

```

1  --Persistence.xml
2  <?xml version="1.0" encoding="UTF-8" ?>
3  <persistence (...) >
4  <persistence-unit name="todos"
5  transaction-type="RESOURCE_LOCAL">
6  <class>Todo</class>
7  <properties> (...) </properties>
8  </persistence-unit>
9  <persistence-unit name="people"
10 transaction-type="RESOURCE_LOCAL">
11 <class>Person</class>
12 <class>Family</class>
13 </persistence-unit>
14 </persistence>
15 --Family.java
16 @Entity
17 public class Family {
18     @Id @GeneratedValue(strategy =
19     GenerationType.TABLE)
20     private int id;    private String
21     description;
22     @OneToMany
23     private final List<Person> members;
24     // getters & setters omitted
25 }
26 --Person.java
27 @Entity
28 public class Person {
29     @Id
30     @GeneratedValue(strategy =
31     GenerationType.TABLE)
32     private String id;    private String
33     firstName;
34     private String lastName;
35     // getters & setters omitted
36 }
37 --Job.java
38 @Entity
39 public class Job {
40     @Id
41     @GeneratedValue(
42     strategy = GenerationType.TABLE)
43     private String id;    private String
44     function;
45     // getters & setters omitted
46 }
47 --JpaTest.java
48 public class JpaTest {
49     static final String UNIT_NAME =
50     "people";
51     private EntityManagerFactory factory;
52     EntityManager em = factory
53     .createEntityManager();
54     @Before
55     public void setUp() throws Exception
56     {
57         factory = Persistence
58         .createEntityManagerFactory(UNIT_NAME);
59         this.em =
60         factory.createEntityManager();
61         em.getTransaction().begin();
62         Query q=em.createQuery("select m
63         from Person m");
64         boolean createNewEntries =
65         (q.getResultList().size() == 0);
66         if (createNewEntries) {
67             assertTrue(
68             q.getResultList().size() == 0
69             );
70             Family family = new Family();
71             family.setDescription("F Knopfs");
72             em.persist(family);
73             for (int i = 0; i < 40; i++) {
74                 Person person = new Person();
75                 person.setFirstName("Jim_" + i);
76                 person.setLastName("Knopf_" + i);
77                 em.persist(person);
78                 family.getMembers().add(person);
79                 em.persist(person);
80                 em.persist(family);
81             }
82             em.getTransaction().commit();
83             em.close();
84         }
85         @Test
86         public void checkAvailablePeople() {
87             Query q=em
88             .createQuery("select m from Person
89             m");
90             assertTrue(
91             q.getResultList().size() == 40
92             );
93             em.close();
94         }
95         @Test
96         public void checkFamily() {
97             Query q=
98             em.createQuery("select f from Family
99             f");
100             assertTrue(
101             q.getResultList().size()==1);
102             assertTrue(
103             ((Family) q.getSingleResult())
104             .getMembers().size()==40
105             );
106             em.close();
107         }
108         @Test(expected =
109         javax.persistence.NoResultException.class)
110         public void deletePerson() {
111             em.getTransaction().begin();
112             Query q = em.createQuery(
113             "SELECT p FROM Person p
114             WHERE p.firstName = :firstName
115             AND p.lastName = :lastName");
116             q.setParameter("firstName",
117             "Jim_1");
118             q.setParameter("lastName",
119             "Knopf_!");
120             Person user =
121             (Person) q.getSingleResult();
122             em.remove(user);
123             em.getTransaction().commit();
124             Person person =
125             (Person) q.getSingleResult();
126             em.close();
127         }
128     }
129 }

