Victoria University of Wellington

SWEN 301 Structured Methods

Assignment 3 (Individual) 2024

Objectives

The aim of this assignment is to design and implement a networked service suitable to store log information.

Contribution to Final Grade

20 %

Managing Code

You are expected to manage your source code. A GitLab group has been created for each student, and these groups must be used to create the project to be submitted:
https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2024/assignment-3/{username}

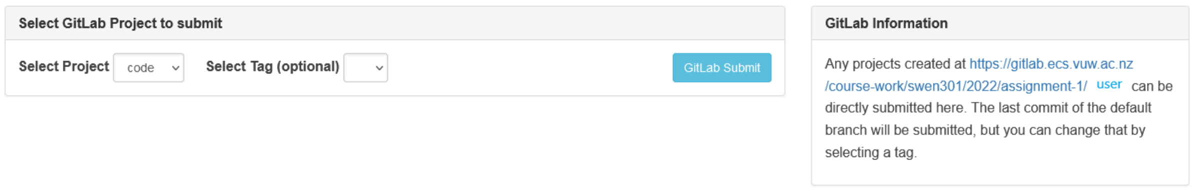
We expect that the repository is actually used, i.e. you should frequently commit. One final commit at the end is not acceptable and may be penalised.

The use of public external repositories (GitHub, Bitbucket etc) is not permitted, and if detected (using repository search and duplications with submissions detected via Moss) will be penalised.

It is expected that you use the exact names stipulated in the assignment brief. This is a requirement that will be strictly enforced (i.e., you will lose points for violations as acceptance tests will fail), in particular as the importance of naming conventions is part of the discussion in this course.

What and How to submit

There is no file upload, submission is via the gitlab-assessment system integration. It is highly recommended to create a tag for submission. In the submission system, you will see the following dialog:


Select GitLab Project to submit

Select Project code

Select Tag (optional)

GitLab Submit

GitLab Information

Any projects created at https://gitlab.ecs.vuw.ac.nz/course-work/swen301/2022/assignment-1/ user can be directly submitted here. The last commit of the default branch will be submitted, but you can change that by selecting a tag.

On submission, the project will be cloned, checked out (last commit in main or selected tag if available), and submitted.

You can (and should) verify what exactly you have submitted – the submission system will create a tag named **engr-submit-<timestamp>** for your submission.

It is expected that you verify your submission as follows:

1. **clone the repo into a new folder**
2. **checkout this tag created by the submission system (engr-submit-<timestamp>)**
3. **build your project with maven (not the IDE) and ensure that the build succeeds (compilation succeeds, tests pass)**

If you don't do this and submit the wrong code or the wrong version, the submission might be marked zero !

To be marked, the project **must be a valid Maven project**, i.e. at least the following command must succeed when executed in the project root folder on a standard ECS lab machine:

```
mvn compile
```

Other maven commands (like `mvn test` and `mvn package`) will be used during marking as well to assess particular aspects of the assignment.

Make sure that sources and `pom.xml` are all **included**, but files used and generated by IDEs (IntelliJ, Eclipse) to manage project settings and files generated by Maven during the build are **excluded**. Submit only the minimum required. There are no requirements w.r.t. IDEs / tools that can be used, all mainstream Java IDEs have Maven support built-in or supported via plugins.

Prepare

Read hints and forum posts !

Tutorials on relevant topics will be offered in weeks 9 and 10, followed by helpdesks. There are several highly relevant examples that have been / will be discussed in the lecture - analyse them for project structure, programming patterns used, dependencies, plugins and documentation: https://github.com/jensdietrich/se-teaching .

Background

The aim of this assignment is to develop HTTP-based services to store log information. This could then for instance be used as a backend for a log4j appender that stores logs remotely. These services are specified in a document using the Swapper / OpenAPI format:

https://app.swaggerhub.com/apis/jensdietrich/logstore/2.0.0

Tasks and Marking Schedule

Work **individually** to create the following program in Java 17, using the Maven project structure and layout. The following are prerequisites for your project to be marked. Failure to achieve any of those may result in zero marks:

PREREQ 1 The project must be a valid Maven project, and at least **`mvn compile`** must succeed. No additional steps (like locally installing additional dependencies before running Maven) are required.

