GJA - Java Web Applications Examples

A Java examples project for GJA class at BUT FIT. This is the global readme with all information and all examples. This readme has been copied (and further specified for concrete examples) in each of the examples folders so they can be distributed separately.

Javadoc Documentation

Automatically generated Javadoc documentation can be found in the `doc/` folder in each project subfolder. Open the `index.html` in each documentation to see the Javadoc.

Prerequisites

The project was developed in Apache Netbeans 16 with Eclipse Temurin JDK 17 and Eclipse GlassFish 7 (`Jakarta EE 10 API`). Other prerequisite is that the user is somehow familiar with using Netbeans and deploying Java Web Applications onto the server. If not, user can read the provided documentation.

Precompiled .war files

- For people, that just want to try the project, but don't want to build it themselves, they can download compiled `.war/.jar` files from the project's github https://github.com/bartak-v/gja. The `.war` file for each project is in the subfolder of the project.

Installation and run

- Install `GlassFish 7`
- Run `GlassFish` (either manually or through Netbeans)
- Deploy the compiled `.war` file onto the `GlassFish` server (defaultly through http://localhost:4848)
- You can also build the project yourself through standard practice in `Netbeans` (or manually using `Maven`).

Building and deploying the projects with Netbeans 16 and GlassFish 7

Environment Setup

- First, install Eclipse Temurin JDK 17 and check that the `JAVA_HOME` is set correctly to the proper path, to where you have your JDK installed. `JAVA_HOME` should be set to `...example_jdk_path/` . Environment variable setup is out of scope of this project and we recommend `Google` if you have trouble with this (if it does not setup automatically when installing etc.).
- Install Apache Netbeans 16.
- To correctly deploy the examples, download and install GlassFish 7 on your computer. Save it in some User-owned folder that you have permissions to.
- To set-up `Netbeans` to use the `GlassFish 7` server: Open `Netbeans 16` . Click on `Tools -> Servers` in the top bar.
- Click `Add Server...` . Choose `GlassFish Server` and `Next` .
- Set the `Installation Location` to the folder where you got your `Glassfish 7` downloaded. Select `Local Domain` .
- DO NOT ACCEPT THE TERMS OF SERVICE AND DO NOT DOWNLOAD `GlassFish 6.2.5` , just click `Next` .
- `Netbeans` will pretend, that your installation is version `6.2.5` but it will correctly work and run the version `7.0.0` .
- In the next step set you can leave the `Defaults` and click `Finish` etc.
- In the Server tab you should have set `Java Platform` to `JDK 17` for the server.
- You should now have correctly set-up `Netbeans` to use `GlassFish 7.0.0`

Building and deploying the example projects in Netbeans 16

- Start `Netbeans` and click on the `File` menu in the top left corner.
- Select `Open Project` from the drop-down menu.
- In the file browser window that appears, navigate to the location where your project is stored and select the project folder.
- Click the `Open` button.

- Click on the `Build menu` in the top menu bar.
- Select `Clean and Build Project` from the drop-down menu.
- If the build is successful, you should see a message indicating that the build was successful. If there are any errors, they will be displayed in the output window at the bottom of the Netbeans window.
- You can now click the `Play` button or press `F6` or go to `Run -> Run Project`, if you setup `Netbeans` correctly, web browser should start with the Project running.
- You should also be able to manage the server from the `Services -> Servers -> GlassFish Server` (start, stop, open admin GUI etc.).
- Follow the instructions for each project (as they can be different than this) to successfully deploy and run the example.

Deploying .war files to GlassFish manually (Linux but it should work on Windows too)

- You can also deploy the project manually. After Building the project, the resulting `.war` file should be in the `target/` folder in the root of the project.
- To correctly deploy the examples, download and install `GlassFish 7` on your computer. You can download it from the following link: https://GlassFish.java.net/download.html
- Once `GlassFish` is installed, start the `domain`. You can start the `domain` by running the following command (in the `GlassFish` installation folder go to `GlassFish/bin/`) and run: `./asadmin start-domain` - for this to work you have to have correctly set the `JAVA_HOME` environment variable to where you have your JDK installed.
- Next, open a web browser and navigate to the `GlassFish Administration Console` at the following URL: http://localhost:4848/
- If it wants login credentials, either leave them empty (username `admin` and empty password or you can follow https://docs.oracle.com/cd/E18930_01/html/821-2416/giubb.html or it should be `admin admin` ...)
- Log in to the `Administration Console` using the default username.
- In the left navigation menu, click on the `Applications` link.
- Click on the `Deploy...` button.
- In the `Deploy Applications` screen, click on the `Choose File` button and select the example `.war` file that you want to deploy.
- In the next screen, you can specify deployment options such as the context root and the target server (you can leave the defaults). Make any necessary changes and click on the `Finish` button to deploy the `.war` file.
- Click on the `Applications` on the left again and `Launch` the specified Application. It should show you the links, but we recommend to change the `Context Root` in the application to something like `/servlet_jsp_example` (instead of `/servlet_jsp_example-14374286702991946667.0` etc.) or just leave it and use it as the root of the examples.
- The `.war` file will now be deployed to GlassFish and should be accessible at the specified context root. You can check the URLs we specify in the examples section.

