Main table';

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `person`

--

DROP TABLE IF EXISTS `person`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `person` (

`person\_id` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

`user\_id` bigint(20) unsigned NOT NULL COMMENT 'Reference to a user',

`name` varchar(64) NOT NULL COMMENT 'Name or simple description of a person (must be UNIQUE)',

PRIMARY KEY (`person\_id`),

UNIQUE KEY `name\_UNIQUE` (`user\_id`,`name`),

CONSTRAINT `fk\_person\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`user\_id`) ON UPDATE CASCADE

) ENGINE=InnoDB AUTO\_INCREMENT=38 DEFAULT CHARSET=utf8 COMMENT='People that might be related to specific operations (debts and so on)';

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `user`

--

DROP TABLE IF EXISTS `user`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!40101 SET character\_set\_client = utf8 \*/;

CREATE TABLE `user` (

`user\_id` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

`username` varchar(64) NOT NULL COMMENT 'The UNIQUE username',

PRIMARY KEY (`user\_id`),

UNIQUE KEY `username\_UNIQUE` (`username`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8 COMMENT='Dictionary of the users';

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;

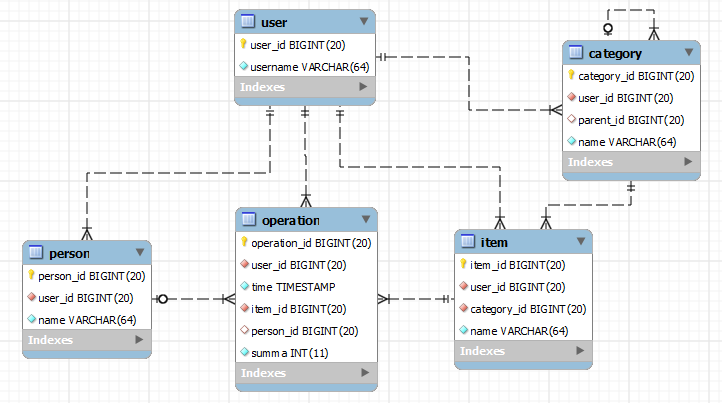
-- Dump completed on 2014-03-10 21:51:38

Скрипт был создан командой **mysqldump**:

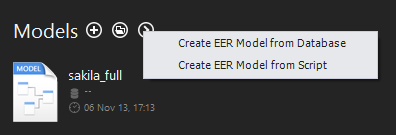
mysqldump -u mitrakov -p --no-data varlam > script.sql

Ссылка: <http://stackoverflow.com/questions/6175473/mysql-export-schema-without-data>

Ниже показана схема в графическом виде:



Для создания диаграммы применялся MySQL Workbench инструмент «Create EER Model from Database».



Общая идея такова:

* Таблица **user** является ассоциативным массивом {id: логин}
* Таблица **operation** является главной: в ней отражается факт покупки товара/услуги. При этом выставляется время покупки, сумма покупки и ссылка на товарную позицию (**item\_id**)
* Таблица **item** содержит имена всех товарных позиций и ссылки на категорию
* Таблица **category** служит для обобщения товарных позиций. Например, позиции «Ремонт авто», «Шиномонтаж», «АЗС», «Жидкость для стёкол» можно объединить в 1 категорию «Авто». Также таблица имеет поле **parent\_id** – ссылку на родительскую категорию.
* Таблица **person** содержит сведения о людях, причастных к операции (напр. «Дал в долг»)

Несколько замечаний:

1. Атрибут **user\_id** имеется в каждой таблице. Это значит, что имена товаров, категорий и персон будут своими для каждого пользователя.
2. Для времени (**operation.time**) был выбран тип TIMESTAMP, поскольку в MySQL только для него существует default-константа CURRENT\_TIMESTAMP. Таким образом, время операции вставится автоматически
3. Для всех внешних ключей применялись следующие атрибуты триггеров:
   * on update: cascade
   * on delete: restrict
4. Для таблиц, в которых есть уникальное имя (**category**, **item**, **person**) при создании внешнего ключа на **user** автоматически создаётся индекс (назовём его X). В нашем случае мы специально отдельно создаём UNIQUE-индекс по полям {**user\_id**, **name**}. Делается это для того, чтобы для каждого пользователя все имена были уникальными. Поскольку **user\_id** в индексе стоит первым, то X можно удалить.

## Разработка тестового сервлета

### Стек технологий

* Платформа: Java
* Язык программирования: Java/Scala
* Фреймворк: Spring Framework
* Система сборки: Maven
* Сервер приложений: Tomcat
* ORM: Hibernate
* Дополнительно: EJB3, JPA, JTA

### Создание Maven-проекта

Инструменты:

* JDK 1.7.0\_45
* Apache Maven 3.1.1
* IntelliJ IDEA 13.0.2 (Build 133.696) Community Edition
* Scala Plugin for IntelliJ IDEA

В качестве системы сборки можно порекомендовать:

* Maven
* Ant
* SBT (Simple Build Tool)
* Gradle

Для Scala-проектов рекомендуется применять SBT, однако в нашем проекте применялся Maven.

Создание проекта: New Project → Maven[[2]](#footnote-2)

Далее следует указать параметры проекта (имя, каталог, SDK и т.д.).

В параметрах Maven было указано:

* GroupId: ru.tomtrix.fvds
* ArtifactId: varlam
* Version: 1.0.0

Для задания шаблона применялся archetype: **org.scala-tools.archetypes:scala-archetype-simple**, однако много лишних настроек из шаблона было удалено. После создания проекта следует подождать, пока Maven не загрузит шаблон из archetype.

#### Изменение проекта

1. pom.xml
   1. изменено:
      1. <inceptionYear>2014</inceptionYear>
      2. <scala.version>2.10.3</scala.version>
   2. добавлено:
      1. <name>Varlam Project</name>
      2. <url>http://varlam.com</url>
      3. В **maven-scala-plugin** добавлено: <version>2.15.2</version>
      4. В секцию properties:   
         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   3. удалено:
      1. dependency: junit
      2. dependency: org.spec
      3. plugin: maven-eclipse-plugin
      4. секция reporting
2. удалено из проекта содержимое каталога test/scala/ru.tomtrix.fvds (тесты будем писать позже)
3. файл Application.scala переименован в VarlamStart.scala и слегка переписан:

**package** ru.tomtrix.fvds

**object** VarlamStart **extends** *App* {

println("Hello, world!")

}

1. Также чуть позже в pom.xml в секцию **configuration** Scala-плагина было добавлено следующее:

<**args**>

<**arg**>-target:jvm-1.6</**arg**>

<**arg**>-unchecked</**arg**>

<**arg**>-deprecation</**arg**>

<**arg**>-feature</**arg**>

</**args**>

<**jvmArgs**>

<**jvmArg**>-Xms128m</**jvmArg**>

<**jvmArg**>-Xmx1024m</**jvmArg**>

</**jvmArgs**>

**target-jvm** указывает целевую версию JVM (поднятие версии до 1.7 вызвало ошибку версии классов, поэтому 1.6 оказалось наилучшим вариантом). Следующие 3 параметра: доп. опции компилятора Scala, а **jvmArgs** увеличивает размер кучи для ускорения работы компилятора[[3]](#footnote-3).

Ссылка по теме: <http://scala-tools.org/mvnsites/maven-scala-plugin/example_compile.html>

Заглушка проекта готова. Для запуска нужно в окне проекта найти файл **VarlamStart.scala** и в его контекстном меню выбрать **Run ‘VarlamStart’**.

Ниже представлен код pom.xml:

<**project** *xmlns*="http://maven.apache.org/POM/4.0.0" *xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xsi*:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<**modelVersion**>4.0.0</**modelVersion**>

<**groupId**>ru.tomtrix.fvds</**groupId**>

<**artifactId**>varlam</**artifactId**>

<**version**>1.0.0</**version**>

<**name**>Varlam Project</**name**>

<**url**>http://varlam.com</**url**>

<**inceptionYear**>2014</**inceptionYear**>

<**properties**>

<**scala**.version>2.10.3</**scala**.version>

</**properties**>

<**repositories**>

<**repository**>

<**id**>scala-tools.org</**id**>

<**name**>Scala-Tools Maven2 Repository</**name**>

<**url**>http://scala-tools.org/repo-releases</**url**>

</**repository**>

</**repositories**>

<**pluginRepositories**>

<**pluginRepository**>

<**id**>scala-tools.org</**id**>

<**name**>Scala-Tools Maven2 Repository</**name**>

<**url**>http://scala-tools.org/repo-releases</**url**>

</**pluginRepository**>

</**pluginRepositories**>

<**dependencies**>

<!--Scala Library-->

<**dependency**>

<**groupId**>org.scala-lang</**groupId**>

<**artifactId**>scala-library</**artifactId**>

<**version**>*${scala.version}*</**version**>

</**dependency**>

</**dependencies**>

<**build**>

<**sourceDirectory**>src/main/scala</**sourceDirectory**>

<**testSourceDirectory**>src/test/scala</**testSourceDirectory**>

<**finalName**>*${artifactId}*</**finalName**>

<**plugins**>

<!--Scala Build Plugin-->

<**plugin**>

<**groupId**>org.scala-tools</**groupId**>

<**artifactId**>maven-scala-plugin</**artifactId**>

<**executions**>

<**execution**>

<**goals**>

<**goal**>compile</**goal**>

<**goal**>testCompile</**goal**>

</**goals**>

</**execution**>

</**executions**>

<**configuration**>

<**scalaVersion**>*${scala.version}*</**scalaVersion**>

<**args**>

<**arg**>-target:jvm-1.5</**arg**>

</**args**>

</**configuration**>

</**plugin**>

</**plugins**>

</**build**>

</**project**>

### DataBase Reverse-Engineering

В качестве ORM использовался проект Hibernate. Данная ORM предоставляет весьма удобный инструмент для реверс-инжиниринга: Hibernate-Tools (<http://hibernate.org/tools>). Этот инструмент поставляется в 2-х формах:

* плагин для IDE Eclipse
* Ant-Task

Ant-Task был использован сообществом Codehaus как основа для разработки Maven-плагина **hibernate3-maven-plugin**. Этот плагин позволяет решить все те задачи, которые решает стандартный Hibernate-Tools.