```

```

1  --NewsEntityFacade.java
2  @Stateless
3  public class NewsEntityFacade {
4      @PersistenceContext(unitName = "NewsApp")
5      private EntityManager em;
6      public void create(NewsEntity newsEntity)
7      {
8          em.persist(newsEntity);    }
9      public void edit(NewsEntity newsEntity) {
10         em.merge(newsEntity);    }
11     public void remove(NewsEntity newsEntity)
12     {
13         em.remove(em.merge(newsEntity));    }
14     public NewsEntity find(Object id) {
15         return em.find(NewsEntity.class, id);
16     }
17     public List<NewsEntity> findAll() {...}
18     public int count() { (...) }
19 }
20 --NewsEntity.java
21 @Entity
22 public class NewsEntity implements
23 Serializable {
24     private static final long
25     serialVersionUID = 1L;
26     @Id @GeneratedValue(strategy =
27     GenerationType.AUTO)
28     private Long id;
29     private String title;
30     private String body;
31     // getters & setters omitted
32 }
33
34 --NewMessage.java
35 @MessageDriven(mappedName = "jms/NewMessage",
36 activationConfig =
37 { @ActivationConfigProperty(propertyName =
38 "acknowledgeMode", propertyValue = "Auto-
39 acknowledge"),
40 @ActivationConfigProperty(propertyName =
41 "destinationType", propertyValue =
42 "javax.jms.Queue") })
43 public class NewMessage implements
44 MessageListener {
45     @Resource
46     private MessageDrivenContext mdc;
47     @PersistenceContext(unitName = "NewsApp-
48 ejbPU")
49     private EntityManager em;
50     public NewMessage() { (...) }
51     public void onMessage(Message message) {
52         ObjectMessage msg = null;
53         if (message instanceof ObjectMessage) {
54             msg = (ObjectMessage) message;
55             NewsEntity e =
56                 (NewsEntity) msg.getObject();
57             save(e);
58         }
59     }
60     public void save(Object object) {
61         em.persist(object);    }

```

```

62 }
63
64 --Message1.java
65 @WebServlet(name="Message1",
66 urlPatterns={"/Message1"})
67 public class Message1 extends HttpServlet {
68     @EJB private SessionManagerBean
69     sessionManagerBean;
70     @EJB private NewsEntityFacade
71     newsEntityFacade;
72     protected void
73     processRequest(HttpServletRequest request,
74     HttpServletResponse response) throws (...) {
75         request.getSession(true);
76         PrintWriter out = response.getWriter();
77         (...)
78         out.println("<body>");
79         for (NewsEntity elem :
80     newsEntityFacade.findAll() ){
81             out.println("
82 <b>"+elem.getTitle()+" </b><br />");
83             out.println(elem.getBody()+"<br />
84 ");
85         }
86         (...)
87     }
88 }
89 --Message2.java
90 @WebServlet(name = "Message2", urlPatterns =
91 {"/Message2"})
92 public class Message2 extends HttpServlet {
93     @Resource(mappedName =
94     "jms/NewMessageFactory")
95     private ConnectionFactory cFactory;
96     @Resource(mappedName = "jms/NewMessage")
97     private Queue que;
98
99     protected void
100     processRequest(HttpServletRequest request,
101     HttpServletResponse response) throws (...) {
102         String title =
103             request.getParameter("title");
104         String body =
105             request.getParameter("body");
106         if ((title != null) && (body != null)){
107             Connection conn =
108                 cFactory.createConnection();
109             Session sess =
110                 conn.createSession(...);
111             MessageProducer mProducer=
112                 sess.createProducer(que);
113             ObjectMessage msg =
114                 sess.createObjectMessage();
115             NewsEntity e = new NewsEntity();
116             e.setTitle(title); e.setBody(body);
117             msg.setObject(e);
118             mProducer.send(msg);
119             mProducer.close();
120             conn.close();
121         }
122     }

```

123 <u>--persistence.xml</u> 124 <?xml version="1.0" encoding="UTF-8"?> 125 <persistence (...) > 126 <persistence-unit name="NewsApp-ejbPU" 127 transaction-type="JTA"> 128 <provider>org.eclipse.persistence.jpa.Persistenc 129 eProvider</provider> 130 <jta-data-source>jdbc/sample</jta-data-source> 131 <class>ejb.NewsEntity</class>	132 <exclude-unlisted-classes>true</exclude- 133 unlisted-classes> 134 <properties> 135 <property name="eclipselink.ddl-generation" 136 value="create-tables"/> 137 </properties> 138 </persistence-unit> 139 </persistence> 140
---	---

