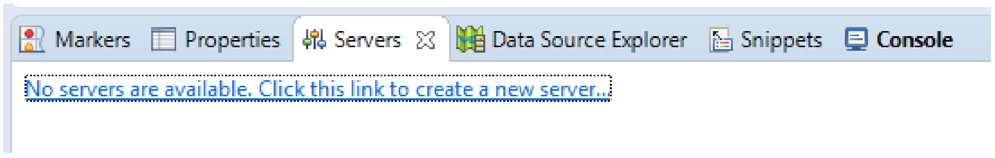
All data, which normally would be hold in a data base, for lab purpose will stay in the memory storage.

Developer environment

- Eclipse
- Server Wildfly 16 or newer

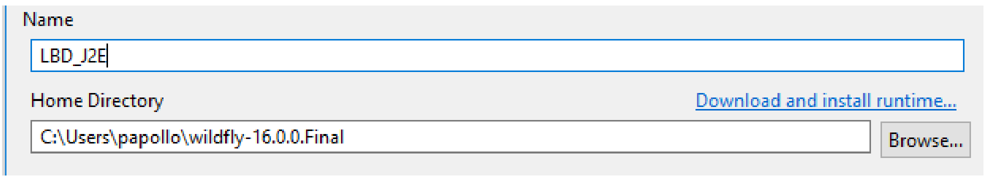
Setup environment

1. Install server:



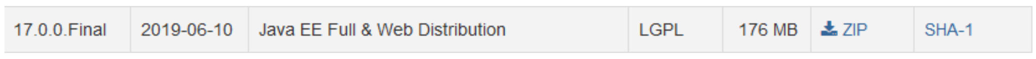
No servers are available. Click this link to create a new server...

Select Wildfly 16.0 and procced. Leave default settings and go to next page.


Name: LBD_J2E

Home Directory: C:\Users\papollo\wildfly-16.0.0.Final

You can download server directly from eclipse or if something goes wrong you can download it from wildfly site (https://wildfly.org/downloads/ and download Wildfly server 16.0 or higher).

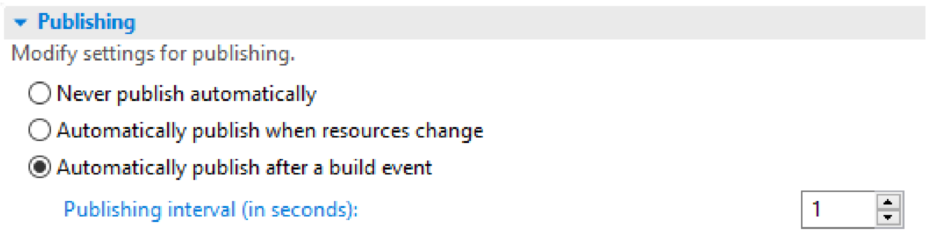


17.0.0.Final	2019-06-10	Java EE Full & Web Distribution	LGPL	176 MB	ZIP	SHA-1

2. Configure hot deployment on wildfly server:

Go to tab servers -> double click on installed server.

In newly opened window “Overview” find dropdown named “Publishing” and check third option:

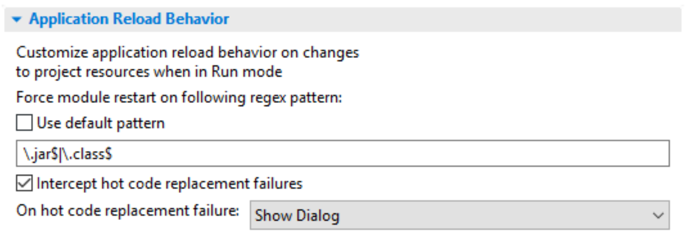

Publishing

Modify settings for publishing.

- Never publish automatically
- Automatically publish when resources change
- [✓ Automatically publish after a build event]

Publishing interval (in seconds): 1

Next go to dropdown “Application Reload Behavior”, check use default pattern checkbox and edit the regex pattern to: “`\.jar$|\.class$`”.