В нашем случае плагин настроен для решения следующей задачи: требуется подключиться к существующей БД и на основе метаданных сгенерировать Entity-классы (стандарта EJB3). При этом требуется включить генерацию в процесс сборки, чтобы можно было свободно редактировать схему БД и не бояться за актуальность Entity-классов. Иными словами, чтобы при каждой сборке перегенерировались Entity-классы.

Ниже представлен код плагина (требуется включить в секцию <build><plugins>)

<!--Hibernate Reverse-Engineer-Plugin-->

<**plugin**>

<**groupId**>org.codehaus.mojo</**groupId**>

<**artifactId**>hibernate3-maven-plugin</**artifactId**>

<**version**>2.2</**version**>

<**executions**>

<**execution**>

<**id**>hbm2cfgxml</**id**>

<**phase**>generate-resources</**phase**>

<**goals**>

<**goal**>hbm2cfgxml</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2cfgxml</**name**>

<**implementation**>jdbcconfiguration</**implementation**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

</**componentProperties**>

</**configuration**>

</**execution**>

<**execution**>

<**id**>hbm2hbmxml</**id**>

<**phase**>generate-resources</**phase**>

<**goals**>

<**goal**>hbm2hbmxml</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2hbmxml</**name**>

<**outputDirectory**>target/classes</**outputDirectory**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

</**componentProperties**>

</**configuration**>

</**execution**>

<**execution**>

<**id**>hbm2java</**id**>

<**phase**>generate-sources</**phase**>

<**goals**>

<**goal**>hbm2java</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2java</**name**>

<**implementation**>configuration</**implementation**>

<**outputDirectory**>src/main/scala</**outputDirectory**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

<**configurationfile**>target/hibernate3/generated-mappings/hibernate.cfg.xml</**configurationfile**>

<**jdk5**>true</**jdk5**>

<**ejb3**>true</**ejb3**>

</**componentProperties**>

</**configuration**>

</**execution**>

</**executions**>

<**dependencies**>

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.29</**version**>

</**dependency**>

</**dependencies**>

</**plugin**>

Настройки содержат 3 задачи (**hbm2cfgxml**, **hbm2hbmxml**, **hbm2java**) и 1 зависимость (**MySQL-JDBC-Driver**).

1. **hbm2cfgxml** – применяется для генерации файла «hibernate.cfg.xml», который будет использоваться в др. задачах. В общем случае проект Hibernate требует наличия этого файла для работы с Session, однако в нашем случае мы имеем дело с проектом JPA, поэтому в runtime этот файл использоваться не будет
2. **hbm2hbmxml** – применяется для создания Mapping-XML-файлов, на базе которых будут созданы Java-классы. Эти XML-файлы также в runtime использоваться не будут
3. **hbm2java** – задача, которая на базе файла конфигурации («hibernate.cfg.xml») и Mapping-файлов будет генерировать Java-классы сущностей. Параметр <ejb3> должен быть обязательно установлен в **true**.

Ссылка на статью: <http://stackoverflow.com/questions/2843949/how-to-configure-hibernate-tools-with-maven-to-generate-hibernate-cfg-xml-hbm>

Также следует упомянуть о задаче **hbm2dao**, которая позволяет создавать DAO для каждого полученного Entity-класса. Её вполне можно включить в цикл работы плагина, однако мы этого делать не будем, а будем писать код DAO вручную. Ниже на всякий случай приведён код подзадачи:

<**execution**>

<**id**>hbm2dao</**id**>

<**phase**>generate-sources</**phase**>

<**goals**>

<**goal**>hbm2dao</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2dao</**name**>

<**implementation**>configuration</**implementation**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

<**configurationfile**>target/hibernate3/generated-mappings/hibernate.cfg.xml</**configurationfile**>

<**jdk5**>true</**jdk5**>

<ejb3>true</ejb3>

</**componentProperties**>

</**configuration**>

</**execution**>

Генерация DAO была исключена из сборки в связи с некоторыми проблемами, о которых будет рассказано позже.

Последний элемент: создание файла с описанием свойств подключения к базе данных (**src/main/resources/database.properties**):

hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

hibernate.connection.driver\_class = com.mysql.jdbc.Driver

hibernate.connection.url = jdbc:mysql://82.146.37.95:3306/varlam

hibernate.connection.username = mitrakov

hibernate.connection.password= 12345

Обратите внимание! Здесь указан внешний URL (82.146.37.95).

Для сборки следует запустить любую команду из жизненного цикла Maven, которая включает фазу **generate-sources**, напр.

mvn clean package

В целом, удобнее это делать не из консоли, а напрямую из IntelliJ IDEA.

Примечание: на данном этапе сгенерированные файлы будут давать ошибку компиляции из-за отсутствия соотв. библиотек.

### Внедрение JPA через Spring Framework

Проект Varlam использует технологию JPA, реализованную в Hibernate. Для этого нужно добавить в POM соответствующие зависимости:

<!--Hibernate Java Persistence API-->

<**dependency**>

<**groupId**>org.hibernate</**groupId**>

<**artifactId**>hibernate-entitymanager</**artifactId**>

<**version**>4.3.1.Final</**version**>

</**dependency**>

<!-- MySQL connector -->

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.29</**version**>

</**dependency**>

Следует обратить внимание: в зависимостях проекта и в зависимостях плагина **hibernate3-maven-plugin** применяется один и тот же **MySQL connector**.

Теперь можно попробовать пересобрать проект и убедиться в отсутствии ошибок компиляции.

JPA-проект вместо стандартного конфигурационного файла hibernate.cfg.xml использует Persistence-Unit, описываемый в файле **src/main/resources/META-INF/persistence.xml**:

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**persistence** *version*="1.0" *xmlns*="http://java.sun.com/xml/ns/persistence"

*xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xsi*:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">

<**persistence-unit** *name*="trixik" *transaction-type*="RESOURCE\_LOCAL">

<**provider**>org.hibernate.jpa.HibernatePersistenceProvider</**provider**>

<**properties**>

<**property** *name*="hibernate.dialect" *value*="org.hibernate.dialect.MySQL5Dialect"/>

<**property** *name*="hibernate.connection.driver\_class" *value*="com.mysql.jdbc.Driver"/>

<**property** *name*="hibernate.show\_sql" *value*="true"/>

<**property** *name*="hibernate.connection.username" *value*="mitrakov"/>

<**property** *name*="hibernate.connection.password" *value*="12345"/>

<**property** *name*="hibernate.connection.url" *value*="jdbc:mysql://127.0.0.1:3306/varlam"/>

<**property** *name*="hibernate.connection.characterEncoding" *value*="UTF-8"/>

</**properties**>

</**persistence-unit**>

</**persistence**>

Обратите внимание! Казалось бы, эти настройки копируют уже имеющийся **database.properties**, однако это не так. Здесь URL установлен в 127.0.0.1 (‘localhost’), поскольку подключение будет производиться локально. В том же случае мы специально указали внешний адрес (82.146.37.95), чтобы **hibernate3-maven-plugin** мог корректно работать с произвольного хоста.

Следующий шаг: подключение **Spring Framework**:

<!--Spring Context Management-->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-context</**artifactId**>

<**version**>4.0.1.RELEASE</**version**>

</**dependency**>

<!--Spring Entity & Transaction Management-->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-orm</**artifactId**>

<**version**>4.0.1.RELEASE</**version**>

</**dependency**>

Одна библиотека позволяет работать со Spring-контекстом, другая – с ORM (в т.ч. Hibernate). Spring нам поможет с инициализацией EntityManagerFactory и управления транзакциями.