Examples

Here follows instructions and information about each example.

Servlet Examples

Number Guesser Game with Cookies (/servlet_example/ExampleServlet)

- For deployment see Section on .war deployment.
- The Servlet example is a Random Number Guessing game. The class implements `HTTP` requests and utilizes `Cookies` for primitive session keeping.
- You can test the Example Servlet via a web browser or "API-testing" program like Postman or `cURL` (manipulating raw HTTP requests sent to e.g.http://localhost:8080/servlet_jsp_example/ExampleServlet) - You need to use something like `Postman` to test DELETE and PUT methods - as they can't be called from `HTML` page.
- You can test `GET`, `POST`, `PUT`, `DELETE` `HTTP` requests on the Servlet.
- By calling `PUT` manually you restart the game. By calling `DELETE` you delete the Cookies and Restart the Game (deletes

your username, high score etc.) - this will be only available through `cURL` and/or `Postman` as they have different sessions than your browser.

- This example, showing basic capabilities of Servlets is also a deterrent example of why it is better to use something like `JSP` for `HTML` rendering.

- `Javadoc` documentation has been generated and put into `doc/`.

Multi-File Upload Servlet (/servlet_upload_example/)

- For deployment see Section on .war deployment.
- Second Servlet example is an updated Multiple File Upload servlet using the capabilities of `Jakarta EE 10` (`Servlet 3.0+` in-house fileupload).
- It's capabilities are to upload multiple files and validate them in some ways (check that they are images, their size etc.).
- You can also view the uploaded images through simple HTML page.

- After deploying, you should find it at http://localhost:8080/servlet_upload_example/ or through `Applications -> Launch` in the `GlassFish Admin GUI`.

- `Javadoc` documentation has been generated and put into `doc/`.

JSP Examples 2.0

- For deployment see Section on .war deployment.
- After deployment, you should find the app running on http://localhost:8080/JSPExamples/
- This is a set of `JSP` examples that showcase various `JSP` scenarios.
- User can test simple `calendar`, shopping `cart`, `mail` and `number guesser game`.
- Instead of creating new `JSP` examples, we have updated and refactored the older GJA `JSP` examples - because they are all-embracing already.
- They have been refactored to run out of box with `JDK 17` and `GlassFish 7` and support deployment out of `Netbeans`.
- The code has been refactored and reformated to use `HTML 5` and `Jakarta EE 10`.

- New unified `Bootstrap 5 UI` (similar to the new servlet examples) has been created and the functionality of the project has been tested with `GlassFish 7` and `JDK 17`.

- `Javadoc` documentation has been generated and put into `doc/`.

JMS Examples

- After 7+ hours of debugging and Googling I was not able to get a basic example HelloWorld JMS example (or any of the older examples) running with `Glassfish 7` and `Jakarta EE 10` - so this part was skipped.

Testing, Maven, JAX Examples

calculator-junit_arquillian

- Demo shows usage of unit testing in Java using JUnit and also integration testing using Arquillian
- Unit tests are implemented on Calculator class, testing the results of basic mathematical operations
- Integration tests are implemented on Student class. A student injects Calculator. This dependency injection is then tested using Arquillian.
- Compiled `.jar` is added to the root of the projects for easy deployment.
- `Javadoc` documentation has been generated and put into `doc/`.

VUTNews-selenium

- A demo of Selenium using Chrome browser to read news from VUT index page.
- Usage of properties, dependencyManagement and main class selection in pom.xml.
- A web browser driver in the root of the project is needed to run the application, a chrome driver can be downloaded at https://chromedriver.chromium.org/downloads
- Compiled `.jar` is added to the root of the projects for easy deployment.
- `Javadoc` documentation has been generated and put into `doc/`.

JAX -WS

- Client and publisher, first run publisher. Client then connects to the publisher at port 6666 (if the port is not free on your machine, change it) and retrieves object HelloWorld. The call *hello.getHelloWorldAsString("fit")* seems to be executed on client side but it is actually executed in the publisher app and the result is retrieved through the port.
- There are also Client and Server handlers. Client handler adds MacAddress to request, Server handler reads it and validates. The handler configuration is in *handler-client.xml* and *handler-server.xml*.
- Compiled `.jar` is added to the root of the projects for easy deployment.
- `Javadoc` documentation has been generated and put into `doc/`.

