Politecnico di Milano

A.A. 2015-2016

Software Engineering 2: “myTaxiService”

Code Inspection Document

Davide Azzalini (855185), Fabio Azzalini (855182)

1 Introduction

In this first chapter we give an introduction and a overall description of the whole document. In particular we present the purpose and the scope of the Code Inspection Document. We also provide some other useful information such as a list of the definitions, abbreviations and acronym used.

1.1 Purpose

This document represent the Integration Test Plan Document (ITPD). The individual software modules (components) are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the costumer.

Integration testing ensures that the unit-tested modules interact correctly.

This document is supposed to be written before the integration test actually happens.

1.2 Scope

The government of a large city aims at optimizing it's taxi service. At this stake myTaxiService will be created, a brand-new application available in two variations: web application and mobile application.

In particular, the attention will be focused on making the access to the service easier from the passengers and on guaranteeing a fairer distribution of the work load among all taxi drivers.

The application will allow passengers to request a ride and taxi drivers to decide weather or not to give the ride. Once a taxi driver has been allocated the passenger will be informed about the waiting time and the code of the incoming taxi.

Furthermore, the application will ensure a more efficient management of the service since each taxi is provided with a GPS tracker and the city is divided in zones, as a result a reduction of waiting times it is expected.

In addition to the standard taxi service (in which the costumer contacts the taxi driver in the exact moment he/she needs the service), will also be possible for the costumers to make reservations for future rides.

Important is also for the system to be developed in such a way that a future addition of some new feature can be made easily and without too much effort.

Moreover, the possibility for payments to be electronic, hence recorded, should help decrease the amount of tax evasion as all transactions can be tracked and accounted for.

1.3 List of Definitions and Abbreviations

- Integration Test Plan Document (ITPD)

1.4 List of Reference Documents

- Assignment 3 - Code Inspection.pdf

1.5 Overall Description

The following is the structure of this integration test plan document:

- In **chapter 2** are listed the class and methods assigned to our group.
- In **chapter 3** is given a brief introduction about the context in which the method operate.
- In **chapter 4** is performed the code inspection using as reference a checklist.

2 Assigned class and methods

The class assigned to our group is called “WebModule”.
Namespace: com.sun.enterprise.WebModule
Path relative to the root of GlassFish project: appserver/web/web-glue/src/main/java/com/sun/enterprise/web/WebModule.java

The methods of the WebModule class assigned to our group are the following:

- Name: start( )
  Start Line:659
- Name: callServletContainerInitializers( )
  Start Line: 772
- Name: addAdHocPathAndSubtree(String path , String subtree , AdHocServletInfo servletInfo)
  Start Line: 873
- Name: addAdHocPaths(Map < String , AdHocServletInfo > newPaths)
  Start Line: 905
- Name: addAdHocSubtrees(Map < String , AdHocServletInfo > newSubtrees)
  Start Line: 931
- Name: getAdHocServletName(String path)
  Start Line: 981

3 Functional role

3.1 Introduction to Web Applications in JEE

In the Java EE platform, web components provide the dynamic extension capabilities for a web server. Web components can be Java servlets, web pages implemented with JavaServer Faces technology, web service endpoints, or JSP pages.

Servlets are Java programming language classes that dynamically process requests and construct responses. Java technologies, such as JavaServer Faces and Facelets, are used for building interactive web applications. (Frameworks can also be used for this purpose).

Web components are supported by the services of a runtime platform called a web container. A web container provides such services as request dispatching, security, concurrency, and lifecycle management.
Certain aspects of web application behavior can be configured when the application is installed, or *deployed*, to the web container. The configuration information can be specified using Java EE annotations or can be maintained in a text file in XML format called a web application deployment descriptor (DD).

Web application developers often use third-party frameworks in their applications. To use these frameworks, developers need to register the frameworks by specifying deployment descriptors for the frameworks in the application's web.xml file.
Web fragments enable web frameworks to self-register, eliminating the need for the developer to register them through deployment descriptors.