PREREQ 2 The project must be set to use Java 17. To do this, you must set the following properties in your pom.xml:

```
<properties>
   <maven.compiler.target>17</maven.compiler.target>
   <maven.compiler.source>17</maven.compiler.source>
 </properties>
```

PREREQ 3 The server must be startable with the single mvn command **mvn jetty:run**, and stoppable with the single command **mvn jetty:stop**. When the server is running, the services must be accessible at http://localhost:8080/logstore/ (for instance: http://localhost:8080/logstore/logs) respectively.

PREREQ 4 You **must** use the jakarta version of the servlet API, i.e. use the following dependency:

```
<dependency>
   <groupId>jakarta.servlet</groupId>
   <artifactId>jakarta.servlet-api</artifactId>
```

```
    <version>6.1.0</version>
    <scope>provided</scope>
</dependency>
```

this means that you need to import `jakarta.*` packages, for examples how to set up dependencies and imports see those example repos used in lectures:

- https://github.com/jensdietrich/se-teaching/blob/main/service-whiteboxtesting/pom.xml
- https://github.com/jensdietrich/se-teaching/blob/main/servlets/pom.xml

Do **not** use (a dependency to) the old servlet API with the group id `javax.servlet`.

The services to be implemented are described in this specification: https://app.swaggerhub.com/apis/jensdietrich/restappender/2.0.0 (alternative link if swaggerhub is slow: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/SWEN301_2024T2/Assignments/a3-swagger-spec-offline.zip)

Tasks:

1. Implement the **logs/** services implemented using the standard servlet **doGet** , **doPost** and **doDelete** methods in a class **nz.ac.wgtn.swen301.a3.server.LogsServlet**. This class must have a default constructor (public, no parameters [1]). [1 mark]
   a. implement the **GET logs** service in a public method **nz.ac.wgtn.swen301.a3.server.LogsServlet::doGet** [2 marks] [2]
   b. implement **POST logs** service in a public method **nz.ac.wgtn.swen301.a3.server.LogsServlet::doPost** [1 mark]
   c. implement **DELETE logs** service in a public method **nz.ac.wgtn.swen301.a3.server.LogsServlet::doDelete** [1 marks]

   No real persistency is required, the server application should simulate a database by storing logs in a public static field **DB** defined in **nz.ac.wgtn.swen301.a3.server.Persistency** of type **java.util.List** (you are free to choose the element type of the list). You may use any of the following json libraries: json.org , gson, jackson, fastjson, jsonp. Choose a recent version, some of those may offer multiple modules (multiple Maven artifacts), use only the ones needed.
2. Implement whitebox tests using JUnit 5 in the classes **nz.ac.wgtn.swen301.a3.server.TestGetLogs**, **nz.ac.wgtn.swen301.a3.server.TestPostLogs and nz.ac.wgtn.swen301.a3.server.TestDeleteLogs** for the **logs/** services using the spring mock testing framework. The respective permitted dependencies or tests are

[1] Note that there is a default constructor that is suitable for this purpose -- https://www.tutorialspoint.com/Default-constructor-in-Java

[2] This notion means: in a method with name **doGet** in the class **LogsServlet** in package **nz.ac.wgtn.swen301.a3.server .** This methods will handle GET requests made to http://localhost:8080/logstore/logs .

