Chapter 7

Programmer's guide

The application is deployed on `cwtest.felk.cvut.cz` virtual server. Tomcat is used as a servlet container serving application. After war archive is compiled, application is deployed using tomcat manager application located at https://cwtest.felk.cvut.cz/manager/html.

7.1 Development

7.1.1 Obtaining the source code

Complete source code is located in git repository at https://gitlab.fel.cvut.cz/svobodat/rozvrhy-support-apps. You can browse generated javadoc at http://cwtest.felk.cvut.cz:8080/rozvrhy_studentu/javadoc/index.html.

7.1.2 Importing project to IDE

In "Eclipse" or "Spring tool suite" (which is extended Eclipse), one can follow these steps:

1. Open new workspace at repository directory (choose workspace at IDE startup)
2. Open File -> Import -> Maven -> Existing maven projects
3. Select the pom.xml of project and click "Finish"

7.1.3 Productivity tips

Google plugin for eclipse[1] is not required for development. However, it is recommended as it can be helpful to beginners. Don't forget to use Source tab to generate getters, setters, constructors, toString methods and even methods based on interface. Static type check is one of biggest advantages of using Java over using dynamically typed language.

List of useful shortcuts:

1. `ctrl+o` - resolves imports

[1] https://developers.google.com/eclipse/?hl=cs

2. `ctrl+shift+t` - finds type (class, enum or interface) by name
3. `ctrl+l` - go to line

Creating junit test: right click on class that is to be tested, "new", "new JUnit test case"

7.1.4 Code style

The code adopts the standard Java conventions. Transient annotations are written with fully qualified names, so that they can be distinguished.Names of enums are prepended with `E`, names of interfaces with `I`.

7.1.5 Running the application with jetty

Maven plugin named "jetty-maven-plugin" is used to run application locally. Use target `make jetty` to run server. To run client application in development mode, use target make `gwtdev`.

7.1.6 Speeding up jetty startup

Jetty is scanning classpath for Serlvet 3 annotations. Project does not use Servler 3 features, so it would be only waste of time. Setting empty regular expression to `WebInfIncludeJarPattern` configuration variable skips scanning and speeds up startup of jetty significantly.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configure class="org.eclipse.jetty.webapp.WebAppContext">
3  <Call name="setAttribute">
4   <Arg>org.eclipse.jetty.server.webapp.WebInfIncludeJarPattern</Arg>
5   <Arg>^$</Arg>
6  </Call>
7 </Configure>
```

Listing 11: Setting empty regular expression to speed up startup of Jetty

7.1.7 Testing with JUnit

JUnit is powerful testing framework, that is used to run tests. However, application doesn't have 100% test coverage and tests are not design for automated testing, but for programmer to check wheter service returns what it should. Programmer can write test, or look at existing test and run it without starting web server, but cannot run all tests at once. Enabling automated testing would require mocking KOSapi web service, which was out of time frame of this work.

To run a test, use "test" target of "make" and set variable "TEST" to class name of testcase you want to run. You don't need to supply fully qualified name. An example of running test: `make test TEST=IStudentDAOTest`. Note that when test ends with error, your exception is located in "target/surefire-reports" folder, not in console output of maven. In order to use

dependency injection in JUnit test, test must extend `BaseTest` class, that is annotated to be run with `SpringJUnit4ClassRunner` class.

```
1 @RunWith(SpringJUnit4ClassRunner.class)
2 @ContextConfiguration(locations = {"/appContextJUnit.xml"})
3 public abstract class BaseTest { ... }
```

Listing 12: BaseTest class annotation that allows to use dependency injection in JUnit tests

7.2 Build

7.2.1 Makefile

Makefile is used because it provides interface to the build system that is generally known by programmers. It also allows to define useful targets.
Some of useful targets:

1. all - shows help
2. runserver - runs jetty and gwt codeserver for development
3. jetty - runs jetty
4. compile - compiles GWT application, generates javadoc, compiles server application, produces war that can be deployed
5. clean - cleans all compiled code

7.3 Production requirements

Server application requires at least Java 6 runtime environment and servlet container, such as Tomcat 6. Server with multicore processor is recommended since the application and web server are using concurrent processing. Client application requires modern web browser with CSS3 selectors capabilities, JavaScript and cookies enabled.

7.4 Development requirements

Developer must use unix based operating system [2] with Java 7 JRE (because of jetty), Java 6 JDK, GWTSDK (is downloaded by Makefile), wget (to download GWTSDK) and Maven 3. GWT plugin must be present in web browser for GWT development. Port `8080` must be open for listening.

[2] Ubuntu or MAC OS is fine

7.5 Instalation for development

1. Install Java 7 JDK, or Java 7 JRE and Java 6 JDK, if you don't have them
2. Install Maven 3 and wget packages if you don't have them
3. Checkout project from GIT repository
4. Run command `make runserver` that will do everything that is needed to the point that jetty and GWT DevMode is running
5. Open application in DevMode