Application Reload Behavior

Customize application reload behavior on changes to project resources when in Run mode

Force module restart on following regex pattern:

☐ Use default pattern

\.jar$|\.class$

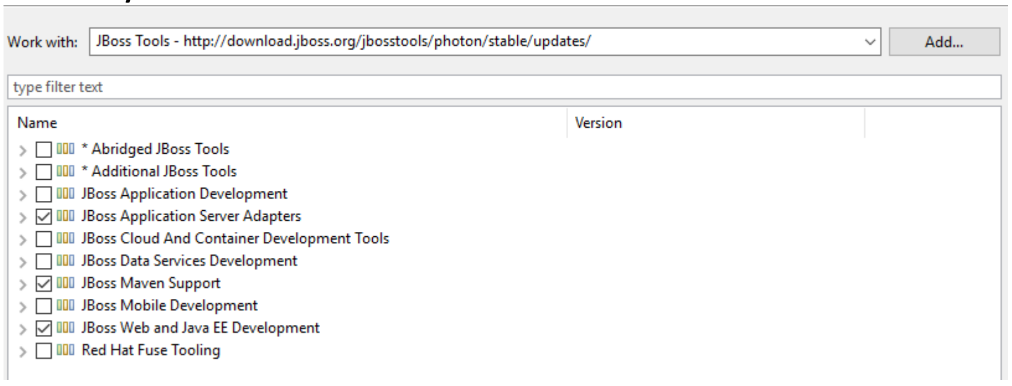
☑ Intercept hot code replacement failures

On hot code replacement failure: Show Dialog

3. Install JBossTools

In eclipse go to “Help > Install New Software…”. In field “Work with” paste: http://download.jboss.org/jbosstools/photon/stable/updates/

Select all necessary:

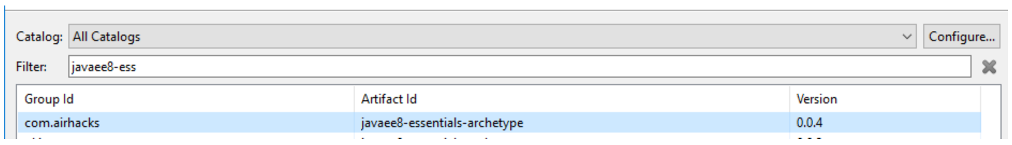


Work with: JBoss Tools - http://download.jboss.org/jbosstools/photon/stable/updates/

Name	Version
☐ * Abridged JBoss Tools	
☐ * Additional JBoss Tools	
☐ JBoss Application Development	
☑ JBoss Application Server Adapters	
☐ JBoss Cloud And Container Development Tools	
☐ JBoss Data Services Development	
☑ JBoss Maven Support	
☐ JBoss Mobile Development	
☑ JBoss Web and Java EE Development	
☐ Red Hat Fuse Tooling	

4. Add new Maven project:

Create new project by clicking: File > new > Maven Project
In first window leave everything as default and proceed.
In next window filter by **javaee8-essentials-archetype**
Choose one with **com.airhacks** Group Id



Catalog: All Catalogs

Filter: javaee8-ess

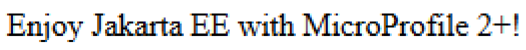
Group Id	Artifact Id	Version
com.airhacks	javaee8-essentials-archetype	0.0.4

Click next and as the Group Id set: “pl.fis.<name>.<last_name>”
As Artifact Id set: **“lbd-jax-rs**”
Example : pl.fis.jan.kowalski.lbd

5. Run project

To start project click right mouse button on the your project in the Project Explorer Tab on the left. Select “Run As” -> “Run on server”. Then choose an existing server. Select one you’ve already created go next and add you project into server deployments.

If everything goes right you should be able to see under address: http://127.0.0.1:8080/lbd-jax-rs/resources/ping/
Result string:



```
Enjoy Jakarta EE with MicroProfile 2+!
```

Warm up

All uris to the endpoints should have a form:

`"http://<domain>/lbd-jax-rs/api/<endpoint>”`

1. Create endpoint for http-GET method. Return a spaceship list using uri `"/v1/spaceships"`.
2. Create endpoint for http-POST method adding „X-Wing“ spaceship to the ship list. Use uri `"/v1/spaceships"`.
   Check using GET method the occurrence of added ship in the list.

