ISEngineering

Technische Universität Berlin

Enterprise Computing: Exercise 4 – Platforms

Markus Klems, Stefan Tai

Task 1

a) Your multi-tiered Java EE application needs to integrate with a legacy Enter-prise Resource Planning (ERP) system. Which tier in the Java EE application is most suitable for the ERP system?

b) Shortly describe in your own words the responsibilities of web tier and business tier.

c) Shortly explain: which method of the HttpServlet is most suitable for loading initialization parameters that are specified in the deployment descriptor?

Task 1

```
@Entity
public class Employee {
  @Id
  private long id;
  private String status;
  ...
  @ManyToMany
   private List<Project> projects;
  .....
}
```

```
 @Entity
public class Project {
  @Id
  private long id;
  ...
  @ManyToMany(mappedBy=
    "projects")
  private List<Employee>
    employees;
  ...
}
```

Task 1

d) How many tables are created in the database to store these entities and their relationship?

e) Based on the entities given in d), please write a query using the Java Persistence Query Language that deletes employees from the database which have no projects and who have the status “fired”.

Task 2

Create a GWT+AppEngine project in Eclipse with generated GreetingService sample code.

a) Add a logging filter servlet that logs every interaction with the service. The logging filter must output the statement “A new greeting!” with the log level “INFO” every time the GreetingService has been called by a user.

Google App Engine – Eclipse IDE setup

Technische Universität Berlin

Installing the Google Plugin for Eclipse

To use the plugin you must be running Java version 7 and a recent version of Eclipse. You can install the Google Plugin for Eclipse using the software update feature of Eclipse. Be sure to use the plugin that corresponds to your version of Eclipse. Follow the installation instructions provided at the links below or, if you are familiar with installing Eclipse plugins, you can simply paste the appropriate plugin link directly into Eclipse.

Eclipse version	Installation instructions	Direct plugin link
Eclipse 4.4 (Luna)	Plugin for Eclipse 4.4 (Luna)	https://dl.google.com/eclipse/plugin/4.4
Eclipse 4.3 (Kepler)	Plugin for Eclipse 4.3 (Kepler)	https://dl.google.com/eclipse/plugin/4.3
Eclipse 3.8/4.2 (Juno)	Plugin for Eclipse 3.8/4.2 (Juno)	https://dl.google.com/eclipse/plugin/4.2

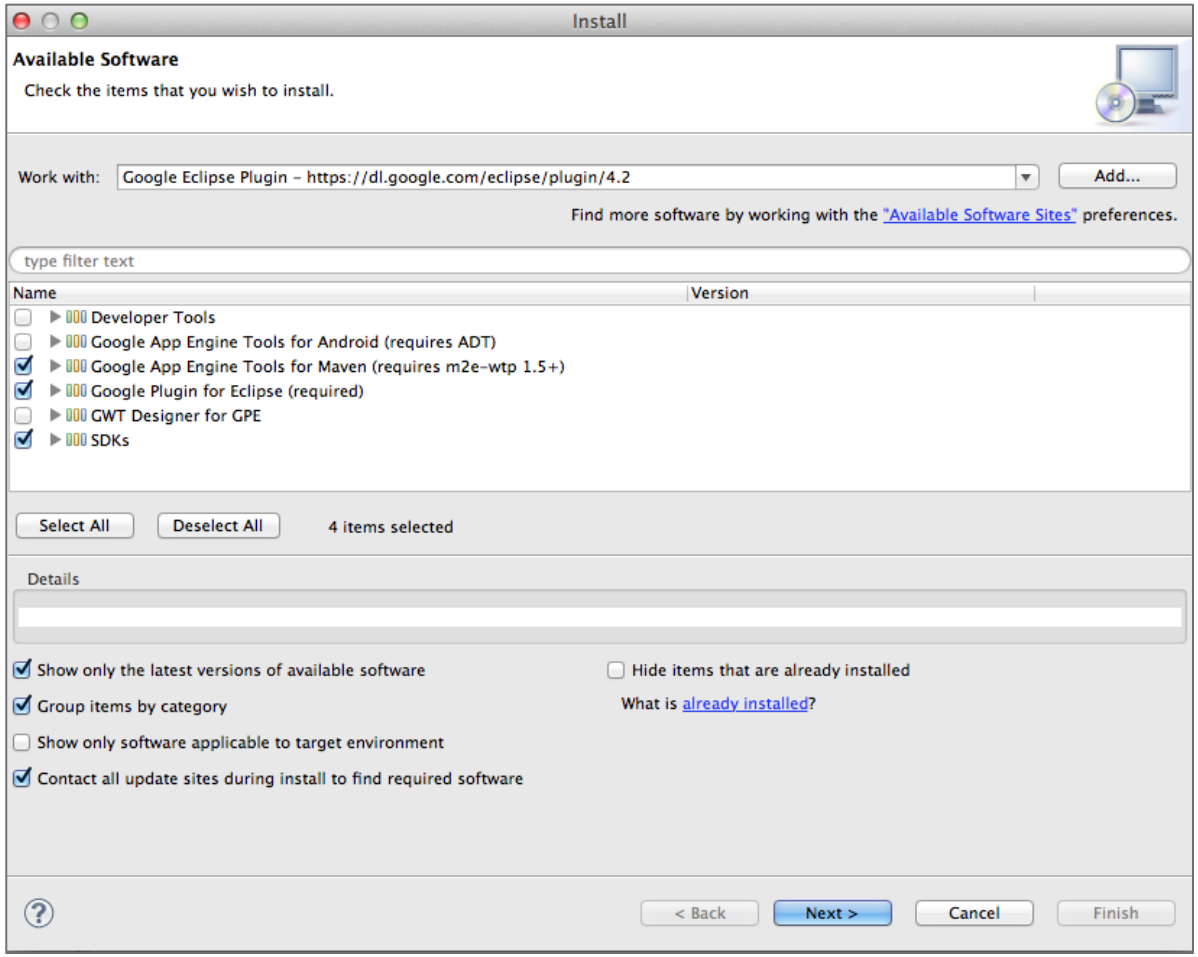
If you are looking for older versions of the plugin, click here.