(a recent version of – any version 6.* is allowed) https://mvnrepository.com/artifact/org.springframework/spring-web/ and https://mvnrepository.com/artifact/org.springframework/spring-test/). Test for the following: possible status codes, content types and data returned (by GET requests).

   a. Test cases for **nz.ac.wgtn.swen301.a3.server.LogsServlet::doGet** in **nz.ac.wgtn.swen301.a3.server.TestGetLogs** [2 marks]
   b. Test cases for **nz.ac.wgtn.swen301.a3.server.LogsServlet::doPost** in **nz.ac.wgtn.swen301.a3.server.TestPostLogs** [1 mark]
   c. Test cases for **nz.ac.wgtn.swen301.a3.server.LogsServlet::doDelete** in **nz.ac.wgtn.swen301.a3.server.TestDeleteLogs** [1 mark]

   Tests must use JUnit **assert*** methods, **not** the Java **assert** statement (this applies to other testing tasks too).

3. Implement the following **stats/*** services used to extract statistics in various formats from the log server, details are described in the specification. All classes **must** have a default constructor (public, zero parameters). The services must not set, use or rely on the **Content-Disposition** header.
   a. **GET stats/csv** in a **public** method **nz.ac.wgtn.swen301.a3.server.StatsCSVServlet::doGet** [1 marks]
   b. **GET stats/html** in a **public** method **nz.ac.wgtn.swen301.a3.server.StatsHTMLServlet::doGet** [1 marks]
   c. **GET stats/excel** in a **public** method **nz.ac.wgtn.swen301.a3.server.StatsExcelServlet::doGet** , you may use this library to generate excel documents. Use the POI **XSSF** API https://mvnrepository.com/artifact/org.apache.poi/poi/, https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml/ for this purpose. [2 marks]
4. Write whitebox tests using spring mock library (see part 1 for details) for the **stats*/** services. Test format, content type and the correctness of the data.
   a. **nz.ac.wgtn.swen301.a3.server.TestStatsCSV** to test **GET stats/csv** -- you must parse the CSV data returned in the test [1 marks]
   b. **nz.ac.wgtn.swen301.a3.server.TestStatsHTML** to test **GET stats/html** -- you must parse the HTML page returned, you may use https://mvnrepository.com/artifact/org.jsoup/jsoup/1.18.1 for this purpose [1 mark]
   c. **nz.ac.wgtn.swen301.a3.server.TestStatsXLS** to test **GET stats/excel** -- you must parse the EXCEL data returned, you may use the poi library (dependency) used for other tasks for this purpose [1 mark]
5. Write an executable class **nz.ac.wgtn.swen301.a3.client.Client.** This class accepts the following parameters (in this order) passed to **main**:
   a. type (either csv or excel) -- the type of data requested
   b. fileName -- a local file name used to store the data fetched – this can be a path (for instance /users/jens/tmp/logs.csv or c://users/jens/tmp/logs.csv)

   This program needs to use a HTTP client. The following libraries may be used: OKHttp, Apache HTTP Client or Google HTTP. Instead of using one of those libraries, you can also use the HTTP built into Java 11.
   The program will connect to a server running on http://localhost:8080/logstore/ (the

server has to be started separately, i.e. this program must not start the server), and prints an error message if the server is not available. Otherwise, it will connect to **stats/csv or stats/excel** depending on the value of the first argument (type), and store the data in a local file named fileName.
Example: **“java -cp <someclasspath> nz.ac.wgtn.swen301.a3.client.Client excel logs.xlsx”** will download log stats in excel format from http://localhost:8080/logstore/**stats/excel** **using a GET request** , and store the data in an excel file **logs.xlsx** in the current folder (for instance, **target/** if **java** was started in **target/**) [4 marks]

Hints

1. If the services cannot be started, the assignment will be marked zero, so focus on this part first.
2. The expected data format is JSON. A common misunderstanding, in particular for POST requests, is to encode this as a form with some parameter key (say “data”), and then in the servlet to pick up this data with `request.getParameter("data")`. **This is not correct !!**

Penalties

1. violating the Maven standard project layout (submissions may be marked zero if **maven compile** does not work)
2. Using local dependencies and the use of absolute file names in projects. (-1 marks for each violation)

Change History / Updates and Clarifications

date	**change**	**remarks**
20 Sept 24	changed max marks of task 5 from 3 to 4	total marks now add up to 20