Для создания контекста прописываем XML-файл (**src/main/resources/trix.xml**):

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**beans** *xmlns*="http://www.springframework.org/schema/beans"

*xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xmlns*:context="http://www.springframework.org/schema/context" *xmlns*:tx="http://www.springframework.org/schema/tx"

*xsi*:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<!--Force Spring to inject dependencies via annotations-->

<**context:annotation-config** />

<!--Inject the Transaction Manager via @Transactional annotation-->

<**tx:annotation-driven** *transaction-manager*="transactionManager" />

<!--Spring Entity Manager Factory built on META-INF/persistence.xml-->

<**bean** *id*="entityManagerFactory" *class*="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">

<**property** *name*="persistenceUnitName" *value*="trixik" />

</**bean**>

<!--Spring Transaction Manager-->

<**bean** *id*="transactionManager" *class*="org.springframework.orm.jpa.JpaTransactionManager">

<**property** *name*="entityManagerFactory" *ref*="entityManagerFactory" />

</**bean**>

<!--My DAO-->

<**bean** *id*="dao" *class*="ru.tomtrix.fvds.DAO" />

</**beans**>

1. **context:annotation-config** позволяет Спрингу управлять зависимостями в коде через аннотации (в нашем случае через @Transactional).
2. **tx:annotation-driven** позволяет внедрить Менеджера Транзакций в любой участок кода, помеченного как @Transactional (метод или класс целиком). Менеджер Транзакций создаётся ниже.
3. **bean id="entityManagerFactory"** создаёт Фабрику Менеджера Сущностей, которая считывает контекст из модуля **trixik**, описанного в **persistence.xml**.
4. **bean id="transactionManager"** создаёт Менеджера Транзакций
5. **bean id="dao"** создаёт экземпляр DAO, который мы опишем в след. разделе.

### Создание универсального DAO

Ниже представлен код DAO.scala.

**package** ru.tomtrix.fvds

**import** *javax.persistence.\_*

*import org.springframework.transaction.annotation.Transactional*

/\*\*

\* Created by Mitrakov

\*/

@Transactional

**class** DAO {

@PersistenceContext private **val** entityManager: EntityManager = *null*

**def** **persist**(*transientInstance*: AnyRef) {

println("persisting Operation instance")

**try** {

entityManager.persist(transientInstance)

println("persist successful")

}

**catch** {

**case** *re*: RuntimeException =>

re.printStackTrace()

**throw** re

}

}

**def** **remove**(*persistentInstance*: AnyRef) {

println("removing Operation instance")

**try** {

entityManager.remove(entityManager.merge(persistentInstance))

println("remove successful")

}

**catch** {

**case** *re*: RuntimeException =>

re.printStackTrace()

**throw** re

}

}

**def** **merge**[T](*detachedInstance*: T): T = {

println("merging Operation instance")

**try** {

**val** result: T = entityManager.merge(detachedInstance)

println("merge successful")

result

}

**catch** {

**case** *re*: RuntimeException =>

re.printStackTrace()

**throw** re

}

}

**def** **findById**[T](*id*: Long, *clazz*: Class[T]): T = {

println("getting Operation instance with id: " + id)

**try** {

**val** instance: T = entityManager.find(clazz, id)

println("get successful")

instance

}

**catch** {

**case** *re*: RuntimeException =>

re.printStackTrace()

**throw** re

}

}

}

Код был воссоздан на базе DAO, сгенерированного плагином hibernate3-maven-plugin для некоторой сущности (подзадача **hbm2dao**). Далее класс будет несколько дополнен.

Основные отличия от автоматически генерируемого DAO-файла:

* Класс написан на Scala
* Все методы параметризованы типом T, что позволяет иметь ровно один DAO для любого Entity-класса. Напомним, что Hibernate-Tools для N сущностей генерирует N практически одинаковых DAO.
* Класс содержит аннотацию @Transactional, через которую Spring Framework внедряет Менеджера Транзакций.
* Метод remove переписан так, что сначала выполняется merge, а уже потом непосредственно remove. Если сущность была получена в ходе разных транзакций, то эта техника позволяет избежать ошибки:

java.lang.IllegalArgumentException: Removing a detached instance

Замечание 1: аннотация @Transactional должна принадлежать Spring Framework. Следующее определение будет неверным:

import javax.transaction.Transactional

Замечание 2: **entityManager** также будет внедрён с помощью Spring Framework через аннотацию @PersistenceContext.

Замечание 3: пока что класс пишет в stdout. Чуть позже мы настроим нормальный логгер.

### Создание тестового web-проекта

Для веб-коммуникации проект будет использовать технологию Java Servlet 3.1. Для создания веб-проекта требуется:

1) добавить в POM директиву упаковки war:

<packaging>war</packaging>

2) создать файл **/src/main/webapp/WEB-INF/web.xml**:

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**web-app**>

<**display-name**>Trix Web Application</**display-name**>

<**servlet**>

<**servlet-name**>VarlamStart</**servlet-name**>

<**servlet-class**>ru.tomtrix.fvds.VarlamStart</**servlet-class**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>VarlamStart</**servlet-name**>

<**url-pattern**>/start</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

Секция Servlet специфицирует имя сервлета и его класс, а секция Servlet-Mapping: соответствующий URL.

3) Добавляем в проект зависимость:

<!-- Servlet-API with PROVIDED scope -->

<**dependency**>

<**groupId**>javax.servlet</**groupId**>

<**artifactId**>javax.servlet-api</**artifactId**>

<**version**>3.0.1</**version**>

<**scope**>provided</**scope**>

</**dependency**>

Обязательно указываем Scope provided! Это значит, что зависимость нужна на этапе компиляции, но она должна быть исключена из финальной сборки, поскольку Servlet-API уже предоставляется контейнером. В противном случае Tomcat будет выдавать ошибку о конфликте классов.

4) В POM-файле в секции Build следует добавить:

<finalName>${artifactId}</finalName>

Это заставит Maven назвать сборку «varlam.war» вместо стандартного «varlam-1.0.0.war»

5) Пишем код (src/main/scala/ru/tomtrix/fvds/VarlamStart.scala)

**package** ru.tomtrix.fvds

**import** *scala.compat.Platform*

*import javax.servlet.http.\_*

*import javax.persistence.EntityManagerFactory*

*import org.springframework.context.support.ClassPathXmlApplicationContext*

*import ru.tomtrix.fvds.db.Person*

**class** VarlamStart **extends** *HttpServlet* {

**val** factory = **new** ClassPathXmlApplicationContext("trix.xml"); //Spring XML

**val** emf = factory.getBean("entityManagerFactory").asInstanceOf[EntityManagerFactory]

**val** dao = factory.getBean("dao").asInstanceOf[DAO]

override **def** **doGet**(*req*: HttpServletRequest, *resp*: HttpServletResponse): Unit = {

**val** writer = resp.getWriter

**try** {

dao persist **new** Person("Васёк" + Platform.currentTime)

**val** v = dao findById(*1L*, classOf[Person])

v setName("Tommy" + Platform.currentTime)

dao merge v

writer println "All right!"

}

**catch** {**case** *x*: Throwable => writer println x}

writer.close()

}

override **def** **destroy**(): Unit = {

emf.close()

*super*.destroy()

}

}

* factory получает ссылку на Context Спринга
* emf – ссылка на Фабрику Менеджера Сущностей. По факту, класс DAO уже содержит ссылку на Менеджера Сущностей; emf требуется только для того, чтобы корректно завершить работу сервлета в методе destroy()
* dao – экземпляр нашего DAO-класса

На этом создание тестового сервлета завершено.

### Deploying

Инструменты:

* SFTP или SCP-клиент для копирования файлов на сервер

Сначала выполняем сборку с помощью Maven (фаза package или выше) и получаем готовый WAR-архив в папке target. При сборке также будут перегенерированы все Entity-классы.

Теперь следует скопировать архив на сервер в каталог **/var/lib/tomcat7/webapps**. В нашем случае сначала архив был загружен в домашнюю директорию, а уже затем скопирован от имени суперпользователя:

sudo cp ~/varlam.war /var/lib/tomcat7/webapps

После копирования Tomcat через некоторое время сам запустит web-приложение (так называемый hot-deploying)

Замечание 1. Пока что мы копируем файл вручную, позже этот процесс автоматизируем

Замечание 2. Все возможные ошибки будут записываться в лог Томката: **/var/log/tomcat**

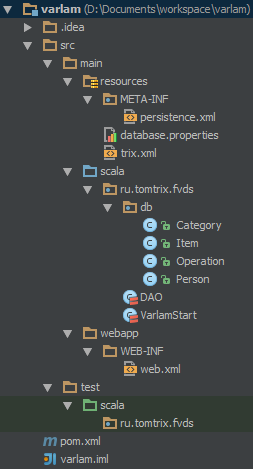
Теперь можно открывать страницу <http://82.146.37.95/varlam/start> в браузере. Вывод:

All right!

Далее с помощью MySQL-клиента можно убедиться в том, что все изменения в БД прошли успешно.