3.2 WebModule class

The "WebModule" class is a public class that extends the "PwcWebModule" class, implements the "Context" interface and represents a web module for use by the Application Server.

According to their JavaDoc the "PwcWebModule" class represents a web module (servlet context), and the "Context" interface a web application.

The following is the JavaDoc of the class:

```
109 /**
110  * Class representing a web module for use by the Application Server.
111  */
```

In the Java EE architecture, a web module is the smallest deployable and usable unit of web resources. A web module contains web components and static web content files, such as images, which are called web resources. A Java EE web module corresponds to a web application as defined in the Java Servlet specification.

A web module can be deployed as an unpacked file structure or can be packaged in a JAR file known as a Web Archive (WAR) file. Because the contents and use of WAR files differ from those of JAR files, WAR file names use a .war extension. To deploy a WAR on the GlassFish Server, the file must contain a runtime deployment descriptor. The runtime DD is an XML file that contains such information as the context root of the web application and the mapping of the portable names of an application's resources to the GlassFish Server's resources.

The main functionalities of the class are the following:

- set a web container;
- set the default-web.xml;
- start a web module;
- stop a web module;
- manage servlet container initializers;
- manage ad-hoc paths and subtrees;
- manage ad-hoc valves and pipelines;
- configure Catalina properties;
- manage development descriptors;
- configure web services;
- configure the class loader;
- manage sessions;
- manage security aspects.

3.3 start() method

This is a public method of the class "WebModule", and it is used to start the web module.

This method overrides the start() method of the StandardContext class, a class extended by PwcWebModule.

The following are the JavaDoc and the declaration of the method:

```
655⊖    /**
656      * Starts this web module.
657      */
658⊖    @Override
659     public synchronized void start() throws LifecycleException {
```

- ServletContainerInitializer: interface which allows a library/runtime to be notified of a web application's startup phase and perform any required programmatic registration of servlets, filters, and listeners in response to it.

- webBundleDescriptor: an object that represents all the deployment information about a web application.

- AbsoluteOrderingDescriptor: deployment object representing the absolute-ordering of a web fragment.

The firs part of the method is used to get an interestList of ServletContainerInitializers.
If the webBundleDescriptor is not null we extract from it the AbsoluteOrderingDescriptor. The AbsoluteOrderingDescriptor is then used to create a list with the references to the web fragments. The last step is now to build a mapping of each web fragment with its JAR file. Now we have all the information we need to get a list of all the initializers.
ServletContainerInitializer processing in fact can be controlled per JAR file via fragment ordering. If an absolute ordering is defined, then only those JARs included in the ordering will be processed for ServletContainerInitializers.

In the second part of the method by a call to its ancestor method (the one we are overriding) we start and register Tomcat mbeans. Then Catalina listener and valves are configured. This can only happen after this web module has been started, in order to be able to load the specified listener and valves classes.

3.4 callServletContainerInitializers() method

This is a protected method of the class “WebModule”, and it is used to call the ServletContainerInitializers.

This method overrides the callServletContainerInitializers() method of the StandardContext class, a class extended by PwcWebModule.

The following is the declaration of the method, the JavaDoc is not provided.

```
771⊖    @Override
772     protected void callServletContainerInitializers()
773             throws LifecycleException {
```

- ServletContext: is a group of servlet that share data and resources. Defines a set of methods that a servlet uses to communicate with its servlet container.

- ServletContextListener: interface for receiving notification events about ServletContext lifecycle changes.

The method starts with a call to its ancestor method (the one we are overriding). We don't have the code of the ancestor method, but probably it is the one which actually performs the call, since the other part of our method is only used to remove JSF ServletContextListeners.

Then the method Remove any JSF related ServletContextListeners from non-JSF apps. This can be done reliably only after all ServletContainerInitializers have been invoked, because system-wide ServletContainerInitializers may be invoked in any order, and it is only after JSF's FacesInitializer has been invoked that isJsfApplication(), which checks for the existence of a mapping to the FacesServlet in the app, may be used reliably because such mapping would have been added by JSF's FacesInitializer.

