Zusammenfassung 1.Test

pom.xml

1. Add Packaging

Add under Version section:

```
<packaging>war</packaging>
```

2. Add properties

```
 <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <properties>
```

3. Add dependencies

You have to add the javaee-web-api dependency. Use ALT + EINF
It should look like this:

```
<dependencies>
    <dependency>
        <groupId>javax<groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>8.0</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

4. Add Build Name

You have to add a build name. This name has to be part of path in wildfly config

```
<build>
    <finalName>NameOfProject</finalName>
</build>
```

Rest

RestConfig

The class RestConfig has to be annotated with

```
@ApplicationPath("api/bsp")
```

The class also have to extend Applications

RestConfig

Here we config the methods for inserting and reading from database
The class have to be annotated with:

```
@Transactional
@ApplicationScoped
```

Then you have to add an Entity Manager:

```
@PersistenceContext
EntityManager entityManager;
```

Than you can insert update read delet,..... for example:

```
 public ToDo createToDo(ToDo todo){
        //Persist into db
        entityManager.persist(todo);
        return todo;
    }

    public ToDo updateToDo(ToDo toDo){
        entityManager.merge(toDo);
        return toDo;
    }

    public ToDo findToDoById(Long id){
        return entityManager.find(ToDo.class,id);
    }

    public List<ToDo> getTodos(){
        return entityManager.createQuery("Select t from ToDo t", ToDo.class).getResultList();
    }
```

RestClass

Is just to combine Services with Path and Method
Needs to be annotated with

```
@Path("todo")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
```

Than the Service has to be innjected with

```
@Inject
ToDoService toDoService;
```

Than you can create Methods annotated with a Path and a Type. For Example:

```
@Path("new")
    @POST
```

```
    public Response createToDo(ToDo toDo){
        toDoService.createToDo(toDo);
        return Response.ok(toDo).build();
    }

    @Path("update")
    @PUT
    public Response updateToDo(ToDo toDo){
        toDoService.updateToDo(toDo);
        return Response.ok(toDo).build();
    }

    @Path("{id}")
    @GET
    public ToDo getTodo(@PathParam("id") Long id){
        return toDoService.findToDoById(id);
    }@Path("new")
    @POST
    public Response createToDo(ToDo toDo){
        toDoService.createToDo(toDo);
        return Response.ok(toDo).build();
    }

    @Path("update")
    @PUT
    public Response updateToDo(ToDo toDo){
        toDoService.updateToDo(toDo);
        return Response.ok(toDo).build();
    }

    @Path("{id}")
    @GET
    public ToDo getTodo(@PathParam("id") Long id){
        return toDoService.findToDoById(id);
    }

    @Path("list")
    @GET
    public List<ToDo> getToDos(){
        return toDoService.getTodos();
    }

    @Path("list")
    @GET
    public List<ToDo> getToDos(){
        return toDoService.getTodos();
    }
```

JPA

Entities

are Classes with the annotation

```
@Entity
```

The primary Key’s are annotated with:

```
@Id
```

Generated values are annotated with:

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

Methods that should be excecuted before start of Lifecycle have to be annotaed with:

```
@PrePersist
```

Hints

DB

You start the db with

```
/opt/db-derby-10.14.2.0-bin/bin/startNetworkServer -noSecurityManager
```

Tests

Test’s are annotatated with

```
@Test
```

and the classes have to be placed in the test/java folder.

Github

Git Comments

```
    1) Specify the type of commit:

    add: Add something new
    remove: deleted something
    update: Update something
    refactor: Renaming something
    feat: The new feature you are adding to a particular application
    fix: A bug fix
    style: Feature and updates related to styling
    refactor: Refactoring a specific section of the codebase
    test: Everything related to testing
    docs: Everything related to documentation
    chore: Regular code maintenance.[ You can also use emojis to represent commit types]

    2) Separate the subject from the body with a blank line
    3) Your commit message should not contain any whitespace errors
    4) Remove unnecessary punctuation marks
    5) Do not end the subject line with a period
    6) Capitalize the subject line and each paragraph
    7) Use the imperative mood in the subject line
    8) Use the body to explain what changes you have made and why you made them.
```

```
9) Do not assume the reviewer understands what the original problem was, ensure you add it.
10) Do not think your code is self-explanatory
11) Follow the commit convention defined by your team
```

GitIgnore

```
#Maven target directory
target/

# Compiled class file
*.class

# Log file
*.log

# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
```