How to set up a Google AppEngine webapp with JSF 2.2, JPA 2.0 and Dependency Injection features? [on hold]

Title says it all. This post is more for helping other Java programmers to run those technologies in Google AppEngine (SDK v1.9.48 by the time this post was typed in):

Frameworks:

- Datanucleus 3.1.1 (JPA 2.0)
- Oracle Mojarra 2.2.4 (JSF 2.2).
- Google Guice 4.0 (DI 1.0)

About JSF, don't forget the workaround for this session data loss bug

This is how I got it to work:

The most important configuration is in web.xml. The JSF initialization **MUST RUN FIRST**: I found out that `com.sun.faces.config.ConfigureListener` is in charge of that step and it always looks for the `FacesServlet` declaration. Since JSF requests **MUST** be served by Guice with a `FacesHttpServlet` wrapper (which I'll post later) in order to enable DI, then:

I declared the `FacesServlet` WITHOUT `<servlet-mapping>`s (I figured out that step by trial-error coding).

It's only declared to initialize the `FacesContextFactory`. This is the MUST-HAVE structure of the web.xml:

```
            <?xml version="1.0" encoding="utf-8"?>
            <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
                xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
                <display-name>BrochureForce</display-name>

                <description>Purchase orders generator configured to run on the Google
AppEngine.</description>

                <context-param>
                    <description>Project stage (Development or Production)</description>
                    <param-name>javax.faces.PROJECT_STAGE</param-name>
                    <param-value>Development</param-value>
                </context-param>

                <context-param>
                    <description>
                                Designate client-side state saving, since GAE doesn't
handle
                                server side (JSF default) state management.
                    </description>
                    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
                    <param-value>client</param-value>
                </context-param>

                <context-param>
                    <description>Sets the default suffix for JSF pages to
.xhtml</description>
                    <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
                    <param-value>.xhtml</param-value>
```

```
                </context-param>

                <context-param>
                    <description>
                            When enabled, the runtime initialization and default
ResourceHandler
                            implementation will use threads to perform their functions.
Set this
                            value to false if threads aren't desired (as in the case of
single-threaded
                            environments such as the Google AppEngine).
                            Note that when this option is disabled, the ResourceHandler
will not
                            pick up new versions of resources when ProjectStage is
development.
                        </description>
                    <param-name>com.sun.faces.enableThreading</param-name>
                    <param-value>false</param-value>
                </context-param>

                <context-param>
                    <description>Allows dependency-injection into
ManagedBeans</description>
                    <param-name>com.sun.faces.injectionProvider</param-name>
                    <param-value>mypackage.jsf.JsfInjectionProvider</param-value>
                </context-param>

                <context-param>
                    <description>Specify JBoss Expression Language Over
Default</description>
                    <param-name>com.sun.faces.expressionFactory</param-name>
                    <param-value>org.jboss.el.ExpressionFactoryImpl</param-value>
                </context-param>

                <!-- JSF INITIALIZATION GOES FIRST!! -->
                <servlet>
                    <description>
                            JSF 2 Servlet. There's NO servlet-mapping defined for this
servlet because
                            it's declared here in order to enforce the FacesFactory to
load properly
                            so that an instance of this servlet can be injected in the
FacesHttpServlet
                            used by Guice to serve JSF requests and as injection provider
at the same time.
                            Furthermore, the "load-on-startup" property is set to "0" to
tell Jetty
                            that this servlet MUST be loaded first.
                    </description>
                    <servlet-name>JSF Servlet</servlet-name>
                    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
                    <load-on-startup>0</load-on-startup>
                </servlet>
                <listener>
                    <description>JSF Initialization.</description>
                    <listener-class>com.sun.faces.config.ConfigureListener</listener-
class>
                </listener>
                <!-- JSF INITIALIZATION GOES FIRST!! -->

                <listener>
                    <description>PERSISTENCE ENGINE INITIALIZATION AND SHUTDOWN.
</description>
                    <listener-
class>mypackage.listener.PersistenceManagerSetupListener</listener-class>
                </listener>

                <!-- ***** Specify session timeout of thirty (30) minutes. ***** -->
                <session-config>
                    <session-timeout>30</session-timeout>
                </session-config>

                <welcome-file-list>
                    <welcome-file>index.jsf</welcome-file>
                    <welcome-file>index.xhtml</welcome-file>
                </welcome-file-list>
                <!-- ****************************************************** -->
                <!-- DI API initialization (Google Guice Implementation). -->
                <!-- ****************************************************** -->
                <filter>
                    <description>Google Guice filter which enables DI.</description>
                    <filter-name>GuiceFilter</filter-name>
                    <filter-class>com.google.inject.servlet.GuiceFilter</filter-class>
                </filter>
                <filter-mapping>
                    <filter-name>GuiceFilter</filter-name>
                    <url-pattern>/*</url-pattern>
                </filter-mapping>
                <listener>
                    <description>
                                This listener initializes the Guice injector and wraps the
JSF Servlet
                                into a HttpServlet in order to serve JSF requests via
Guice Filter.
```

```
            </description>
            <listener-class>mypackage.listener.GuiceListener</listener-class>
        </listener>
        <!-- ********************************************** -->
    </web-app>
```

