**Architecture of Java Source Code Compiler Web App**

By Aldian Fazrihady

**Feature Tests**

There are several feature related questions whose answers will make us sure that we have implemented all the required features or not. Several automated tests are created to answer these questions.

**Has the Web Service successfully extracted the zip archive and compiled Java source code files?**

The zip archive extraction and the Java source code files compilation happen in the Web Service code. Specifically, it happens in the method compile of class WebServices.

achine generated alternative text:
WebServices 
CompilationResuIt compi le(St

All calls to web services (except for login) require an accessToken as one of the parameter. The requestor of compile service must be a valid user account. AccessToken will identify the user trying to access the compile service. If accessToken is invalid, the compile service must stop from servicing the request.

In the Web Service system, each user of the system has his own space to compile Java files. To conserve space, each new upload will erase and overwrite files from previous upload. The deletions are only done to the files of the current user, so the files of all other users are untouched. The result log of all compilations are logged to the database, so not everything is gone after previous uploads are deleted.



After cleaning up the user disk space, the zip file will be placed to the space and extracted. The zip file ideally contains only Java source code files. The files will be enumerated and compiled one by one. The compilation will be logged to a persisted an entity CompilationResult, and also returned to the caller.

achine generated alternative text:
@Entity Comp ilat ion Re sult 
—private lo



The test for this case is in method testWsCompile of class AppCompilerWebApplicationTests.

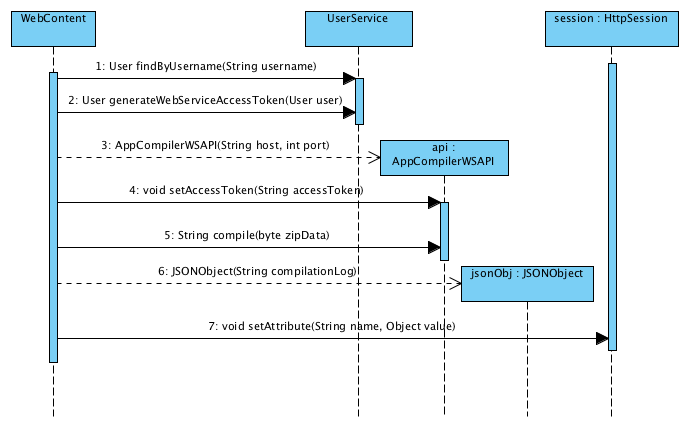
achine generated alternative text:
Oal!dWOJSAusa1 P!O,•, 

**Has the Web App client successfully sent zip files to Web Service and retrieved response from the Web Service?**

The home page has a form to upload a zip file. After choosing a zip file and clicking submit button, browser will send the file to the Web App.

Having received a zip file from the browser form, the Web App immediately sends the zip file to the Web Service. Web App accesses the Web Service using Web Service API.

achine generated alternative text:
WebContent.handIeFiIeUpIoad(MuItlpartFl e 



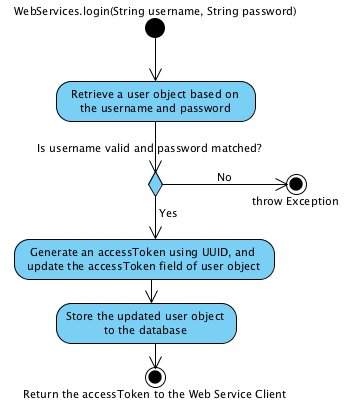
The test for this case is in method testWebHandleFileUpload and testWsApiCompile of class AppCompilerWebApplicationTests.

achine generated alternative text:
ppCompiIerWebAppIicationTest 
void testWeb

achine generated alternative text:
ppCompiIerWebAppIicationTest 
void testWsA

**Can the Web Service be accessed by any client, not only by this Web App?**

A Web Service should be accessible by any type of client including a command line app client as long as it has a valid access token. A client app can generate an access token by sending a valid username and password to the web service. A valid username and password for the Web Service is also a valid username and password for the Web App. Using the Web App, a user can register his username and password.



The test for this case is in method testWsLogin of class AppCompilerWebApplicationTests





**Can a new user register his/her account?**

If a user not logged in, the page will be redirected to login page. If the user thinks he doesn't have any username and password for this system, he can click the link to open registration page.

In the registration page the user can enter a new username and password and then click submit.

The test for this case is in method testWebRegistrationPost of class AppCompilerWebApplicationTests



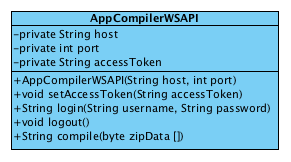
**Controller Classes**

Controller classes manage information and UI flow. This app has two controller classes. Class WebContent is the controller class to handle Web App. Class WebServices is the controller class to handle Web Service.

achine generated alternative text:
()uoawado 
(uolssas uo!ssasduH 'lapow lapo

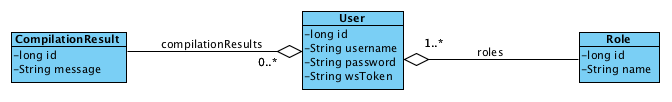
**Web Service API Class**

The controller class WebContent is not allowed to directly access class WebServices because they are on separate tiers and ideally installed on different machines. WebContent access WebServices using API class AppCompilerWSAPI



**Model Classes**

There are business domain classes that need to be persisted to database. These are the class CompilationResult, User, and Role.