Endpoint – basics

Implement space fleet management RESTful application that provides following functionality:

3. Provides spaceship list as xml representation. Use uri `"/v2/spaceships"`.
   Response should looks like:

```
<spaceships>
    <ship>Battlestar Galactica</ship>
    <ship>Elysium</ship>
    …
</spaceships>
```

4. Provides spaceship list as json representation. Use uri `"/v2/spaceships"`.
   Response should looks like:

```
{
    "spaceships": [
        "Battlestar Galactica",
        "Elysium",
        …
    ]
}
```

5. Adding spaceship as xml to the ship list. Use uri `"/v2/spaceships"`.

```
<ship>X-Wing</ship>
```

   Check using GET method the occurrence of added ship in the list.

6. Adding spaceship as json to the ship list. Use uri `"/v2/spaceships"`.

```
{ "spaceship": "B-Wing" }
```

   Check operation result using GET method.

7. Removing specified spaceship from the ship list. Use uri `"/v2/spaceships/<spaceship name>"`.
   Check removing operation result using GET method.

Binding

Extending the application by using JSON binding.

8. Provide space fleet name with ship's list as a json representation. Use uri `"/v3/space-fleet"`.

   Response should be formatted as follows:

   ```
   {
       "name": "FIS Space Fleet",
       "ships": [
       {
            "name": "Battlestar Galactica",
            "speed": 3.5
        },
        …
        ]
    }
   ```

9. Add new spaceship to the space fleet passing its json representation to the server. To be added spaceship has a name and a speed. Use uri `"/v3/space-fleet"`.

   Check using GET method the occurrence of added ship in the list.

10. Get detail spaceship information using `"/v3/ships/<spaceship name>"`. As an endpoint response an object should be used.

Validation and exception handling

1. Throw own `ResourceNotFoundException` on calling not existing spaceship detail information (eg. “B-Wing”). Use `"/v4/space-fleet/<spaceship name>"` uri. Implement generic solution that returns json instead of default error page.

```
{
    "status": "NOT_FOUND",
    "message": "resource '<spaceship name>' not found"
}
```

2. Extend spaceship representation by adding *year-of-manufacturing* field as java’s LocalTime.

```
{
    "year-of-manufacturing": "2003-10-07",
    "name": "Elysium",
    "speed": 5
}
```

3. Get space fleet as a json representation. Use uri `"/v4/space-fleet"`.
   Check operation result using GET method.

```
{
    "name": "FIS Space Fleet",
    "ships": [
            {
            "name": "Battlestar Galactica",
            "speed": 3.5,
            "year-of-manufacturing": "2003-10-07"
            },
    …
    ]
}
```

4. Add new spaceship to the space fleet passing its json representation to the server. Use uri `"/v4/space-fleet"`.

   Extend spaceship by adding fields:
   `year-of-manufacturing` – optional but need to be in the past if present
   `name` – required and need to be not blank
   `speed` – required and need to be positive

   Check spaceship information by calling `"/v4/space-fleet/<spaceship name>"`.

5. On validation failure return a json error message containing validation error message as follow:

```
{
    "status": "BAD_REQUEST",
    "field-errors": [
    { "<field name>" : : "<here an error message>" },
    …
    ]
}
```

API-Documentation

1. Document and generate documentation of the v4-endpoints using openAPI/Swagger.

Additional tasks

1. Update validation error message by using user readable message. Use user language (grasp in `ContainerRequestFilter)` to choose right message translation. Support Polish and English (default) language at least.
2. Every response of GET request should be valid/cached for five minutes.
3. Create new GET endpoint by using `"/v4/space-fleet/ships”` uri. It returns a list of spaceships with its detail information. Returned spaceships should be sorted and may be requested by a user. Allowed sorting modes are:
    a. `asc` - ascending (default),
    b. `desc` - descending,
    c. `name` (default)
    d. `speed`

   Combining of sorting methods is allowed (eg. `desc` and `speed)`
4. Extend previous response by adding paging information. Every spaceship should have relative link to its detail information:

   ```
   {    "rel": "self", “href” : “relative uri” }
   ```

   Five spaceships should be returned in the response. The response should have links to first, previous, next and last page if applicable.
5. Write Junit tests for the paging calculation functionality.

Initial spaceship list

```
"Battlestar Galactica", "Elysium", "Reapers", "USS Enterprise", "Spaceball One",
"Millennium Falcon", "Death Star", "Starkiller Base", "Tie Fighter", "USCSS
Prometheus"
```