Second, I'm not trying to inject a managed bean instance into another anymore. Instead, I'm sharing a bound business logic instance into the beans (in other words, emulating EJB behavior). This is what I did:

1. I defined a `@BindingAnnotation` for the business logic implementation:

```
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import com.google.inject.BindingAnnotation;

@Documented
@BindingAnnotation
@Retention(RUNTIME)
@Target({ TYPE })
public @interface BusinessLogic {}
```

2. I defined a business logic interface with its implementation and annotated both with the `@BusinessLogic` annotation (This is an example that registers a visit made to the page. The fields are: the visit number, the source IP and the timestamp):

```
import java.util.List;
import mypackage.annotation.BusinessLogic;
import mypackage.dataaccess.entity.Visit;

@BusinessLogic
public interface VisitsHandler {
    public void insertVisit();
    public List<Visit> getPageVisits();

    // Propiedades
    public String getCurrentVisit();
    public void setCurrentVisit(String currentVisit);
}
```

and its implementation:

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import mypackage.annotation.BusinessLogic;
import mypackage.jsf.logic.VisitsHandler;
import mypackage.dataaccess.PersistenceManager;
import mypackage.dataaccess.Queries;
import mypackage.dataaccess.entity.Visit;

@BusinessLogic
public class VisitsHandlerImpl implements VisitsHandler {
    private String currentVisit;

    public void insertVisit() {
        PersistenceManager pMgr = PersistenceManager.getInstance();
        Visit newVisit = new Visit();
        newVisit.setUserIp("127.0.0.1");
        newVisit.setTimestamp(new Date(System.currentTimeMillis()));
        pMgr.insert(newVisit);
        pMgr = null; // Dereference the singleton instance.
        this.currentVisit = newVisit.toString();
    }

    @SuppressWarnings("rawtypes")
    public List<Visit> getPageVisits() {
        PersistenceManager pMgr = PersistenceManager.getInstance();
        List<Visit> visitsList = new ArrayList<Visit>();
        List visits = pMgr.executeJpqlQuery(Queries.JPQL_VISITS);
        for (Object v : visits) {
            visitsList.add((Visit) v);
        }
        pMgr = null; // Dereference the singleton instance.
        return visitsList;
    }

    /**
     * @return the currentVisit
     */
    public String getCurrentVisit() {
        return currentVisit;
    }

    /**
     * @param currentVisit
     *            the currentVisit to set
     */
    public void setCurrentVisit(String currentVisit) {
        this.currentVisit = currentVisit;
```

```
            }
        }
```

To avoid reinstantiation of the business logic objects, I defined a single instance for the DI binding:

```
        import mypackage.jsf.logic.VisitsHandler;
        import mypackage.jsf.logic.impl.VisitsHandlerImpl;
        interface InjectorConstants {

            // Url patterns for FacesServlet, as it would be defined in web.xml
            static String[] JSF_SERVLET_URL_PATTERNS = new String[] { "*.jsf",
"*.xhtml" };

            // BUSINESS LOGIC OBJECTS.
            static Class<VisitsHandler> VISITS_HANDLER = VisitsHandler.class;
            static VisitsHandler VISITS_HANDLER_IMPL = new VisitsHandlerImpl();
        }
```

Now, the Guice module with the object bindings:

```
        import javax.faces.webapp.FacesServlet;
        import javax.inject.Singleton;

        import mypackage.cdi.annotation.ViewScoped;
        import mypackage.cdi.annotation.ViewScopedImpl;
        import mypackage.cdi.listener.PostConstructTypeListener;
        import mypackage.jsf.FacesHttpServlet;
        import com.google.inject.matcher.Matchers;
        import com.google.inject.servlet.ServletModule;

        public class JSFModule extends ServletModule {
            private void businessLogicBindings() {

bind(InjectorConstants.VISITS_HANDLER).toInstance(InjectorConstants.VISITS_HANDLER_IMPL);
            }

            private void systemBindings() {
                // Add support for the @PostConstruct annotation for Guice-injected
                // objects.
                bindListener(Matchers.any(), new PostConstructTypeListener(null));

                // Binding a custom implementation of "@ViewScoped" scope.
                bindScope(ViewScoped.class, new ViewScopedImpl());
            }

            private void jsfBindings() {
                // Define and bind FacesServlet as singleton object
                // so it can be injected in FacesHttpServlet's constructor.
                bind(FacesServlet.class).in(Singleton.class);

                // JSF patterns to be served by FacesHttpServlet.
                for (String urlPattern : InjectorConstants.JSF_SERVLET_URL_PATTERNS) {
                    serve(urlPattern).with(FacesHttpServlet.class);
                }
            }

            @Override
            protected void configureServlets() {
                // Guice injector bindings.
                this.systemBindings();
                this.businessLogicBindings();
                this.jsfBindings();
            }
        }
```

The `businessLogicBindings()` method associates the business logic interface with the implementation instance. On the other hand, you can see on this line: `serve(urlPattern).with(FacesHttpServlet.class);` , Guice will reroute JSF requests to a HttpServlet wrapper with an injected `FacesServlet` instance:

```
        import java.io.IOException;
        import javax.faces.webapp.FacesServlet;
        import javax.inject.Inject;
        import javax.inject.Singleton;
        import javax.servlet.Servlet;
        import javax.servlet.ServletConfig;
        import javax.servlet.ServletException;
        import javax.servlet.ServletRequest;
        import javax.servlet.ServletResponse;
        import javax.servlet.http.HttpServlet;
        import javax.servlet.http.HttpServletRequest;
        import javax.servlet.http.HttpServletResponse;

        @Singleton
        public class FacesHttpServlet extends HttpServlet {

            private static final long serialVersionUID = 1L;

            private final Servlet facesServlet;

            @Inject
            public FacesHttpServlet(FacesServlet facesServlet) {
                this.facesServlet = facesServlet;
```

```
                }

                @Override
                public void init(ServletConfig config) throws ServletException {
                    this.facesServlet.init(config);
                }

                @Override
                public ServletConfig getServletConfig() {
                    return this.facesServlet.getServletConfig();
                }

                @Override
                public String getServletInfo() {
                    return this.facesServlet.getServletInfo();
                }

                @Override
                public void destroy() {
                    super.destroy();
                    this.facesServlet.destroy();
                }

                @Override
                public void service(ServletRequest req, ServletResponse resp) throws
ServletException, IOException {
                    HttpServletRequest httpReq = (HttpServletRequest) req;
                    String reqUrl = httpReq.getRequestURL().toString();
                    // A hack to redirect the index page. It's been throwing an error if
the
                    // "/index.[xhtml|jsf]" is not explicitly specified in the request
URL.
                    if(reqUrl.toLowerCase().endsWith("index.xhtml")) {
                        ((HttpServletResponse)
resp).sendRedirect(reqUrl.replace("index.xhtml", "index.jsf"));
                    } else {
                        this.facesServlet.service(req, resp);
                    }
                }
            }
```

Now, the listener that initializes the injector:

```
            import java.util.HashMap;
            import mypackage.cdi.JSFModule;
            import mypackage.cdi.JsfInjectionProvider;
            import com.google.inject.AbstractModule;
            import com.google.inject.Guice;
            import com.google.inject.Injector;
            import com.google.inject.servlet.GuiceServletContextListener;

            public class GuiceListener extends GuiceServletContextListener {
                protected AbstractModule module;
                protected static Injector injector;
                private static HashMap<String, Object> instancesMap;

                public GuiceListener() {
                    // Bean instance list to ensure that we inject a unique bean instance.
                    instancesMap = new HashMap<>();

                    // Create the injector.
                    injector = Guice.createInjector(new JSFModule());
                }

                @Override
                public Injector getInjector() {
                    return injector;
                }