Jersey

- A Jersey demo showing basic usage of the framework. The application runs at http://localhost:8080/jersey.
- There are 2 controller Classes, first *Basic* shows basic HTTP GET processing. Second *Arguments* shows retrieving user input multiple ways.
- `Javadoc` documentation has been generated and put into `doc/`.

EJB and JSF Examples

EJB StatefulBean and StatelessBean

- Two projects showing the difference between stateful and stateless beans.
- The stateful bean project shows a bank account bean, which balance is preserved thanks to the bean being stateful across client requests.
- The stateless bean project is very simillar in structure. This time a @Stateless annotation is used on the bean because there is no point in holding a state of a calculator (at least not in this example, where are no intermediate results)
- applications runs at http://localhost:8080/StatefullBean and http://localhost:8080/StatelessBean
- `Javadoc` documentation has been generated and put into `doc/`.

JSFPageNavigation

- Showcase of types of navigations between pages that can be used in JSF. There are navigation rules in *faces-config.xml*, usage of commandLinks and commandButtons. Also a difference between redirection and forwarding. All the necessary information can be found in comments in source code.
- application runs at http://localhost:8080/JSFPageNavigation
- `Javadoc` documentation has been generated and put into `doc/`.

JSFEventListeners

- Showing some basic events and listeners in JSF. There is:
  - ActionListener which is called on button click
  - SystemListener which listens on application start and stop
  - ValueChangeListener which listens on second *selectOneMenu* element in *homepage.xhtml*. There is also shown a direct method call in first *selectOneMenu* element.
- The listeners (except SystemListener) change values in UserData bean.
- application runs at http://localhost:8080/JSFEventListeners
- `Javadoc` documentation has been generated and put into `doc/`.

JSFCustomComponent

- Creation of custom register component in JSF. Take a look at *register.xml* and see its usage in *default.xhtml*.
- The new component is configured via defined attributes like , which must be provided when you want to use the new component. Then its implementation is rendered.
- application runs at http://localhost:8080/JSFCustomComponent

- `Javadoc` documentation has been generated and put into `doc/`.

JSFAjax

- in *home.xml* is shown a usage of ajax to change *UserData* bean attribute *name*. After the Ajax call is processed, the *outputText* element is rerendered (defined in *render="outputMessage"* attribute in *f:ajax* element).
- application runs at http://localhost:8080/JSFAjax
- `Javadoc` documentation has been generated and put into `doc/`.

Primefaces Examples

- This is a set of examples that showcase some functions of Primefaces 12.0.
- For deployment see Section on .war deployment.
- Older examples were adapted and refactored to work with Jakarta EE 10, Glassfish 7 and JDK 17. Other maven dependencies were also updated to latest versions.
- Incompatible and non-working examples were removed.
- `Javadoc` documentation has been generated and put into `doc/` in each example.

JPA and Hibernate Examples

JPA-SE

- Shows basic usage of entity manager.
- The application uses ObjectDB for its easy implementation (no DB configuration needed).
- Objects of Point class are stored in local object database, then being queried upon.
- Example shows selecting all points from database and two aggregation functions.
- Compiled `.jar` is added to the root of the projects for easy deployment.
- `Javadoc` documentation has been generated and put into `doc/`.

JPA-EE

- The example once again uses ObjectDB.
- Similar to usage in SE, but this example uses user input and HTTP protocol to create objects to save
- ServletContextListener initializes object database and closes it on application stop
- on HTTP request *GuestServlet* checks if parameter *name* is not null. If not, creates a new Guest and saves into the object db
- The parameter is set via form in *guest.jsp*ä
- application runs at http://localhost:8080/JPA-EE
- `Javadoc` documentation has been generated and put into `doc/`.

HibernateExample and HibernateAnnotation

- Usage of Hibernate framework. A MySQL connection needs to be configured in *hibernate.cfg.xml*. The example also provides SQL file for table creation.
- There is a *Employee* class and *Employee.hbm.xml* mapping defined upon the class. The mapping is registered in *hibernate.cfg.xml*.
- *ManageEmployee* class implements operations:
  - addBatchEmployees() - usage of persist() and flush() to execute SQL commands in batches
  - addEmployee(fname, lname, salary) - add record to table
  - deleteEmployee(ID) - delete record from table by ID
  - listEmployeesEntity() - select all records
  - listEmployeesScalar() - projection
  - updateEmployee(ID, salary) - update record by ID
- *MyInterceptor* has methods, which are called on Hibernate events like *onSave*. The interceptor is created when a DB session is created:

```
// Session session = factory.openSession();
Session session = factory.withOptions().interceptor(new MyInterceptor()).openSession();
```

- HibernateAnnotation is a project (with no functionality) with annotation Employee mapping instead of XML.
- Compiled `.jar` is added to the root of the projects for easy deployment.
- `Javadoc` documentation has been generated and put into `doc/`.