If you are having trouble installing from the update sites (due to firewall issues, for instance), please see this FAQ entry.

As an alternative to installing from the update site, you can install the Google Plugin for Eclipse by downloading and installing an archive of the update site.

Reference: https://cloud.google.com/appengine/docs/java/tools/eclipse

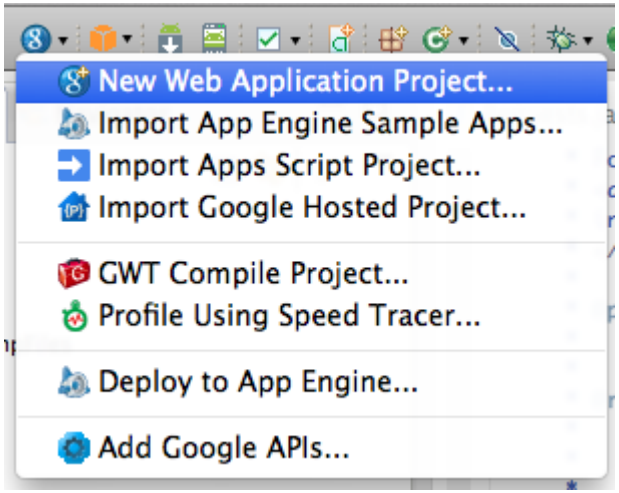
Google App Engine – Eclipse IDE setup



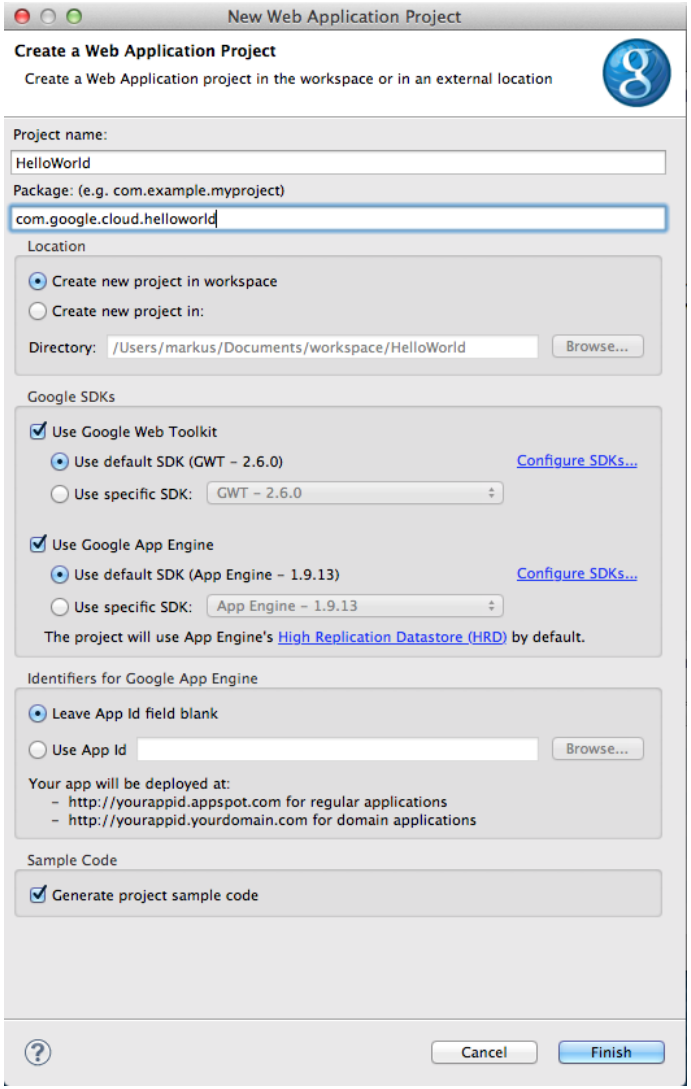
Task 2 - Preparation

1. Create a web application project with generated code
2. Google > GWT compile
3. Run as > Web application
4. Open in your browser: http://localhost:8888/

Google App Engine – create web application project



Google App Engine – create web application project



GreetingService.java

```
package com.google.cloud.guestbook.client;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

/**
 * The client-side stub for the RPC service.
 */
@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {
    String greetServer(String name) throws IllegalArgumentException;
}
```

GreetingServiceAsync.java

```
package com.google.cloud.guestbook.client;

import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 * The async counterpart of <code>GreetingService</code>.
 */
public interface GreetingServiceAsync {
    void greetServer(String input, AsyncCallback<String> callback)
        throws IllegalArgumentException;
}
```

GreetingApp.java

```
package com.google.cloud.guestbook.client;
…
/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */
public class GreetingApp implements EntryPoint {
…
/**
 * Create a remote service proxy to talk to the server-side Greeting service.
 */
private final GreetingServiceAsync greetingService =
    GWT.create(GreetingService.class);
```

GreetingApp.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
    final Button sendButton = new Button("Send");
    final TextBox nameField = new TextBox();
…
    // Create a handler for the sendButton and nameField
    class MyHandler implements ClickHandler, KeyUpHandler {
    /**
     * Fired when the user clicks on the sendButton.
     */
    public void onClick(ClickEvent event) {
    sendNameToServer();
    }
…
```

GreetingApp.java

```
/**
 * Send the name from the nameField to the server and wait for a response.
 */
private void sendNameToServer() {
….
greetingService.greetServer(textToServer,
    new AsyncCallback<String>() {
        public void onFailure(Throwable caught) {
            // Show the RPC error message to the user
            …
        }
        public void onSuccess(String result) {
            …
        }
    });
```

GreetingServiceImpl.java

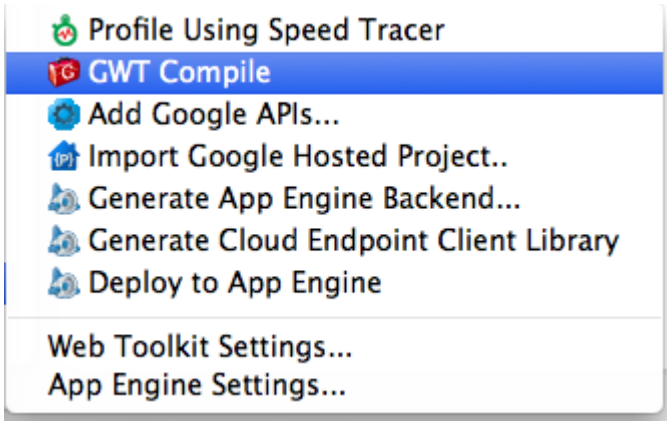
```
package com.google.cloud.guestbook.server;
…
/**
 * The server-side implementation of the RPC service.
 */
@SuppressWarnings("serial")
public class GreetingServiceImpl extends RemoteServiceServlet implements
        GreetingService {

    public String greetServer(String input) throws IllegalArgumentException {
            …
            return "Hello … “;
    }
}
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app … >