                /**
                 * given a class, generates an injected instance. Useful when an API call
is
                 * needed internally.
                 */
                public static <T> T getInstance(Class<T> type) {
                    return injector.getInstance(type);
                }

                /**
                 * given an injectable instance, injects its dependencies and make sure to
                 * only inject one.
                 */
                public static void injectMembers(Object instance) {
                    Object obj = null;
                    if (JsfInjectionProvider.isBusinessLogicObject(obj)) {
                        String instanceClassName = instance.getClass().getName();
                        Object mappedInstance = instancesMap.get(instanceClassName);
                        if (mappedInstance == null) {
                            // It's a new bean instance. It's stored in the bean map
                            // to be able to reuse it.
                            instancesMap.put(instanceClassName, instance);
                            obj = instance;
                        } else {
                            // There's already a bean instance. Let's reuse it!.
```

```
                        obj = mappedInstance;
                    }
                } else { // it should be a managed bean.
                    obj = instance;
                }
                injector.injectMembers(obj);
            }
        }
```

Last, but not least, Mojarra must register our DI implementation as its DI provider (see the `<context-param>` `com.sun.faces.injectionProvider` value):

```
        import javax.faces.bean.ManagedBean;
        import mypackage.cdi.annotation.BusinessLogic;
        import mypackage.cdi.listener.GuiceListener;
        import com.sun.faces.spi.InjectionProviderException;
        import com.sun.faces.vendor.WebContainerInjectionProvider;

        public class JsfInjectionProvider extends WebContainerInjectionProvider {
            @Override
            public void inject(Object obj) throws InjectionProviderException {
                if (isManagedBean(obj) || isBusinessLogicObject(obj)) {
                    GuiceListener.injectMembers(obj);
                }
            }

            /**
             * As an arbitrary choice, the choice here is to inject only into
             * {@code @ManagedBean} instances, so that other classes - not written by
us
             * - wouldn't be injected too. This choice could be altered.
             *
             * @param obj
             *            A JSF bean instance (annotated with @ManagedBean).
             * @return
             */
            private boolean isManagedBean(Object obj) {
                return obj != null && obj.getClass().getAnnotation(ManagedBean.class)
!= null;
            }

            public static boolean isBusinessLogicObject(Object obj) {
                return obj != null &&
obj.getClass().getAnnotation(BusinessLogic.class) != null;
            }
        }
```

All of this working altogether (yet ommitting the JPA part, which is not relevant at this point): `ExampleBean`:

```
        import java.io.Serializable;
        import java.util.List;

        import javax.annotation.PostConstruct;
        import javax.faces.bean.ManagedBean;
        import javax.inject.Inject;

        import mypackage.jsf.logic.VisitsHandler;
        import mypackage.dataaccess.entity.Visit;

        @ManagedBean(name="jsfbExample")
        public class ExampleBean implements Serializable {

            private static final long serialVersionUID = 1L;

            @Inject
            private VisitsHandler visitsHandler;

            @PostConstruct
            public void init() {
                System.out.println("ExampleBean - Injection works! visitsHandler = " +
visitsHandler); // It works.
            }

            /**
             * Method to test EL engine processing with parameters.
             * @param param
             * @return
             */
            public void insertVisit() {
                this.visitsHandler.insertVisit();
            }

            public List<Visit> getPageVisits() {
                return this.visitsHandler.getPageVisits();
            }


            /**
             * @return the currentVisit
             */
            public String getCurrentVisit() {
                return this.visitsHandler.getCurrentVisit();
            }
```

```
    /**
     * @param currentVisit
     *            the currentVisit to set
     */
    public void setCurrentVisit(String currentVisit) {
        this.visitsHandler.setCurrentVisit(currentVisit);
    }
}
```

Now, you can create a *.xhtml file as your inxdex and put this testing code on it:

```
<!DOCTYPE html
         PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:ui="http://java.sun.com/jsf/facelets">
<h:head id="head">
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Welcome to JSF 2.1 on the Google AppEngine!</title>
</h:head>
<h:body>
        <h:form>
            <h:outputText id="lastVisit" value="#{jsfbExample.currentVisit}"
/><br/>
            <h:commandButton value="New visit!"
                actionListener="#{jsfbExample.insertVisit()}">
                <f:ajax execute="@this" render="pageVisitsList" />
            </h:commandButton>
            <h:commandButton value="Last inserted visit!">
                <f:ajax execute="@this" render="lastVisit" />
            </h:commandButton>
            <h:panelGrid id="pageVisitsList">
                <c:forEach var="visit" items="#{jsfbExample.pageVisits}">
                    <h:outputText value="#{visit.toString()}" />
                </c:forEach>
            </h:panelGrid>
        </h:form>
</h:body>
</html>
```

JPA feature is easier since its configuration neither depends on JSF nor DI. `PersistenceManagerSetupListener`:

```
package mypackage.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import mypackage.dataaccess.PersistenceManager;
import mypackage.utils.StringMap;

public class PersistenceManagerSetupListener implements ServletContextListener
{

    @Override
    public void contextInitialized(ServletContextEvent servletContextInitEvt)
{

      // This is only a wrapper over HashMap<String, String>
        StringMap initProperties = new StringMap();

        // Check the System properties to determine if we are running on cloud
        // or not, and set up the JDBC driver accordingly.
        String platform =
System.getProperty("com.google.appengine.runtime.version").toLowerCase()
                .contains("google app engine") ? "cloud" : "dev";
        initProperties.put("datanucleus.ConnectionURL",
System.getProperty(platform + ".db.url"));
        initProperties.put("datanucleus.ConnectionDriverName",
System.getProperty(platform + ".db.driver"));
        initProperties.put("datanucleus.ConnectionUserName",
System.getProperty(platform + ".db.user"));
        initProperties.put("datanucleus.ConnectionPassword",
System.getProperty(platform + ".db.password"));
        // I implemented password encryption. See Datanucleus'
"ConnectionEncryptionProvider" interface documentation.
       initProperties.put("datanucleus.ConnectionPasswordDecrypter",
                System.getProperty(platform + ".db.encryptionProvider"));

        //
**********************************************************************************

        // THESE 2 ARE A MUST-HAVE!!!
        //
**********************************************************************************

        initProperties.put("datanucleus.identifier.case",
System.getProperty("persistencemanager.identifier.case"));
        initProperties.put("datanucleus.storeManagerType",
System.getProperty("persistencemanager.storeManagerType"));
        //
**********************************************************************************
```

```
                    initProperties.put("datanucleus.NontransactionalRead",
System.getProperty("persistencemanager.NontransactionalRead"));
                    initProperties.put("datanucleus.NontransactionalRead",
System.getProperty("persistencemanager.NontransactionalRead"));
                    initProperties.put("datanucleus.NontransactionalWrite",
System.getProperty("persistencemanager.NontransactionalWrite"));
                    initProperties.put("datanucleus.singletonEMFForName",
                            System.getProperty("persistencemanager.singletonEMFForName"));
                    initProperties.put("javax.persistence.query.timeout",
System.getProperty("persistencemanager.query.timeout"));
                    initProperties.put("datanucleus.datastoreWriteTimeout",
System.getProperty("persistencemanager.datastoreWriteTimeout"));

                    // Initialize persistence engine.
                    PersistenceManager.initialize(initProperties);
                }

                @Override
                public void contextDestroyed(ServletContextEvent
servletContextDestroyedEvt) {
                    PersistenceManager.shutdown();
                }
            }
```

All the persistence init properties are defined in `app-engine.xml`. Its basic structure:

```
            <appengine-web-app ...>
                <application>cloud-project-id</application>
                <version>1</version>
                <threadsafe>true</threadsafe>
                <system-properties>
                    <!-- Cloud platform properties (their name starts with "cloud") -->
                    <property name="cloud.db.url"
                        value="jdbc:google:mysql://(cloud-connection-name)/(db-name)" />
                    <property name="cloud.db.driver"
                        value="com.google.appengine.api.rdbms.AppEngineDriver" />
                        <!--  ...  -->
                    <!-- Dev platform properties (their name starts with "dev") -->
                    <property name="dev.db.url" value="jdbc:mysql://(db-server):(db-
port)/(db-name)" />
                    <property name="dev.db.driver" value="com.mysql.jdbc.Driver" />
                        <!--  ...  -->
                    <!-- Datanucleus properties -->
                    <!-- ********************************************** -->
                    <!-- THESE 2 ARE A MUST-HAVE!!! Others are optional -->
                    <!-- ********************************************** -->
                    <property name="persistencemanager.storeManagerType" value="rdbms" />