Code 3 - REST with Java

```

1  --Hello.java
2  @Path("/hello")
3  public class Hello {
4      @GET @Produces(MediaType.TEXT_PLAIN)
5      public String sayPlainTextHello() {
6          return "Hello Jersey";
7      }
8      @GET @Produces(MediaType.TEXT_XML)
9      public String sayXMLHello() {
10         return "<?xml version=\"1.0\"?>"
11             + "<hello> Hello Jersey" + "</hello>";
12     }
13     @GET @Produces(MediaType.TEXT_HTML)
14     public String sayHtmlHello() {
15         return "<html> " + "<title>"
16             + "Hello Jersey" + "</title>"
17             + "<body><h1>" + "Hello Jersey"
18             + "</body></h1>" + "</html> ";
19     }
20 }
21 -- Test.java
22 public class Test {
23     public static void main(String[] args) {
24         ClientConfig config = new
25         ClientConfig();
26         Client client =
27         ClientBuilder.newClient(config);
28         WebTarget target = client
29             .target(getBaseURI());
30         String response = target.path("rest")
31             .path("hello").request()
32             .accept(MediaType.TEXT_PLAIN)
33             .get(Response.class).toString();
34         String plainAnswer =
35         target.path("rest")
36             .path("hello")
37             .request().accept(MediaType.TEXT_PLAIN)
38             .get(String.class);
39         String xmlAnswer = target.path("rest")
40             .path("hello").request()
41             .accept(MediaType.TEXT_XML)
42             .get(String.class);
43         String htmlAnswer= target.path("rest")
44             .path("hello").request()
45             .accept(MediaType.TEXT_HTML)
46             .get(String.class);
47         (...);
48     }
49     private static URI getBaseURI() {
50         return
51         UriBuilder.fromUri("http://localhost:8080/
52         com.vogella.jersey.first").build();
53     }
54 }
55 -- web.xml
56 <?xml version="1.0" encoding="UTF-8"?>
57 <web-app (...)>
58     <display-
59     name>com.vogella.jersey.first</display-
60     name>
61     <servlet>
62         <servlet-name>Jersey REST
63         Service</servlet-name>
64         <servlet-
65         class>org.glassfish.jersey.servlet.Servlet
66         Container</servlet-class>
67         <init-param> <param-name>
68         jersey.config.server.provider.packages</pa
69         ram-name>
70         <param-value>
71         com.vogella.jersey.first</param-value>
72         </init-param>
73         <load-on-startup>1</load-on-startup>
74     </servlet>
75     <servlet-mapping>
76         <servlet-name>Jersey REST
77         Service</servlet-name>
78         <url-pattern>/rest/*</url-pattern>
79     </servlet-mapping>
80 </web-app>
81
82

```

Code 4 - A TRUE RESTFUL Example

```

1  --Todo.java
2  @XmlRootElement
3  public class Todo {
4      private String summary;
5      private String description;
6      // getters & setters omitted
7  }
8  --TodoResource.java
9  @Path("/todo")
10 public class TodoResource {
11
12     @GET
13     Produces({MediaType.APPLICATION_XML})
14     public Todo getXML() {
15         Todo todo = new Todo();
16         todo.setSummary("Todo Summary");
17         todo.setDescription("Todo
18 Description");
19         return todo;
20     }
21
22     @GET
23     Produces({MediaType.APPLICATION_JSON})
24     public Todo getJSON() {
25         Todo todo = new Todo();
26         todo.setSummary("Todo Summary");
27         todo.setDescription("Todo
28 Description");
29         return todo;
30     }
31     @GET    @Produces({ MediaType.TEXT_XML
32 })
33     public Todo getHTML() {
34         Todo todo = new Todo();
35         todo.setSummary("XML Todo Summary");
36         todo.setDescription("Todo
37 Description");
38         return todo;
39     }
40 }

```

```

41
42 -- TodoTest.java
43 public class TodoTest {
44     public static void main(String[] args)
45     {
46         ClientConfig config = new
47 ClientConfig();
48         Client client =
49             ClientBuilder.newClient(config);
50         WebTarget target =
51             client.target(getBaseURI());
52         String xmlResponse =
53 target.path("rest")
54     .path("todo").request()
55     .accept(MediaType.TEXT_XML)
56     .get(String.class);
57         String xmlAppResponse
58 =target.path("rest")
59     .path("todo").request()
60     .accept(
61         MediaType.APPLICATION_XML)
62     .get(String.class);
63         String jsonResponse =
64 target.path("rest")
65     .path("todo").request()
66     .accept(
67         MediaType.APPLICATION_JSON)
68     .get(String.class);
69         (...)
70     }
71     private static URI getBaseURI() {
72         return
73 UriBuilder.fromUri("http://localhost:8080
74 /com.vogella.jersey.jaxb").build();
75     }
76 }
77
78
79
80

```

```

1  -- Account.java
2  @Entity
3  public class Account {
4      @Id          @GeneratedValue
5      Long id;
6      String username;
7      @JsonIgnore
8      private String password;
9      @OneToMany(mappedBy = "account")
10     private Set<Bookmark> bookmarks = new
11     HashSet<>();
12
13     // constructors, getters & setters
14     omitted
15 }
16 --AccountRepository.java
17 public interface AccountRepository extends
18 JpaRepository<Account, Long> {
19     Optional<Account> findByUsername(String
20 username);