  <!-- Servlets -->
  <servlet>
    <servlet-name>greetServlet</servlet-name>
    <servlet-class>com.google.cloud.guestbook.server.GreetingServiceImpl
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>greetServlet</servlet-name>
    <url-pattern>/guestbook/greet</url-pattern>
  </servlet-mapping>

</web-app>
```

GWT compile & run locally

1. Google > GWT compile
2. Run as > Web application
3. Open in your browser: http://localhost:8888/



Task 2

b) Use JPA to persist each incoming user name in an Employee Entity. The Entity has an autogenerated Key and in addition to the name also a hire date which is equal to the time when the server has received the service call. Before saving the new employee, query all existing employees and return them in the response message.

Task 2 b) - Preparation

- Create a class EMF.java like this one

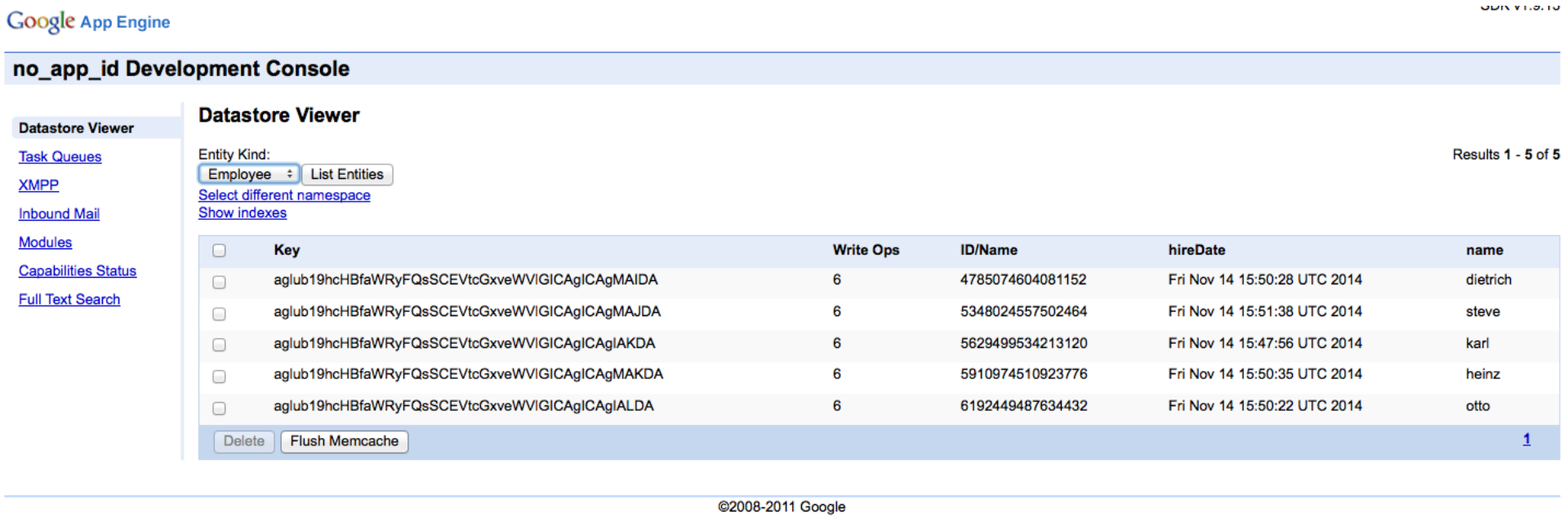
```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public final class EMF {
    private static final EntityManagerFactory emfInstance =
        Persistence.createEntityManagerFactory("transactions-optional");
    private EMF() {}
    public static EntityManagerFactory get() {
        return emfInstance;
    }
}
```

Task 2 b) - Preparation

- JPA Query examples:
  - `em.persist(newEmployee)`
  - `javax.persistence.Query q = em.createQuery(`
    `"select e from Employee e", Employee.class)`
- Don’t forget to `em.close()`

Task 2 b) - Preparation

- Go to http://localhost:8888/_ah/admin/ and view your Entities



Task 2 b) – Expected result (example)