                    <!-- This means that all DB identifiers MUST be defined in lowercase.
-->
                    <property name="persistencemanager.identifier.case" value="LowerCase"
/>
                    <!-- ********************************************** -->
                        <!--  ...  -->
                </system-properties>
                <sessions-enabled>true</sessions-enabled>
                <async-session-persistence enabled="false" />
                <static-files>
                    <exclude path="/**.xhtml" />
                </static-files>
            </appengine-web-app>
```

You must define at least one persistence unit (in "persistence.xml"):

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

                <persistence-unit name="<MyPersistenceUnit">
                    <!-- DATANUCLEUS' JPA 2.0 PERSISTENCE PROVIDER CLASS -->
                    <provider>org.datanucleus.api.jpa.PersistenceProviderImpl</provider>

                    <!-- ENTITY CLASSES -->
                    <class>mypackage.dataaccess.entity.Visit</class>

                    <!-- DON'T PROCESS UNLISTED CLASSES AS ENTITY CLASSES. -->
                    <exclude-unlisted-classes>true</exclude-unlisted-classes>
                </persistence-unit>
            </persistence>
```

and some initialize and shutdown methods in your persistence manager object(s) to create and destroy the EntityManagerFactory and the EntityManager(s). Something like this:

```
                public static void initialize(Map properties) {
                    if (!isInitialized) {
                        if (properties == null) {
                            emfInstance =
Persistence.createEntityManagerFactory("MyPersistenceUnit");
                        } else {
                            emfInstance =
Persistence.createEntityManagerFactory("MyPersistenceUnit", properties);
                        }
                        emInstance = emfInstance.createEntityManager();
                        isInitialized = true;
                    }
                }

                public static void shutdown() {
                    try {
                        emInstance.close();
                    } catch (Exception e) {}
                    try {
                        emfInstance.close();
                    } catch (Exception e) {}
                }
```

The "Visit" class is just an Entity class which maps the 3 fields (Number of visit, source IP and timestamp) and it's registered in the "persistence.xml" file.

This is the list of libraries (at least half of them come bundled in GAE SDK)

Hope this guide can help others to create great J2EE apps in GAE.

java google-app-engine jpa jsf-2 guice

edited 8 mins ago

asked 23 hours ago

iherazo

5 ● 4

put on hold as unclear what you're asking by Neil Stockton, Modus Tollens, Martijn Pieters ♦ 8 hours ago

Please clarify your specific problem or add additional details to highlight exactly what you need. As it's currently written, it’s hard to tell exactly what you're asking. See the How to Ask page for help clarifying this question.

If this question can be reworded to fit the rules in the help center, please edit the question.

5 According to the guidelines of this site you should post a question in the question part. Then answer your own question in the answer part. In any event this type of posts (i.e. helping others) is generally encouraged. – MouseEvent 22 hours ago

You may have to wait a few hours since posting before SO allow's answering own post. – MouseEvent 22 hours ago

3 Please post your guide as a self answered question. Posts on SO must follow the question/answer pattern. – Modus Tollens 16 hours ago

@Modus Tollens I've already clarified this is not a question but a guide to help others. How / where should I post that clarification then? – iherazo 3 hours ago

@iherazo You can formulate the problem as a question and post the guide as an answer to the question. – Modus Tollens 3 hours ago

@Modus Tollens I just wanted to help. If providing some guidelines without posting a question isn"t allowed on the SO community, please tell me and I won't do it again. – iherazo 3 hours ago

@Modus Tollens Ok. I got you. But I won't post the entire guide again. If I do that, then I'll create another post and put a reference to this one. – iherazo 3 hours ago

@iherazo Because this is a question and answer site the posts need to have a specific, on-topic question. This allows users to post their answers (including self-answers), and it allows other users to vote on these answers. Posting q and a as a question breaks the system. But your post is good and you can still easily edit it to conform to q/a (by editing the question and posting an answer). Just take care to formulate a good, on-topic question. – Modus Tollens 3 hours ago

@iherazo No, please don't use this post as a resource to link to it. Please edit it instead! Or post q/a and delete this. – Modus Tollens 3 hours ago

@iherazo If you are still unsure what to do, please ask on meta.stackoverflow.com – Modus Tollens 3 hours ago

@iherazo (You should not link to this as answer for other questions because this will likely be deleted. All your hard work would be lost.) – Modus Tollens 3 hours ago

@Modus Tollens Done. I've edited this post as you've told me to in order to comply with SO guidelines. – iherazo 1 hour ago