**Implementation Stack**

The implementation of this Web App and Web Service is built on top of Spring framework.

Spring framework allow us minimize the attention on infrastructure code and allow us to focus on business code.

The Web app and Web Service is implemented in one source code bundle/package. Only at testing and deployment stage then, the bundle is copied to two different tier/machine to serve different purposes, as web app or as web service.

**Deployment**

The system currently has 3 tiers.

These are the 3 tiers:

1. The database server.
2. The web service.
3. The web app.

Those tiers can be installed on separate machines. They can also be installed on a single machines. We can experiment to find the best machine combinations.

**Database Server Installation**

We can choose any DBMS for this app. In the development I am using PostgreSQL. These are the step to deploy database:

1. Install MySQL or PostgreSQL on the machine intended for the database server. For example, you install the database at host 111.111.111.001 and port 5432
   * Database machine host: 111.111.111.001
   * Database machine port: 5432
2. Create a database and a user account to access the database. For example, you create a database named java\_app\_compiler, a user to access that database named aldian, and a password for that database and user, HoL1d4yz.
   * Database name: java\_app\_compiler
   * Username: aldian
   * Password: HoL1d4yz
3. You need those information about your database machine host, port, database name, username, and password to be put on two configuration files later. Therefore, don't loose that values, record them on somewhere save.
4. Start the database server.

**Install The Latest Version of Java 1.8**

Install Java 1.8 on the machines for Web Service and Web App. This step is completed if from the command line on that machines you can execute java and javac, and if you execute javac -version it shows the Java version you just installed.

**Install The Latest Version of Maven 3**

Install at least Maven 3.2.1 on the machines for Web Service and Web App. This step is completed If you execute mvn --version it shows the version you just installed.

**Web Service Installation**

On the zip file I uploaded to crossover when submitting this assignment there is a folder named app-compiler-web. Copy the folder to the machine that will run the Web Service. Navigate to that folder using command line shell. We are going to edit the configuration file at src/main/resources/application.properties.

Let us say the machine for Web Service is at host **111.111.111.002** and we are going to run it at port **8383**. The database configuration is using the value from the section Database Server Installation above. Therefore, you are going to put these values on application.properties:

**spring.jpa.database**=**POSTGRESQL**

**spring.datasource.platform**=**postgres**

**spring.jpa.show-sql**=**true**

**spring.jpa.hibernate.ddl-auto**=**update**

**spring.database.driverClassName**=**org.postgresql.Driver**

**spring.datasource.url**=**jdbc:postgresql://111.111.111.001:5432/java\_app\_compiler**

**spring.datasource.username**=**aldian**

**spring.datasource.password**=**HoL1d4yz**

**server.port**=**8383**

**app.ws.host**=**111.111.111.002**

**app.ws.port**=**8383**

If you are going to use database system other than PostgreSQL, you need to put different values to several attributes above.

**Web App Installation**

Copy app-compiler-web to the machine that will run the Web App. Navigate to that folder using command line shell. We are going to edit the configuration file at src/main/resources/application.properties.

Let us say the machine for Web App is at host **111.111.111.003** and we are going to run it at port **8080**. The database configuration is using the value from the section Database Server Installation above. This Web App will access Web Service at 111.111.111.002:8383. Therefore, you are going put these values on application.properties:

**spring.jpa.database**=**POSTGRESQL**

**spring.datasource.platform**=**postgres**

**spring.jpa.show-sql**=**true**

**spring.jpa.hibernate.ddl-auto**=**update**

**spring.database.driverClassName**=**org.postgresql.Driver**

**spring.datasource.url**=**jdbc:postgresql://111.111.111.001:5432/java\_app\_compiler**

**spring.datasource.username**=**aldian**

**spring.datasource.password**=**HoL1d4yz**

**server.port**=**8080**

**app.ws.host**=**111.111.111.002**

**app.ws.port**=**8383**

**Run the Web Service and Initialize the Database**

Go to the machine for the Web Service. Using command line terminal, change directory to folder app-compiler-web. Type and execute this command:

mvn spring-boot:run

Database tables will be generated, the Web Service will be ready to accept requests.

**Run The Test**

Go to the machine for the Web App. The test requires the Web Service to be already run because the test also tests the connection to the Web Service. If you haven't run it, please run the Web Service using the instruction in the previous section.

Using command line terminal, change directory to folder app-compiler-web. Type and execute this command:

mvn test

Hopefully the app will pass all test on your machine.

**Run the Web App**

Go to the machine for the Web App. Using command line terminal, change directory to folder app-compiler-web. Type and execute this command:

mvn spring-boot:run

**Access the Web App using Web Browser**

Using web browser, open the URL of the Web App at <http://111.111.111.003:8080> (or other host and port depending on your configuration).

At first you need to register your account. After registration you can login and upload java files archived in a zip file. The Web App will send the zip file to the Web Service. The Web Service will extract the zip archive and compile Java files inside the archive. The Web App will receive the compilation log from the Web Service. The log will be shown in the web browser.

**Future Improvements**

When I got the chance to improve this app, these are the list I want to improve:

1. The UI surely needs fixing.
2. I want to allow user see the history of his compilation.
3. Compiling large archive of java files surely will take some time. Therefore, using job queue to run the java compilation is more appropriate then directly run the compilation on web service. Surely I will need to implement more method in the web service to allow client to check whether or not compilation has finished.