### Структура проекта

Ниже показана полная структура проекта:



#### pom.xml

<**project** *xmlns*="http://maven.apache.org/POM/4.0.0" *xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xsi*:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<**modelVersion**>4.0.0</**modelVersion**>

<**groupId**>ru.tomtrix.fvds</**groupId**>

<**artifactId**>varlam</**artifactId**>

<**packaging**>war</**packaging**>

<**version**>1.0.0</**version**>

<**name**>Varlam Project</**name**>

<**url**>http://varlam.com</**url**>

<**inceptionYear**>2014</**inceptionYear**>

<**properties**>

<**scala**.version>2.10.3</**scala**.version>

<**db**.package>*${project.groupId}*.db</**db**.package>

</**properties**>

<**repositories**>

<**repository**>

<**id**>scala-tools.org</**id**>

<**name**>Scala-Tools Maven2 Repository</**name**>

<**url**>http://scala-tools.org/repo-releases</**url**>

</**repository**>

</**repositories**>

<**pluginRepositories**>

<**pluginRepository**>

<**id**>scala-tools.org</**id**>

<**name**>Scala-Tools Maven2 Repository</**name**>

<**url**>http://scala-tools.org/repo-releases</**url**>

</**pluginRepository**>

</**pluginRepositories**>

<**dependencies**>

<!--Scala Library-->

<**dependency**>

<**groupId**>org.scala-lang</**groupId**>

<**artifactId**>scala-library</**artifactId**>

<**version**>*${scala.version}*</**version**>

</**dependency**>

<!--Spring Context Management-->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-context</**artifactId**>

<**version**>4.0.1.RELEASE</**version**>

</**dependency**>

<!--Spring Entity & Transaction Management-->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-orm</**artifactId**>

<**version**>4.0.1.RELEASE</**version**>

</**dependency**>

<!--Hibernate Java Persistence API-->

<**dependency**>

<**groupId**>org.hibernate</**groupId**>

<**artifactId**>hibernate-entitymanager</**artifactId**>

<**version**>4.3.1.Final</**version**>

</**dependency**>

<!-- MySQL connector -->

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.29</**version**>

</**dependency**>

<!-- Servlet-API with PROVIDED scope -->

<**dependency**>

<**groupId**>javax.servlet</**groupId**>

<**artifactId**>javax.servlet-api</**artifactId**>

<**version**>3.0.1</**version**>

<**scope**>provided</**scope**>

</**dependency**>

</**dependencies**>

<**build**>

<**sourceDirectory**>src/main/scala</**sourceDirectory**>

<**testSourceDirectory**>src/test/scala</**testSourceDirectory**>

<**finalName**>*${project.artifactId}*</**finalName**>

<**plugins**>

<!--Scala Build Plugin-->

<**plugin**>

<**groupId**>org.scala-tools</**groupId**>

<**artifactId**>maven-scala-plugin</**artifactId**>

<**executions**>

<**execution**>

<**goals**>

<**goal**>compile</**goal**>

<**goal**>testCompile</**goal**>

</**goals**>

</**execution**>

</**executions**>

<**configuration**>

<**scalaVersion**>*${scala.version}*</**scalaVersion**>

<**args**>

<**arg**>-target:jvm-1.5</**arg**>

</**args**>

</**configuration**>

</**plugin**>

<!--Hibernate Reverse-Engineer-Plugin-->

<**plugin**>

<**groupId**>org.codehaus.mojo</**groupId**>

<**artifactId**>hibernate3-maven-plugin</**artifactId**>

<**version**>2.2</**version**>

<**executions**>

<**execution**>

<**id**>hbm2cfgxml</**id**>

<**phase**>generate-resources</**phase**>

<**goals**>

<**goal**>hbm2cfgxml</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2cfgxml</**name**>

<**implementation**>jdbcconfiguration</**implementation**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

</**componentProperties**>

</**configuration**>

</**execution**>

<**execution**>

<**id**>hbm2hbmxml</**id**>

<**phase**>generate-resources</**phase**>

<**goals**>

<**goal**>hbm2hbmxml</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2hbmxml</**name**>

<**outputDirectory**>target/classes</**outputDirectory**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

</**componentProperties**>

</**configuration**>

</**execution**>

<**execution**>

<**id**>hbm2java</**id**>

<**phase**>generate-sources</**phase**>

<**goals**>

<**goal**>hbm2java</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2java</**name**>

<**implementation**>configuration</**implementation**>

<**outputDirectory**>src/main/scala</**outputDirectory**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

<**configurationfile**>target/hibernate3/generated-mappings/hibernate.cfg.xml</**configurationfile**>

<**jdk5**>true</**jdk5**>

<**ejb3**>true</**ejb3**>

</**componentProperties**>

</**configuration**>

</**execution**>

</**executions**>

<**dependencies**>

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.29</**version**>

</**dependency**>

</**dependencies**>

</**plugin**>

</**plugins**>

</**build**>

</**project**>

#### resources/trix.xml

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**beans** *xmlns*="http://www.springframework.org/schema/beans"

*xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xmlns*:context="http://www.springframework.org/schema/context" *xmlns*:tx="http://www.springframework.org/schema/tx"

*xsi*:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

<!--Force Spring to inject dependencies via annotations-->

<**context:annotation-config** />

<!--Inject the Transaction Manager via @Transactional annotation-->

<**tx:annotation-driven** *transaction-manager*="transactionManager" />

<!--Spring Entity Manager Factory built on META-INF/persistence.xml-->

<**bean** *id*="entityManagerFactory" *class*="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">

<**property** *name*="persistenceUnitName" *value*="trixik" />

</**bean**>

<!--Spring Transaction Manager-->

<**bean** *id*="transactionManager" *class*="org.springframework.orm.jpa.JpaTransactionManager">

<**property** *name*="entityManagerFactory" *ref*="entityManagerFactory" />

</**bean**>

<!--My DAO-->

<**bean** *id*="dao" *class*="ru.tomtrix.fvds.DAO" />

</**beans**>

#### resources/database.properties

hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

hibernate.connection.driver\_class = com.mysql.jdbc.Driver

hibernate.connection.url = jdbc:mysql://82.146.37.95:3306/varlam

hibernate.connection.username = mitrakov

hibernate.connection.password= 541888

#### resources/META-INF/persistence.xml

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**persistence** *version*="1.0" *xmlns*="http://java.sun.com/xml/ns/persistence"

*xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xsi*:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd">

<**persistence-unit** *name*="trixik" *transaction-type*="RESOURCE\_LOCAL">

<**provider**>org.hibernate.jpa.HibernatePersistenceProvider</**provider**>

<**properties**>

<**property** *name*="hibernate.dialect" *value*="org.hibernate.dialect.MySQL5Dialect"/>

<**property** *name*="hibernate.connection.driver\_class" *value*="com.mysql.jdbc.Driver"/>

<**property** *name*="hibernate.show\_sql" *value*="true"/>

<**property** *name*="hibernate.connection.username" *value*="mitrakov"/>

<**property** *name*="hibernate.connection.password" *value*="541888"/>

<**property** *name*="hibernate.connection.url" *value*="jdbc:mysql://127.0.0.1:3306/varlam"/>

<**property** *name*="hibernate.connection.characterEncoding" *value*="UTF-8"/>

</**properties**>

</**persistence-unit**>

</**persistence**>

#### webapp/WEB-INF/web.xml

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**web-app**>

<**display-name**>Trix Web Application</**display-name**>

<**servlet**>

<**servlet-name**>VarlamStart</**servlet-name**>

<**servlet-class**>ru.tomtrix.fvds.VarlamStart</**servlet-class**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>VarlamStart</**servlet-name**>

<**url-pattern**>/start</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

## Автоматизация труда

### Сохранение проекта в Git

#### VCS и Системы управления проектами

Проблема управления проектами решается, как правило, одним из следующих способов:

* Система контроля версий (VCS, Version Control System)
* Система управления проектами (Project Management System)

Второй случай предоставляет более широкий спектр возможностей: он включает непосредственно VCS, а также Wiki, Ussue Tracker, Bug Tracker, форумы, группы новостей и прочие инструменты.

Среди VCS особо популярны:

* Git
* Subversion SVN
* Mercurial
* Bazaar

Среди хостингов ПО наиболее известны:

* Github
* Google Code
* SourceForge
* Launchpad
* BitBacket

Некоторые из них ориентированы на работу с конкретной VCS (напр, Github), другие же поддерживают множество различных VCS.

Ссылка: <http://ru.wikipedia.org/wiki/Сравнение_хостингов_для_проектов_свободного_ПО>

Для данного проекта была выбрана площадка SourceForge и система контроля версий Git.

#### Регистрация на SourceForge

Регистрация в системе SourceForge не должна вызвать особых проблем. После подтверждения регистрации можно создавать проект. В нашем случае проект носит имя **Varlam** (<https://sourceforge.net/projects/varlam/>)

#### Установка и настройка Git для Windows

Инструменты

* Git 1.8.0 (<http://git-scm.com/>)

После установки Git требуется открыть Git Bash и выполнить некоторые настройки.

Первое: настроить учётную запись:

git config --global user.name mitrakov-artem

git config --global user.email "mitrakov-artem@ya.ru"

Следующий этап: установка окончаний строк. Это позволит писать код в любом формате (LF, CRLF), а сохранять в нормальном формате (LF).

git config --global core.autocrlf true

git config --global core.safecrlf false

#### Сохранение кода в Git

Теперь нужно перейти в каталог и инициализировать репозиторий:

git init

Затем следует добавить все нужные файлы каталога. Проще всего добавить сразу все файлы каталога, указав точку («**git add .**»). Исключить некоторые файлы из каталога поможет файл «.gitignore». В нашем случае он выглядит так:

.idea/\*

target/\*

varlam.iml

Теперь можно добавлять:

git add .

Делаем коммит:

git commit -a -m 'Initial commit'

Добавляем удалённый репозиторий (адрес предоставляется сервисом SourceForge)

git remote add origin ssh://mitrakov@git.code.sf.net/p/varlam/code

Загружаем код на удалённый репозиторий:

git push origin master

И последнее: создаём «ветвь слежения» (Tracking-Branch. Это позволяет связать локальную и удалённую ветви, чтобы в будущем можно было вытянуть изменения с помощью **git pull**

git branch --track origin/master

Сделано. Можно теперь зайти на SourceForge и убедиться, что всё ОК.