3.5 addAdHocPathAndSubtree() method

This is a method of the class "WebModule", and it is used to add the ad-hoc Path and/or the ad-hoc Subtree.

The following is the declaration of the method.

```
863     /*
864      * Adds the given ad-hoc path and subtree, along with information about
865      * the servlet that will be responsible for servicing it, to this web
866      * module.
867      *
868      * @param path The ad-hoc path to add
869      * @param subtree The ad-hoc subtree path to add
870      * @param servletInfo Information about the servlet that is responsible
871      * for servicing the given ad-hoc path
872      */
873     void addAdHocPathAndSubtree(String path,
874                                 String subtree,
875                                 AdHocServletInfo servletInfo) {
```

The web container's ad-hoc mechanism allows any interested GlassFish component to have a URI, or an entire URI subtree, serviced by an ad-hoc servlet that is registered programmatically via the ad-hoc registration API, instead of declaratively in the web.xml.
A URI that is mapped to an ad-hoc servlet is referred to as an ad-hoc path.

Likewise, a URI subtree mapped to an ad-hoc servlet is referred to as an ad-hoc subtree.

This method adds the given ad-hoc path and subtree, along with information about the servlet that will be responsible for servicing it, to this web module.

3.6 addAdHocPaths() method

This is a method of the class “WebModule”, and it is used to add the ad-hoc Paths to servlet mappings to this web module.

The following is the declaration of the method.

```
899    /*
900     * Adds the given ad-hoc path to servlet mappings to this web module.
901     *
902     * @param newPaths Mappings of ad-hoc paths to the servlets responsible
903     * for servicing them
904     */
905    void addAdHocPaths(Map<String, AdHocServletInfo> newPaths) {
```

Ad-hoc Path: is a servlet path that is mapped to a servlet not declared in the web module's deployment descriptor.

While addAdHocPathAndSubtree() method can only add one ad-hoc path, this method can add more paths at the same time.

3.7 addAdHocSubtrees() method

This is a method of the class “WebModule”, and it is used to add the ad-hoc Subtrees paths to servlet mappings to this web module.

The following is the declaration of the method.

```
924    /*
925     * Adds the given ad-hoc subtree path to servlet mappings to this web
926     * module.
927     *
928     * @param newSubtrees Mappings of ad-hoc subtree paths to the servlets
929     * responsible for servicing them
930     */
931    void addAdHocSubtrees(Map<String, AdHocServletInfo> newSubtrees) {
```

Ad-hoc Subtree: is a servlet subtree path that is mapped to a servlet not declared in the web module's deployment descriptor.

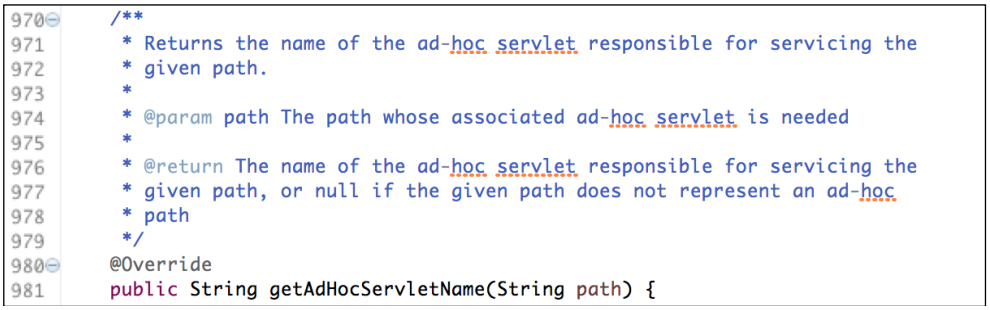
While addAdHocPathAndSubtree() method can only add one ad-hoc subtree path, this method can add more subtree paths at the same time.

3.8 getAdHocServletName() method

This is a method of the class “WebModule”, and returns the name of the ad-hoc servlet responsible for servicing the given path.