```

```

21 }
22 -- Bookmark.java
23 @Entity
24 public class Bookmark {
25     @Id          @GeneratedValue
26     private Long id;
27
28     @JsonIgnore    @ManyToOne
29     private Account account;
30
31     private String uri;
32     private String description;
33     private Bookmark() { } // JPA only
34
35     // constructors , getters & setters
36     omitted
37 }
38
39
40

```

```

42 -- BookmarkRestController.java
43 @RestController @RequestMapping("/{userId}/bookmarks")
44 class BookmarkRestController {
45     private final BookmarkRepository bookmarkRepository;
46     private final AccountRepository accountRepository;
47     @Autowired
48     BookmarkRestController(BookmarkRepository bookmarkRepository,AccountRepository
49 accountRepository) {
50         this.bookmarkRepository = bookmarkRepository;
51         this.accountRepository = accountRepository;
52     }
53     @RequestMapping(method = RequestMethod.GET)
54     Collection<Bookmark> readBookmarks(@PathVariable String userId) {
55         this.validateUser(userId);
56         return this.bookmarkRepository.findByAccountUsername(userId);
57     }
58     @RequestMapping(method = RequestMethod.POST)
59     ResponseEntity<?> add(@PathVariable String userId, @RequestBody Bookmark input) {
60         this.validateUser(userId);
61         return this.accountRepository.findByUsername(userId)
62             .map(account -> {
63                 Bookmark result = bookmarkRepository.save(new Bookmark(account,
64                     input.getUri(), input.getDescription()));
65                 URI location = ServletUriComponentsBuilder
66                     .fromCurrentRequest().path("/{id}")
67                     .buildAndExpand(result.getId()).toUri();
68                 return ResponseEntity.created(location).build();
69             }).orElse(ResponseEntity.noContent().build());
70     }
71     @RequestMapping(method = RequestMethod.GET, value =("/{bookmarkId}")
72     Bookmark readBookmark(@PathVariable String userId, @PathVariable Long bookmarkId) {
73         this.validateUser(userId);
74         return this.bookmarkRepository.findOne(bookmarkId);
75     }
76     private void validateUser(String userId) {
77         this.accountRepository.findByUsername(userId).orElseThrow(
78             () -> new UserNotFoundException(userId));
79     }

```

```

1  -- ProducerTrigger.java
2  @Singleton
3  @Startup
4  public class ProducerTrigger {
5      @Resource ManagedExecutorService mes;
6
7      @PostConstruct
8      public void trigger() {
9          mes.execute(new Producer());
10         System.out.println("Producer
11 triggered...");
12     }
13 }
14
15 --KafkaMDB.java
16 @MessageDriven(activationConfig = {
17     @ActivationConfigProperty(
18         propertyName = "clientId",
19         propertyValue = "testClient"),
20     @ActivationConfigProperty(
21         propertyName = "groupIdConfig",
22         propertyValue = "testGroup"),
23     @ActivationConfigProperty(
24         propertyName = "topics",
25         propertyValue =
26 "${ENV=TOPIC_NAME}"),
27     @ActivationConfigProperty(
28         propertyName =
29 "bootstrapServersConfig",
30         propertyValue =
31 "${ENV=KAFKA_BROKER}"),
32     @ActivationConfigProperty(
33         propertyName = "keyDeserializer",
34         propertyValue =
35 "...StringDeserializer"),
36     @ActivationConfigProperty(
37         propertyName = "valueDeserializer",
38         propertyValue = "
39 ...StringDeserializer")
40 })
41 }
42 public class KafkaMDB implements
43 KafkaListener {
44     public KafkaMDB() { }
45
46     @OnRecord( topics={"payara-kafka-mdb-
47 topic"})
48     public void
49 test(ConsumerRecord<Object,Object> record)
50 {
51     System.out.println("Payara Kafka MDB
52 record " + record );
53 }
54
55 }
56