Теперь после любых изменений в проекте достаточно набрать:

git add .

git commit –m ‘Some text’

git push origin master

Чтобы вытянуть чужие изменения в коде, нужно набрать:

git pull

Замечание 1. Вместо консоли можно использовать множество GUI-инструментов:

* Git GUI (входит в комплект установки),
* Git-plugin в IntelliJ IDEA (предустановлен),
* Tortoise Git.

Замечание 2. Текущее состояние локального репозитория можно посмотреть с помощью команды:

git status

Замечание 3. Поскольку Git Bash является портом Unix-Bash-Shell, пути следует прописывать в Unix-формате, напр.

cd /c/users/tom/documents/

#### Интеграция с IntelliJ IDEA

Git-плагин в IntelliJ IDEA предоставляет массу удобных возможностей. Помимо стандартных команд (**pull**, **push**, **commit**, **stash**, **merge**), он позволяет в редакторе кода отслеживать изменённые места, просматривать историю, отслеживать annotations и др.

Для интеграции Git с IntelliJ IDEA следует открыть проект (убедитесь, что в проекте есть каталог **.git**) и выбрать VCS → Enable Version Control Integration → Git

### Maven-tomcat-plugin

Для автоматического развёртывания приложения на стороне сервера рекомендуется поставить **maven-tomcat-plugin**. Для этого требуется добавить в **pom.xml** в секцию **plugins**:

<!--Deploy Tomcat7 Plugin-->

<**plugin**>

<**groupId**>org.apache.tomcat.maven</**groupId**>

<**artifactId**>tomcat7-maven-plugin</**artifactId**>

<**version**>2.0</**version**>

<**configuration**>

<**url**>http://82.146.37.95:80/manager/text</**url**>

<**username**>trix</**username**>

<**password**>12345</**password**>

<**path**>/*${project.build.finalName}*</**path**>

<**update**>true</**update**>

</**configuration**>

<**executions**>

<**execution**>

<**phase**>install</**phase**>

<**goals**>

<**goal**>deploy</**goal**>

</**goals**>

</**execution**>

</**executions**>

</**plugin**>

Для запуска плагина нужно выполнить его команду **deploy**. Чтобы включить эту команду в жизненный цикл сборки, мы её добавили в фазу **install**. Параметр **update** говорит о том, следует ли выполнить redeploy в случае, если на сервере уже запущен экземпляр web-приложения.

Далее требуется добавить права доступа на Tomcat. На сервере находим файл **/etc/tomcat7/tomcat-users.xml** и в секцию пользователей добавляем следующие параметры:

<**role** *rolename*="manager-script"/>

<**user** *username*="trix" *password*="12345" *roles*="manager-script"/>

Следует оговорить, что настройки часто меняются от версии к версии, поэтому лучше всё-таки ознакомиться с документацией.

Теперь команда **mvn clean install** выполнит деплой на сервер.

Замечание 1. Процесс развёртывания по факту осуществляет приложение **manager**, расположенное в папке **webapps** Томката. В нашем случае почему-то такого приложения не оказалось. Наверное, есть какой-то правильный способ установки нового приложения, но самый простой метод: зайти на официальный сайт проекта и найти страницу с архивами всех версий:

<http://archive.apache.org/dist/tomcat/tomcat-7/>

Далее скачиваем нужную версию Tomcat (напр, v7.0.28) и из архива распаковываем приложение **manager** в каталог **webapps**.

Замечание 2. На самом деле логин и пароль из pom-файла можно скрыть. Для этого вместо них пишем **<server>trix\_server</server>**, а затем в настройках Maven’a (файл **settings.xml**) в секции **servers** надо прописать учётную запись:

<**server**>

<**id**>trix\_server</**id**>

<**username**>trix</**username**>

<**password**>12345</**password**>

</**server**>

Замечание 3. После 15-20 редеплоев сервер Tomcat валится с ошибкой нехватки памяти. С этой проблемой предполагается справиться чуть позже, а пока можно просто сделать перезапуск сервера:

sudo service tomcat7 restart

### Логгирование с Log4j

Существует масса прекрасных логгеров для Java:

* Log4j
* Slf4j
* Logback

Мы воспользуемся первым, как наиболее популярным, хотя существует мнение, что наиболее мощным является Logback.

Добавляем зависимость:

<!--Log4J Logger-->

<**dependency**>

<**groupId**>log4j</**groupId**>

<**artifactId**>log4j</**artifactId**>

<**version**>1.2.17</**version**>

</**dependency**>

В resources добавляем файл **log4j.properties**

# Console appender

log4j.appender.appender1=org.apache.log4j.ConsoleAppender

log4j.appender.appender1.target=System.out

log4j.appender.appender1.layout=org.apache.log4j.PatternLayout

log4j.appender.appender1.layout.conversionPattern=%d{ABSOLUTE} %5p %t %c{1}:%M:%L - %m%n

# File appender

log4j.appender.appender2=org.apache.log4j.FileAppender

log4j.appender.appender2.file=${catalina.base}/logs/varlam.log

log4j.appender.appender2.append=false

log4j.appender.appender2.layout=org.apache.log4j.PatternLayout

log4j.appender.appender2.layout.conversionPattern=%d{ABSOLUTE} %5p %t %c{1}:%M:%L - %m%n

# Root logger

log4j.rootLogger=WARN, appender2

# Main application logger

log4j.logger.mainLogger=DEBUG

# Set other loggers to ERROR level (otherwise they flood the log!)

log4j.logger.org.hibernate.internal.util.xml.DTDEntityResolver=ERROR

log4j.logger.org.hibernate.ejb.HibernatePersistence=ERROR

log4j.logger.org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl=ERROR

Первая секция описывает аппендер для консоли, вторая – для файла, располагаемого в каталоге **${catalina.base}/logs/varlam.log**. Рекомендуется именно этот каталог, поскольку здесь точно не возникнет проблем с правами пользователя.

Секция **Root logger** назначает главный логгер приложения. Мы оставили уровень логгирования WARN и добавили файловый аппендер. Секция **Main application logger** назначает второй логгер. Именно им мы будем пользоваться. Здесь указывать аппендер не нужно, поскольку данный логгер является потомком root-логгера.

Последняя секция была добавлена уже постфактум в виду того, что многие Hibernate’овские классы тоже используют Log4j и при этом наполняют лог всякой ерундой. Для них мы просто повысили уровень логгирования до ERROR.

## Каркас приложения

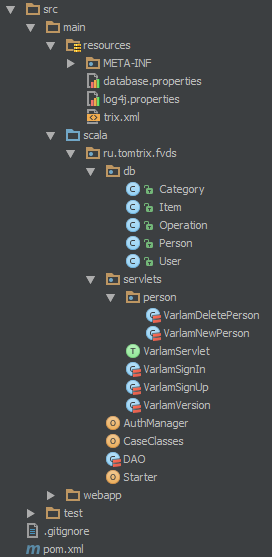
В этом разделе будет представлен код проекта версии 1.0.0 build 53. Очевидно, код будет многократно дописываться/переписываться, поэтому актуальную версию см. в репозиториях:

* https://mitrakov@git.code.sf.net/p/varlam/code
* https://mitrakov@git.code.sf.net/p/fvds-commons/code

Либо на сайте SourceForge:

* <https://sourceforge.net/p/varlam/code>
* <https://sourceforge.net/p/fvds-commons/code>

Общая структура проекта показана ниже:



### Добавление зависимости Commons

Проект использует самописную библиотеку **fvds-commons**:

<!--Common Library -->

<**dependency**>

<**groupId**>ru.tomtrix.fvds</**groupId**>

<**artifactId**>fvds-commons</**artifactId**>

<**version**>1.0.4</**version**>

</**dependency**>

### Изменение web.xml

Поскольку впредь сервлеты будут описываться через аннотации, файл **web-xml** значительно сократился:

<?**xml** *version*="1.0" *encoding*="UTF-8"?>

<**web-app**>

<**display-name**>Trix Web Application</**display-name**>

</**web-app**>

### Глобальный объект Starter.scala

**package** ru.tomtrix.fvds

**import** *javax.persistence.EntityManagerFactory*

***import*** *org.apache.log4j.Logger*

***import*** *org.springframework.context.support.ClassPathXmlApplicationContext*

/\*\*

\* Global application singleton

\*/

**object** Starter {

/\*\* Spring XML context \*/

private **val** factory = **new** ClassPathXmlApplicationContext("trix.xml")

/\*\* Entity Manager Factory<br>'''ATTENTION!''' it must be closed before the application shutdown \*/

**val** emf = factory.getBean("entityManagerFactory").asInstanceOf[EntityManagerFactory]

/\*\* Data Access Object that provides all JPA capabilities \*/

**val** dao = factory.getBean("dao").asInstanceOf[DAO]

/\*\* Global logger \*/

**val** logger = Logger.getLogger("mainLogger")

}

Объект отвечает за глобальные переменные: ссылку на ApplicationContext (**factory**), Фабрику менеджера сущностей (**emf**), экземпляр DAO (**dao**) и логгер (**logger**).