This method override the getAdHocServletName() method of the StandardContext class, a class extended by PwcWebModule.

The following are the JavaDoc and the declaration of the method:



```
970     /**
971      * Returns the name of the ad-hoc servlet responsible for servicing the
972      * given path.
973      *
974      * @param path The path whose associated ad-hoc servlet is needed
975      *
976      * @return The name of the ad-hoc servlet responsible for servicing the
977      * given path, or null if the given path does not represent an ad-hoc
978      * path
979      */
980     @Override
981     public String getAdHocServletName(String path) {
```

The method begins by checking if the given path matches with any of the ad-hoc paths (exact match) then check if the given path starts with any of the ad-hoc subtree paths. And then the method return the servlet name, if any, associated to our path.

4 Code Inspection

4.1 start() method

```
655     /**
656      * Starts this web module.
657      */
658     @Override
659     public synchronized void start() throws LifecycleException {
660         // Get interestList of ServletContainerInitializers present, if any.
661         List<Object> orderingList = null;
662         boolean hasOthers = false;
663         Map<String, String> webFragmentMap = Collections.emptyMap();
664         if (webBundleDescriptor != null) {
665             AbsoluteOrderingDescriptor aod =
666                     ((WebBundleDescriptorImpl)webBundleDescriptor).getAbsoluteOrderingDescriptor();
667             if (aod != null) {
668                 orderingList = aod.getOrdering();
669                 hasOthers = aod.hasOthers();
670             }
671             webFragmentMap = webBundleDescriptor.getJarNameToWebFragmentNameMap();
672         }
673 
674         Iterable<ServletContainerInitializer> allInitializers =
675             ServletContainerInitializerUtil.getServletContainerInitializers(
676                 webFragmentMap, orderingList, hasOthers,
677                 wmInfo.getAppClassLoader());
678         setServletContainerInitializerInterestList(allInitializers);
679 
680         DeploymentContext dc = getWebModuleConfig().getDeploymentContext();
681         if (dc != null) {
682             directoryDeployed =
683                     Boolean.valueOf(dc.getAppProps().getProperty(ServerTags.DIRECTORY_DEPLOYED));
684         }
685         if (webBundleDescriptor != null) {
686             showArchivedRealPathEnabled = webBundleDescriptor.isShowArchivedRealPathEnabled();
687             servletReloadCheckSecs = webBundleDescriptor.getServletReloadCheckSecs();
688         }
689 
690         // Start and register Tomcat mbeans
691         super.start();
```

```
692 
693         // Configure catalina listeners and valves. This can only happen
694         // after this web module has been started, in order to be able to
695         // load the specified listener and valve classes.
696         configureValves();
697         configureCatalinaProperties();
698         webModuleStartedEvent();
699         if (directoryListing) {
700             setDirectoryListing(directoryListing);
701         }
702 
703         hasStarted = true;
704     }
705 
```

1. All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests:

- The variable at line 662 has not a meaningful name.
- The object at line 665 has not a meaningful name

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block):

- The method respect the K&R style.

15. Line break occurs after a comma or an operator:

- The instruction at line 675 violate this rule.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line:

- The instruction at line 676 is non correctly aligned.
- The instruction at line 677 is non correctly aligned.

23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you):

- Missing exception explanation in the javadoc.

33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces "{" and "}"). The exception is a variable can be declared in a 'for' loop:

- The declaration at line 674 is not at the beginning of a block.
- The declaration at line 680 is not at the beginning of a block.