```

```

57 --Producer.java
58 public class Producer implements Runnable
59 {
60     KafkaProducer<String, String> produc;
61     String topic = null;
62     public Producer() {
63         Properties consumerProps = new
64 Properties();
65         consumerProps.put(
66
67 ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
68
69 System.getenv().getOrDefault("KAFKA_BROKER
70 ", "192.168.99.100:9092"));
71         consumerProps.put(
72             ProducerConfig.KEY_SERIALIZER_CLASSG,
73             "...StringSerializer");
74         consumerProps.put(
75
76 ProducerConfig.VALUE_SERIALIZER_CLASS,
77             "...StringSerializer");
78
79         produc = new
80 KafkaProducer<>(consumerProps);
81         topic = System.getenv()
82             .getOrDefault("TOPIC_NAME", "test-
83 topic");
84     }
85
86     static Random rnd = new Random();
87
88     @Override
89     public void run() {
90         (...) // "Producing to topic
91         while (true) {
92             String key = "";
93             produc.send(
94                 new ProducerRecord(topic, "key-" +
95 rnd.nextInt(10), "val-" +
96 rnd.nextInt(10)),
97                 new Callback() {
98                     public void
99 onCompletion(RecordMetadata rm, Exception
100 excptn) {
101                     (...) // sent data
102                 }
103             });
104             try {
105                 Thread.sleep(10000); //take it
106 easy!
107             } catch (InterruptedException ex) {
108                 (...) // log exception
109             }
110         }
111     }
112

```

Code 7 - A store catalogue

```

1  -- StockApplication.java
2  @ApplicationPath("/api")
3  @Path("/items")
4  @Stateless
5  @TransactionAttribute(TransactionAttributeType.
6  NEVER)
7  public class StockApplication extends
8  Application {
9      @EJB CatalogueBean catalogue;
10
11      @POST @Consumes(MediaType.APPLICATION_JSON)
12      @Produces(MediaType.APPLICATION_JSON)
13      public Item
14      createItem(StockCreateRequest item) throws
15      IOException {
16          return catalogue
17              .addItem(item.getName(),
18              item.getStock()); }
19      @GET @Path("/{id}")
20      @Produces(MediaType.APPLICATION_JSON)
21      public Item getItem(@PathParam("id") int
22      id) throws IOException {
23          return catalogue.getItem(id); }
24      @PUT @Path("/{id}")
25      @Consumes(MediaType.APPLICATION_JSON)
26      @Produces(MediaType.APPLICATION_JSON)
27      public Item updateStock(@PathParam("id")
28      int id, StockUpdateRequest update) throws
29      IOException {
30          return catalogue
31              .addStock(id, update.getAmmount());
32      }
33  }
34  --Cart.java
35  @ManagedBean
36  @SessionScoped
37  public class Cart {
38      @EJB CartBean cart;
39
40      public Collection<Item> getItems() {
41          return cart.getItems();
42      }
43      public void add(Item item) {
44          if (!cart.add(item)) // Could not add
45          }
46      public void remove(Item item) {
47          if (!cart.remove(item))// Could not
48      remove
49      }
50      public void purchase() {
51          try {
52              if (cart.purchase()) // successful
53              else //.No items in cart
54          } catch (IOException e) {
55              // Purchase failed
56          }
57      }
58  }
59
60  --CatalogueBean.java
61  @Stateless
62  public class CatalogueBean {
63      (...)
64      public List<Item> getCatalogue() { (...) }
65      public Item getItem(int id) { (...) }
66      public Item addItem(String name, int
67      stock)
68      { (...) }
69      public Item addStock(int id, int amount)
70      { (...) }
71      public void purchase(int id) { (...) }
72  }
73  -- Catalogue.java
74  @ManagedBean
75  public class Catalogue {
76      @EJB CatalogueBean catalogue;
77      public Collection<Item> getItems()
78      throws IOException {
79          return this.catalogue.getCatalogue();
80      }
81  }
82  -- CartBean.java
83  @Stateful
84  public class CartBean{
85      List<Item> items = new ArrayList<>();
86      public boolean add(Item item) {
87          return this.items.add(item); }
88      public boolean remove(Item item) {
89          return this.items.remove(item); }
90      public List<Item> getItems() {
91          return this.items; }
92      @TransactionAttribute(TransactionAttributeType.
93      REQUIRED)
94      public boolean purchase() throws
95      IOException {
96          if (items.isEmpty()) { return false; }
97          for (Item item : items)
98              item.purchase();
99          clear();
100         return true;
101     }
102     @Remove
103     public void clear() {
104         this.items.clear();}
105 }
106 -- Item.java
107 public class Item{
108     private CatalogueBean catalogue;
109     private int id;
110     private String name;
111     private int stock;
112     // constructors, getter setter omitted
113 }

```