### Доступ к JPA: DAO.scala

**package** ru.tomtrix.fvds

**import** *javax.persistence.\_*

*import scala.language.\_*

*import scala.compat.Platform*

*import org.springframework.transaction.annotation.Transactional*

*import ru.tomtrix.fvds.Utils.\_*

*import ru.tomtrix.fvds.Starter.\_*

/\*\*

\* Rewritten version of DAO generated by <a href="http://mojo.codehaus.org/hibernate3-maven-plugin/">hibername3-maven-plugin</a>

\* <br>Injected by '''Spring Framework'''

\*/

@Transactional

**class** DAO {

@PersistenceContext private **val** entityManager: EntityManager = *null*

**def** **persist**(*transientInstance*: AnyRef) = safe {

**val** startTime = Platform.currentTime

entityManager.persist(transientInstance)

logger debug s"persisting done in ${Platform.currentTime - startTime} msec"

}

**def** **remove**(*persistentInstance*: AnyRef) = safe {

**val** startTime = Platform.currentTime

entityManager.remove(entityManager.merge(persistentInstance))

logger debug s"removing done in ${Platform.currentTime - startTime} msec"

}

**def** **merge**[T](*detachedInstance*: T) = safe {

**val** startTime = Platform.currentTime

**val** result = entityManager.merge(detachedInstance)

logger debug s"merging done in ${Platform.currentTime - startTime} msec"

result

}

**def** **findById**[T](*id*: Long, *clazz*: Class[T]) = safe {

**val** startTime = Platform.currentTime

**val** result = entityManager.find(clazz, id)

logger debug s"getting instance by id $id done in ${Platform.currentTime - startTime} msec"

result

}

**def** **findBySQL**[T](*sql*: String, *clazz*: Class[T], *parameters*: Map[String, AnyRef] = Map.empty) = safe {

**val** startTime = Platform.currentTime

**val** query = entityManager.createNativeQuery(sql, clazz)

parameters foreach { p => query.setParameter(p.\_1, p.\_2)}

**val** result = query.getResultList.toArray.map{\_.asInstanceOf[T]}.toList

logger debug s"getting instances by SQL: $sql done in ${Platform.currentTime - startTime} msec"

result

} getOrElse List.empty

**def** **findSingleBySQL**[T](*sql*: String, *clazz*: Class[T], *parameters*: Map[String, AnyRef] = Map.empty) = safe {

**val** startTime = Platform.currentTime

**val** query = entityManager.createNativeQuery(sql, clazz)

parameters foreach { p => query.setParameter(p.\_1, p.\_2)}

**val** result = query.getSingleResult.asInstanceOf[T]

logger debug s"getting instance by SQL: $sql done in ${Platform.currentTime - startTime} msec"

result

}

}

DAO-класс был слегка модифицирован по сравнению с предыдущей версией. Был добавлен логгер (вместо println), метод safe (вместо try-catch), вычисление времени запроса, а также добавлены методы выборки по SQL.

### Контейнер case-классов. CaseClasses.scala

**package** ru.tomtrix.fvds

**import** *javax.servlet.http.HttpServletResponse*

*import ru.tomtrix.fvds.Starter.\_*

/\*\*

\* Singleton containing simple case classes

\*/

**object** CaseClasses {

/\*\*

\* Class that represents a simple HTTP result. Use [[ru.tomtrix.fvds.CaseClasses.Result#write write]] method to get it done

\* **@param** resp HttpServletResponse

\* **@param** status HTTP status (e.g. 200 for OK)

\* **@param** msg text body message response

\*/

**case** **class** Result(*resp*: HttpServletResponse, *status*: Int, *msg*: String) {

/\*\*

\* Commits the HTTP result, sends response to a client & flushes the buffers

\*/

**def** **write**() {

logger debug s"""Response "$msg" sent to client with status=$status """

resp.setCharacterEncoding("UTF-8")

resp.getWriter println msg

resp setStatus status

resp.flushBuffer()

}

}

}

Согласно названию, данный объект содержит определения case-классов. В дальнейшем этот класс будет расширяться. Так, например, класс **Result** используется для отсылки на клиент некоторого сообщения с некоторым статусом.

### Менеджер аутентификации: AuthManager.scala

**package** ru.tomtrix.fvds

**import** *java.util.concurrent.ConcurrentHashMap*

*import scala.language.\_*

*import scala.concurrent.Future*

*import scala.concurrent.duration.\_*

*import akka.util.Timeout*

*import akka.actor.ActorSystem*

*import com.redis.RedisClient*

*import ru.tomtrix.fvds.Utils.\_*

*import ru.tomtrix.fvds.db.User*

*import ru.tomtrix.fvds.Starter.\_*

*import ru.tomtrix.fvds.ExtString.\_*

/\*\*

\* Singleton handling passwords and tokens

\*/

**object** AuthManager {

// Redis dependencies

implicit private **val** system = ActorSystem("redis-client")

implicit private **val** executionContext = system.dispatcher

implicit private **val** timeout = Timeout(*3* seconds)

/\*\* Redis Client \*/

private **val** redis = RedisClient("localhost")

/\*\* Map {Username -> Token} \*/

private **val** users = **new** ConcurrentHashMap[String, String]()

/\*\*

\* Saves new {username; password} pair

\* **@param** username login

\* **@param** hash hash of password

\*/

**def** **saveHash**(*username*: String, *hash*: String) = {

logger debug s"Try to save $username with hash $hash"

**if** (username.isBlank || hash.isBlank) Future.failed(**new** RuntimeException("Specify username & password"))

**else** { // it'd better wrap to a transaction

dao.persist(**new** User(username))

redis.set(username, hash)

}

}

/\*\*

\* Checks the password

\* **@param** username login

\* **@param** hash hash of password

\* **@return** '''true''', if the password is correct

\*/

**def** **checkHash**(*username*: String, *hash*: String) = {

logger debug s"Checking $username with hash $hash"

**if** (username.isBlank || hash.isBlank) Future.failed(**new** RuntimeException("Specify username & password"))

**else** redis.get[String](username) map {

**case** Some(s) => s == hash.str

**case** *None* => *false*

}

}

/\*\*

\* Saves new {username; password} pair

\* <br>'''NOT RECOMMENDED!''' Use [[ru.tomtrix.fvds.AuthManager#saveHash saveHash]] instead

\* **@param** username login

\* **@param** password pure password

\*/

**def** **savePassword**(*username*: String, *password*: String) = {

logger debug s"Try to save $username with password $password"

**if** (username.isBlank || password.isBlank) Future.failed(**new** RuntimeException("Specify username & password"))

**else** saveHash(username, password.toSHA256)

}

/\*\*

\* Checks the password

\* <br>'''NOT RECOMMENDED!''' Use [[ru.tomtrix.fvds.AuthManager#checkHash checkHash]] instead

\* **@param** username login

\* **@param** password pure password

\* **@return** '''true''', if the password is correct

\*/

**def** **checkPassword**(*username*: String, *password*: String) = {

logger debug s"Checking $username with password $password"

**if** (username.isBlank || password.isBlank) Future.failed(**new** RuntimeException("Specify username & password"))

**else** checkHash(username, password.toSHA256)

}

/\*\*

\* Registers a user for a session + generates new token

\* **@param** username login

\* **@return** token

\*/

**def** **register**(*username*: String) = {

logger debug s"Registering $username"

**if** (username.isBlank) ""

**else** {

**val** token = randomString

users.put(username.str, token)

token

}

}

/\*\*

\* Checks whether a user is registered on current session

\* **@param** username login

\* **@param** token token gained by [[ru.tomtrix.fvds.AuthManager#register register]] method

\* **@return** '''true''', if the token is correct

\*/

**def** **isRegistered**(*username*: String, *token*: String) = {

logger debug s"Checking $username with token $token"

!username.isBlank && users.get(username.str).str == token.str

}

}

Этот класс отвечает за регистрацию и проверку аутентификации пользователей. Для начала пользователю нужно зарегистрироваться, используя методы **saveHash** или **savePassword**. Далее при любом входе необходимо выполнить следующие шаги:

* Проверить правильность логина-пароля (**checkHash** или **checkPassword**)
* Получить токен (метод **register**)
* При каждой операции (требующей авторизации) проверять токен (метод **isRegistered**)

Замечание 1. Пока что время жизни токена не ограничено.

Замечание 2. метод **savePassword** принимает своим аргументом пароль в чистом виде, в то время как метод **saveHash** принимает хэш от пароля. Очевидно, второй вариант предпочтительнее с точки зрения безопасности.

Для хранения пар {логин; хэш} применяется нереляционная СУБД **Redis**. Сообщество **debasishg** выпустило 2 библиотеки для работы с **Redis** в Scala. Первая использует обычные блокирующие вызовы. Вторая же (**redisreact\_2.10**) использует асинхронный доступ к данным с помощью библиотеки **akka**. Мы воспользуемся именно этим вариантом:

<!--Redis Reactive Non-Blocking Library -->

<**dependency**>

<**groupId**>net.debasishg</**groupId**>

<**artifactId**>redisreact\_2.10</**artifactId**>

<**version**>0.3</**version**>

</**dependency**>

Асинхронный стиль доступа к данным[[4]](#footnote-4) является более предпочтительным, однако он, как правило, вызывает некоторые трудности в разработке алгоритмов.