4.2 callServletContainerInitializers() method

```
771     @Override
772     protected void callServletContainerInitializers()
773             throws LifecycleException {
774         super.callServletContainerInitializers();
775         if (!isJsfApplication() && !contextListeners.isEmpty()) {
776             /*
777              * Remove any JSF related ServletContextListeners from
778              * non-JSF apps.
779              * This can be done reliably only after all
780              * ServletContainerInitializers have been invoked, because
781              * system-wide ServletContainerInitializers may be invoked in
782              * any order, and it is only after JSF's FacesInitializer has
783              * been invoked that isJsfApplication(), which checks for the
784              * existence of a mapping to the FacesServlet in the app, may
785              * be used reliably because such mapping would have been added
786              * by JSF's FacesInitializer. See also IT 10223
787              */
788             ArrayList<ServletContextListener> listeners =
789                 new ArrayList<ServletContextListener>(contextListeners);
790             String listenerClassName = null;
791             for (ServletContextListener listener : listeners) {
792                 if (listener instanceof
793                         StandardContext.RestrictedServletContextListener) {
794                     listenerClassName = ((StandardContext.RestrictedServletCor
795                 } else {
796                     listenerClassName = listener.getClass().getName();
797                 }
798                 /*
799                  * TBD: Retrieve listener class name from JSF's TldProvider
800                  */
801                 if ("com.sun.faces.config.ConfigureListener".equals(
802                         listenerClassName)) {
803                     contextListeners.remove(listener);
804                 }
805             }
806         }
807     }
808
```

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block):

- The "else" statement at line 795 should be in a new line.

14.When line length must exceed 80 characters, it does NOT exceed 120 characters:

- The instruction at line 794 exceeds 120 characters.

15. Line break occurs after a comma or an operator:

- The declaration of the method at line 772 violate this rule.
- The instruction at line 793 violate this rule.
- The instruction at line 802 violate this rule.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

- The instruction at line 795 is non correctly aligned.

4.3 addAdHocPathAndSubtree() method

```
863     /*
864      * Adds the given ad-hoc path and subtree, along with information about
865      * the servlet that will be responsible for servicing it, to this web
866      * module.
867      *
868      * @param path The ad-hoc path to add
869      * @param subtree The ad-hoc subtree path to add
870      * @param servletInfo Information about the servlet that is responsible
871      * for servicing the given ad-hoc path
872      */
873     void addAdHocPathAndSubtree(String path,
874                                 String subtree,
875                                 AdHocServletInfo servletInfo) {
876
877         if (path == null && subtree == null) {
878             return;
879         }
880
881         Wrapper adHocWrapper = (Wrapper)
882             findChild(servletInfo.getServletName());
883         if (adHocWrapper == null) {
884             adHocWrapper = createAdHocWrapper(servletInfo);
885             addChild(adHocWrapper);
886         }
887
888         if (path != null) {
889             adHocPaths.put(path, servletInfo);
890             hasAdHocPaths = true;
891         }
892
893         if (subtree != null) {
894             adHocSubtrees.put(subtree, servletInfo);
895             hasAdHocSubtrees = true;
896         }
897     }
898
```

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block):

- The method respect the K&R style.

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods):

- There is no need of a blank line at line 876.

14.When line length must exceed 80 characters, it does NOT exceed 120 characters:

- The declaration of the method at line 873, 874 and 875 could have been arranged on the same line.
- The instructions at line 881 and 882 could have been arranged on the same line.

15. Line break occurs after a comma or an operator:

- The instruction at line 882 violate this rule.

33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces "{" and "}"). The exception is a variable can be declared in a 'for' loop:

- The declaration at line 881 is not at the beginning of a block.

4.4 addAdHocPaths() method

```
899     /*
900      * Adds the given ad-hoc path to servlet mappings to this web module.
901      *
902      * @param newPaths Mappings of ad-hoc paths to the servlets responsible
903      * for servicing them
904      */
905     void addAdHocPaths(Map<String, AdHocServletInfo> newPaths) {
906
907         if (newPaths == null || newPaths.isEmpty()) {
908             return;
909         }
910         for (Map.Entry<String, AdHocServletInfo> entry : newPaths.entrySet()) {
911             AdHocServletInfo servletInfo = entry.getValue();
912             Wrapper adHocWrapper = (Wrapper)
913                 findChild(servletInfo.getServletName());
914             if(adHocWrapper == null) {
915                 adHocWrapper = createAdHocWrapper(servletInfo);
916                 addChild(adHocWrapper);
917             }
918             adHocPaths.put(entry.getKey(), servletInfo);
919         }
920
921         hasAdHocPaths = true;
922     }
```

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block):

- The method respect the K&R style.

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods):

- There is no need of a blank line at line 906.
- There is no need of a blank line at line 920.

14.When line length must exceed 80 characters, it does NOT exceed 120 characters:

- The instructions at line 912 and 913 could have been arranged on the same line.

15. Line break occurs after a comma or an operator:

- The instruction at line 913 violate this rule.

4.5 addAdHocSubtrees() method

```
924     /*
925      * Adds the given ad-hoc subtree path to servlet mappings to this web
926      * module.
927      *
928      * @param newSubtrees Mappings of ad-hoc subtree paths to the servlets
929      * responsible for servicing them
930      */
931     void addAdHocSubtrees(Map<String, AdHocServletInfo> newSubtrees) {
932
933         if (newSubtrees == null || newSubtrees.isEmpty()) {
934             return;
935         }
936         for (Map.Entry<String, AdHocServletInfo> entry : newSubtrees.entrySet()) {
937             AdHocServletInfo servletInfo = entry.getValue();
938             Wrapper adHocWrapper = (Wrapper)findChild(servletInfo.getServletName());
939             if(adHocWrapper == null) {
940                 adHocWrapper = createAdHocWrapper(servletInfo);
941                 addChild(adHocWrapper);
942             }
943             adHocSubtrees.put(entry.getKey(), servletInfo);
944         }
945
946         hasAdHocSubtrees = true;
947     }
948
```

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block):

- The method respect the K&R style.

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods):

- There is no need of a blank line at line 932.
- There is no need of a blank line at line 945.

4.6 getAdHocServletName() method

```
970     /**
971      * Returns the name of the ad-hoc servlet responsible for servicing the
972      * given path.
973      *
974      * @param path The path whose associated ad-hoc servlet is needed
975      *
976      * @return The name of the ad-hoc servlet responsible for servicing the
977      * given path, or null if the given path does not represent an ad-hoc
978      * path
979      */
980     @Override
981     public String getAdHocServletName(String path) {
982
983         if (!hasAdHocPaths() && !hasAdHocSubtrees()) {
984             return null;
985         }
986
987         AdHocServletInfo servletInfo = null;
988
989         // Check if given path matches any of the ad-hoc paths (exact match)
990         if (path == null) {
991             servletInfo = adHocPaths.get("");
992         } else {
993             servletInfo = adHocPaths.get(path);
994         }
995
996         // Check if given path starts with any of the ad-hoc subtree paths
997         if (servletInfo == null && path != null && hasAdHocSubtrees()) {
998             for(String adHocSubtree : adHocSubtrees.keySet()) {
999                 if(path.startsWith(adHocSubtree)) {
1000                     servletInfo = adHocSubtrees.get(adHocSubtree);
1001                     break;
1002                 }
1003             }
1004         }
1005
1006         if (servletInfo != null) {
1007             return servletInfo.getServletName();
1008         } else {
1009             return null;
1010         }
1011     }
```

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block):

- The "else" statement at line 1008 should be in a new line.

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods):

- There is no need of a blank line at line 982.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

- The instruction at line 1008 is non correctly aligned.

33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces "{" and "}"). The exception is a variable can be declared in a 'for' loop:

- The declaration at line 987 is not at the beginning of a block.

4.7 Other problems found

- Line 790: the assignment to the local variable "listenerClassName" is useless and should be removed.
- Line 987: the assignment to the local variable "servletInfo" is useless and should be removed.
- Line 998: the iteration should be done over "entrySet" instead of "keySet".

5 Appendix

5.1 References

- JEE 7 Documentation “https://docs.oracle.com/javaee/7/JEETT.pdf”.

5.2 Hours of work

- Davide Azzalini: 15 hours.
- Fabio Azzalini: 15 hours.