### Суперкласс для сервлетов: VarlamServlet.scala

**package** ru.tomtrix.fvds.servlets

import *javax.servlet.http.\_*

*import scala.util.\_*

*import scala.concurrent.Future*

*import scala.concurrent.ExecutionContext.Implicits.global*

*import ru.tomtrix.fvds.Utils.\_*

*import ru.tomtrix.fvds.db.User*

*import ru.tomtrix.fvds.Starter.\_*

*import ru.tomtrix.fvds.ExtString.\_*

*import ru.tomtrix.fvds.AuthManager.\_*

*import ru.tomtrix.fvds.CaseClasses.Result*

/\*\*

\* Superclass for all servlets. Provides additional functions like

\* [[ru.tomtrix.fvds.servlets.VarlamServlet#async async]] and

\* [[ru.tomtrix.fvds.servlets.VarlamServlet#authenticated authenticated]]

\* <br>Ensure that a servlet '''MUST''' extend this trait

\*/

**trait** VarlamServlet **extends** *HttpServlet* {

/\*\*

\* Performs the operation only if a user is registered by [[ru.tomtrix.fvds.AuthManager#register register]] method

\* **@param** req HttpServletRequest

\* **@param** resp HttpServletResponse

\* **@param** f code

\*/

**def** **authenticated**(*req*: HttpServletRequest, *resp*: HttpServletResponse)(*f*: => Unit) = safe {

**if** (isRegistered(req.getHeader("username").str, req.getHeader("token").str)) f

**else** Result(resp, *401*, "Unauthorized").write()

}

/\*\*

\* Performs an asynchronous operation. Ensure that if you use asynchronous code inside a servlet handler then you

\* '''MUST''' wrap it into '''async''' and annotate with [[javax.servlet.annotation.WebServlet WebServlet]] annotation with

\* a value \_\_asyncSupported = true\_\_. Otherwise asynchronous code won't work correctly

\* **@param** req HttpServletRequest

\* **@param** resp HttpServletResponse

\* **@param** f code

\*/

**def** **async**(*req*: HttpServletRequest, *resp*: HttpServletResponse)(*f*: => Future[Result]) = safe {

**if** (req.isAsyncSupported) {

req.startAsync()

**val** context = req.getAsyncContext

context setTimeout *3000L*

f onComplete {

**case** Success(result) =>

result.write()

context.complete()

**case** Failure(e) =>

e.getMessage

Result(resp, *500*, e.getMessage).write()

context.complete()

}

}

**else** logger error "Async is not supported"

}

/\*\*

\* Retrieves a user by it's name

\* **@param** req HttpServletRequest

\*/

**def** **getUser**(*req*: HttpServletRequest) = {

**val** username = req.getHeader("username").str

dao.findSingleBySQL("SELECT \* FROM user WHERE username = :x", classOf[User], Map("x" -> username))

}

/\*\*

\* Destroys the servlet and closes Entity Manager Factory

\*/

override **def** **destroy**(): Unit = {

**if** (emf.isOpen) {

emf.close()

logger info "Shutting down application"

}

*super*.destroy()

}

}

Данный класс (или, точнее, трейт) является потомком для всех сервлетов приложения. Он содержит некторые полезные методы:

* **authenticated**: выполняет функцию только в том случае, если пользователь авторизован
* **async**: выполняет асинхронную функцию (она должна возвращать Future[Result])
* **getUser**: возвращает экземпляр пользователя из БД по его имени

Кроме того, здесь переопределяется метод destroy, который закрывает Фабрику Менеджера Сущностей, открытую ранее в объекте Starter. В связи с этим, НАСТОЯТЕЛЬНО РЕКОМЕНДУЕТСЯ все сервлеты наследовать от данного класса.

Небольшое замечание про асинхронную работу сервлета. В обычном случае исполнение сервлета завершается после отработки последней строки кода метода. Для отправки данных на клиент нужно сбросить внутренний буфер (**response.flushBuffer**) или закрыть объект **writer**[[5]](#footnote-5) (**writer.close**). В асинхронной модели нужно сделать следующее:

* указать, что сервлет работает асинхронно (напр, через аннотацию **WebServlet**)
* начать асинхронную операцию (**request.startAsync**)
* завершить асинхронную операцию (**request.getAsyncContext.complete**)

При этом операция **complete** вызывает сброс буфера.

### Сервлеты

#### Сервлет регистрации

**package** ru.tomtrix.fvds.servlets

import *javax.servlet.http.\_*

*import javax.servlet.annotation.WebServlet*

*import scala.concurrent.ExecutionContext.Implicits.global*

*import ru.tomtrix.fvds.Starter.\_*

*import ru.tomtrix.fvds.ExtString.\_*

*import ru.tomtrix.fvds.AuthManager.\_*

*import ru.tomtrix.fvds.CaseClasses.Result*

/\*\*

\* Registration servlet

\*/

@WebServlet(urlPatterns = Array("/sign/up"), asyncSupported = *true*)

**class** VarlamSignUp **extends** *VarlamServlet* {

override **def** **doPut**(*req*: HttpServletRequest, *resp*: HttpServletResponse): Unit = async(req, resp) {

**val** json = req.getReader.readLine().asJsonObject

**val** username = json.getOrElse("username", "").toString.str

**val** password = json.getOrElse("password", "").toString.str

**val** save = json.getOrElse("client", "").toString.str **match** {

**case** "android" => saveHash(username, password)

**case** \_ => savePassword(username, password)

}

logger debug s"User $username is gonna sign up"

save map {

**case** *true* => Result(resp, *200*, register(username))

**case** *false* => Result(resp, *401*, s"Error!")

}

}

}

Для регистрации передаётся логин и пароль в формате JSON. В зависимости от типа клиента передаваться может либо сам пароль, либо его хэш. Метод выполняется асинхронно.

#### Сервлет авторизации

**package** ru.tomtrix.fvds.servlets

import *javax.servlet.http.\_*

*import javax.servlet.annotation.WebServlet*

*import scala.concurrent.ExecutionContext.Implicits.global*

*import ru.tomtrix.fvds.Starter.\_*

*import ru.tomtrix.fvds.ExtString.\_*

*import ru.tomtrix.fvds.AuthManager.\_*

*import ru.tomtrix.fvds.CaseClasses.Result*

/\*\*

\* Auth servlet

\*/

@WebServlet(urlPatterns = Array("/sign/in"), asyncSupported = *true*)

**class** VarlamSignIn **extends** *VarlamServlet* {

override **def** **doPost**(*req*: HttpServletRequest, *resp*: HttpServletResponse): Unit = async(req, resp) {

**val** json = req.getReader.readLine().asJsonObject

**val** username = json.getOrElse("username", "").toString.str

**val** password = json.getOrElse("password", "").toString.str

**val** isChecked = json.getOrElse("client", "").toString.str **match** {

**case** "android" => checkHash(username, password)

**case** \_ => checkPassword(username, password)

}

logger debug s"User $username trying to log in"

isChecked map {

**case** *true* => Result(resp, *200*, register(username))

**case** *false* => Result(resp, *401*, "Access denied")

}

}

}

Для авторизации передаётся логин и пароль в формате JSON. В зависимости от типа клиента передаваться может либо сам пароль, либо его хэш. Метод выполняется асинхронно.

#### Сервлет для проверки версии

**package** ru.tomtrix.fvds.servlets

import *javax.servlet.http.\_*

*import javax.servlet.annotation.WebServlet*

*import ru.tomtrix.fvds.Utils.\_*

*import ru.tomtrix.fvds.CaseClasses.Result*

/\*\*

\* Simple servlet that shows the current build version

\*/

@WebServlet(urlPatterns = Array("/version"))

**class** VarlamVersion **extends** *VarlamServlet* {

override **def** **doGet**(*req*: HttpServletRequest, *resp*: HttpServletResponse): Unit = safe {

Result(resp, *200*, "Varlam Application 1.0.0.53").write()

}

}

Сервлет служит для отладочных целей. Код выполняется синхронно.

#### Сервлет для добавления новой персоны

**package** ru.tomtrix.fvds.servlets.person

**import** *javax.servlet.http.\_*

*import javax.servlet.annotation.WebServlet*

*import ru.tomtrix.fvds.Starter.\_*

*import ru.tomtrix.fvds.db.Person*

*import ru.tomtrix.fvds.ExtString.\_*

*import ru.tomtrix.fvds.servlets.VarlamServlet*

*import ru.tomtrix.fvds.CaseClasses.Result*

/\*\*

\* Creates a new person

\*/

@WebServlet(urlPatterns = Array("/person/new"))

**class** VarlamNewPerson **extends** *VarlamServlet* {

override **def** **doPut**(*req*: HttpServletRequest, *resp*: HttpServletResponse): Unit = authenticated(req, resp) {

**val** json = req.getReader.readLine().asJsonObject

**val** result = **for** {

name <- json get "name" map {\_.toString.str}

user <- getUser(req)

} **yield** {dao persist **new** Person(user, name); name}

result **match** {

**case** Some(x) => Result(resp, *200*, s"Person $x saved").write()

**case** *None* => Result(resp, *403*, "Fail to save person. Maybe name is empty?").write()

}

}

}

Сервлет добавляет новую персону согласно имени, передаваемого в JSON в поле **name**. Требует аутентификации пользователя. Уникальность имени здесь не контролируется: она контролируется только на уровне СУБД. Код выполняется синхронно.

#### Сервлет удаления персоны

**package** ru.tomtrix.fvds.servlets.person

**import** *javax.servlet.http.\_*

*import javax.servlet.annotation.WebServlet*

*import ru.tomtrix.fvds.Starter.\_*

*import ru.tomtrix.fvds.db.Person*

*import ru.tomtrix.fvds.ExtString.\_*

*import ru.tomtrix.fvds.servlets.VarlamServlet*

*import ru.tomtrix.fvds.CaseClasses.Result*

/\*\*

\* Removes a person

\*/

@WebServlet(urlPatterns = Array("/person/delete"))

**class** VarlamDeletePerson **extends** *VarlamServlet* {

override **def** **doDelete**(*req*: HttpServletRequest, *resp*: HttpServletResponse): Unit = authenticated(req, resp) {

**val** json = req.getReader.readLine().asJsonObject

**val** result = **for** {

name <- json get "name" map {\_.toString.str}

user <- getUser(req)

person <- getPerson(user, name)

} **yield** {dao remove person; name}

result **match** {

**case** Some(x) => Result(resp, *200*, s"Person $x deleted").write()

**case** *None* => Result(resp, *403*, s"Fail to delete a person. Maybe name is empty?").write()

}

}

}

Сервлет служит для удаления персоны по имени, передаваемого в JSON в поле **name**. Требует аутентификации пользователя. Существование записи в БД здесь не контролируется: это контролируется лишь на уровне СУБД. Код выполняется синхронно.

### Общий вид pom.xml

Ниже представлен файл pom.xml с учётом того, что логин и пароль от сервера Tomcat вынесены в настройки Maven (файл **conf/settings.xml**).

<**project** *xmlns*="http://maven.apache.org/POM/4.0.0" *xmlns*:xsi="http://www.w3.org/2001/XMLSchema-instance"

*xsi*:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<**modelVersion**>4.0.0</**modelVersion**>

<**groupId**>ru.tomtrix.fvds</**groupId**>

<**artifactId**>varlam</**artifactId**>

<**packaging**>war</**packaging**>

<**version**>1.0.0</**version**>

<**name**>Varlam Project</**name**>

<**url**>http://varlam.com</**url**>

<**inceptionYear**>2014</**inceptionYear**>

<**properties**>

<**scala**.version>2.10.2</**scala**.version>

<**db**.package>*${project.groupId}*.db</**db**.package>

<**project**.build.sourceEncoding>UTF-8</**project**.build.sourceEncoding>

</**properties**>

<**repositories**>

<**repository**>

<**id**>scala-tools.org</**id**>

<**name**>Scala-Tools Maven2 Repository</**name**>

<**url**>http://scala-tools.org/repo-releases</**url**>

</**repository**>

</**repositories**>

<**pluginRepositories**>

<**pluginRepository**>

<**id**>scala-tools.org</**id**>

<**name**>Scala-Tools Maven2 Repository</**name**>

<**url**>http://scala-tools.org/repo-releases</**url**>

</**pluginRepository**>

</**pluginRepositories**>

<**dependencies**>

<!--Scala Library-->

<**dependency**>

<**groupId**>org.scala-lang</**groupId**>

<**artifactId**>scala-library</**artifactId**>

<**version**>*${scala.version}*</**version**>

</**dependency**>

<!--Spring Context Management-->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-context</**artifactId**>

<**version**>4.0.1.RELEASE</**version**>

</**dependency**>

<!--Spring Entity & Transaction Management-->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-orm</**artifactId**>

<**version**>4.0.1.RELEASE</**version**>

</**dependency**>

<!--Hibernate Java Persistence API-->

<**dependency**>

<**groupId**>org.hibernate</**groupId**>

<**artifactId**>hibernate-entitymanager</**artifactId**>

<**version**>4.3.1.Final</**version**>

</**dependency**>

<!-- MySQL connector -->

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.29</**version**>

</**dependency**>

<!-- Servlet-API with PROVIDED scope -->

<**dependency**>

<**groupId**>javax.servlet</**groupId**>

<**artifactId**>javax.servlet-api</**artifactId**>

<**version**>3.0.1</**version**>

<**scope**>provided</**scope**>

</**dependency**>

<!--Redis Reactive Non-Blocking Library -->

<**dependency**>

<**groupId**>net.debasishg</**groupId**>

<**artifactId**>redisreact\_2.10</**artifactId**>

<**version**>0.3</**version**>

</**dependency**>

<!--Common Library -->

<**dependency**>

<**groupId**>ru.tomtrix.fvds</**groupId**>

<**artifactId**>fvds-commons</**artifactId**>

<**version**>1.0.4</**version**>

</**dependency**>

<!--Log4J Logger-->

<**dependency**>

<**groupId**>log4j</**groupId**>

<**artifactId**>log4j</**artifactId**>

<**version**>1.2.17</**version**>

</**dependency**>

</**dependencies**>

<**build**>

<**sourceDirectory**>src/main/scala</**sourceDirectory**>

<**testSourceDirectory**>src/test/scala</**testSourceDirectory**>

<**finalName**>*${project.artifactId}*</**finalName**>

<**plugins**>

<!--Scala Build Plugin-->

<**plugin**>

<**groupId**>org.scala-tools</**groupId**>

<**artifactId**>maven-scala-plugin</**artifactId**>

<**version**>2.15.2</**version**>

<**executions**>

<**execution**>

<**goals**>

<**goal**>compile</**goal**>

<**goal**>testCompile</**goal**>

</**goals**>

</**execution**>

</**executions**>

<**configuration**>

<**scalaVersion**>*${scala.version}*</**scalaVersion**>

<**args**>

<**arg**>-target:jvm-1.6</**arg**>

<**arg**>-unchecked</**arg**>

<**arg**>-deprecation</**arg**>

<**arg**>-feature</**arg**>

</**args**>

<**jvmArgs**>

<**jvmArg**>-Xms128m</**jvmArg**>

<**jvmArg**>-Xmx1024m</**jvmArg**>

</**jvmArgs**>

</**configuration**>

</**plugin**>

<!--Hibernate Reverse-Engineer-Plugin-->

<**plugin**>

<**groupId**>org.codehaus.mojo</**groupId**>

<**artifactId**>hibernate3-maven-plugin</**artifactId**>

<**version**>2.2</**version**>

<**executions**>

<**execution**>

<**id**>hbm2cfgxml</**id**>

<**phase**>generate-resources</**phase**>

<**goals**>

<**goal**>hbm2cfgxml</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2cfgxml</**name**>

<**implementation**>jdbcconfiguration</**implementation**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

</**componentProperties**>

</**configuration**>

</**execution**>

<**execution**>

<**id**>hbm2hbmxml</**id**>

<**phase**>generate-resources</**phase**>

<**goals**>

<**goal**>hbm2hbmxml</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2hbmxml</**name**>

<**outputDirectory**>target/classes</**outputDirectory**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

</**componentProperties**>

</**configuration**>

</**execution**>

<**execution**>

<**id**>hbm2java</**id**>

<**phase**>generate-sources</**phase**>

<**goals**>

<**goal**>hbm2java</**goal**>

</**goals**>

<**inherited**>false</**inherited**>

<**configuration**>

<**components**>

<**component**>

<**name**>hbm2java</**name**>

<**implementation**>configuration</**implementation**>

<**outputDirectory**>src/main/scala</**outputDirectory**>

</**component**>

</**components**>

<**componentProperties**>

<**packagename**>*${db.package}*</**packagename**>

<**configurationfile**>target/hibernate3/generated-mappings/hibernate.cfg.xml</**configurationfile**>

<**jdk5**>true</**jdk5**>

<**ejb3**>true</**ejb3**>

</**componentProperties**>

</**configuration**>

</**execution**>

</**executions**>

<**dependencies**>

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.29</**version**>

</**dependency**>

</**dependencies**>

</**plugin**>

<!--Deploy Tomcat7 Plugin-->

<**plugin**>

<**groupId**>org.apache.tomcat.maven</**groupId**>

<**artifactId**>tomcat7-maven-plugin</**artifactId**>

<**version**>2.0</**version**>

<**configuration**>

<**url**>http://82.146.37.95:80/manager/text</**url**>

<**server**>trix\_server</**server**>

<**path**>/*${project.build.finalName}*</**path**>

<**update**>true</**update**>

</**configuration**>

<**executions**>

<**execution**>

<**phase**>install</**phase**>

<**goals**>

<**goal**>deploy</**goal**>

</**goals**>

</**execution**>

</**executions**>

</**plugin**>

</**plugins**>

</**build**>

</**project**>

1. На самом деле это оказалось очень трудно заметить. Понадобилось порядка 2-3 часов, чтобы понять, почему сервлет по адресу <http://82.146.37.95/varlam/start> не работает, а стартовая страница работает [↑](#footnote-ref-1)
2. Там же есть SBT [↑](#footnote-ref-2)
3. Xms = initial size, Xmx = max size [↑](#footnote-ref-3)
4. В нашем случае только доступ к данным в Redis является асинхронным. Доступ к данным в MySQL выполняется синхронно [↑](#footnote-ref-4)
5. Что по факту и вызывает сброс буфера [↑](#footnote-ref-5)