

# Marketplace project description

---

## Nota Bene

---

### Get list of all user's orders

We need to show all user's orders with bought products and download links for each product. I found several solutions:

- Get all user's orders, then for each order get order item. Obvious that it is not good solution because we have unpredictable number of database queries.
- Use OneToMany relationship. It looks better, but we can suggest that we don't know how many data will be pulled cascaded from DB (other tables depended on `order` may have OneToMany relationship).
- Get all order items with one query and work with them. But we will need to groupe them in some way.

The last solution was chosen. We get all order items then map them to orders. You can found that `OrderItem` have `@OneToMany` relationship to `download`, we admit that because `download` table have only one relationship (with `order_item`).

Final sql generated by `OrderItemDaoImp.find()` looks like this:

```
1  SELECT
2      order_item.id,
3      product.id,
4      book.id,
5      "order".id,
6      download.id
7      -- Other data from all tables
8  FROM
9      order_item
10     LEFT OUTER JOIN product ON order_item.product_id = product.id
11     LEFT OUTER JOIN book ON product.id = book.id
12     LEFT OUTER JOIN "order" ON order_item."order_id" = "order".id
13     LEFT OUTER JOIN download ON order_item.id = download.order_item_id
14 WHERE
15     "order"."user_id" = 1;
16
```

Convert `List<IOrderItem>` to `List<Order>`:

```
1  @Override public List<IOrder> getOrdersByUserId(final Integer userId) {
2      final List<IOrderItem> orderItemList =
3      orderItemService.getOrderItemsByUserId(userId);
4
5      final Set<IOrder> orderSet = new HashSet<>();
6      orderItemList.forEach(orderItem ->
7          orderSet.add(orderItem.getOrder().addOrderItem(orderItem))
8      );
9
10     final List<IOrder> orderList = new ArrayList<>(orderSet);
11     orderList.sort((o2, o1) -> o1.getCreated().compareTo(o2.getCreated()));
12     return orderList;
13 }
```

We use `Set` to avoid to search proper **order** in list, on the other hand - we need to convert them to sorted list.

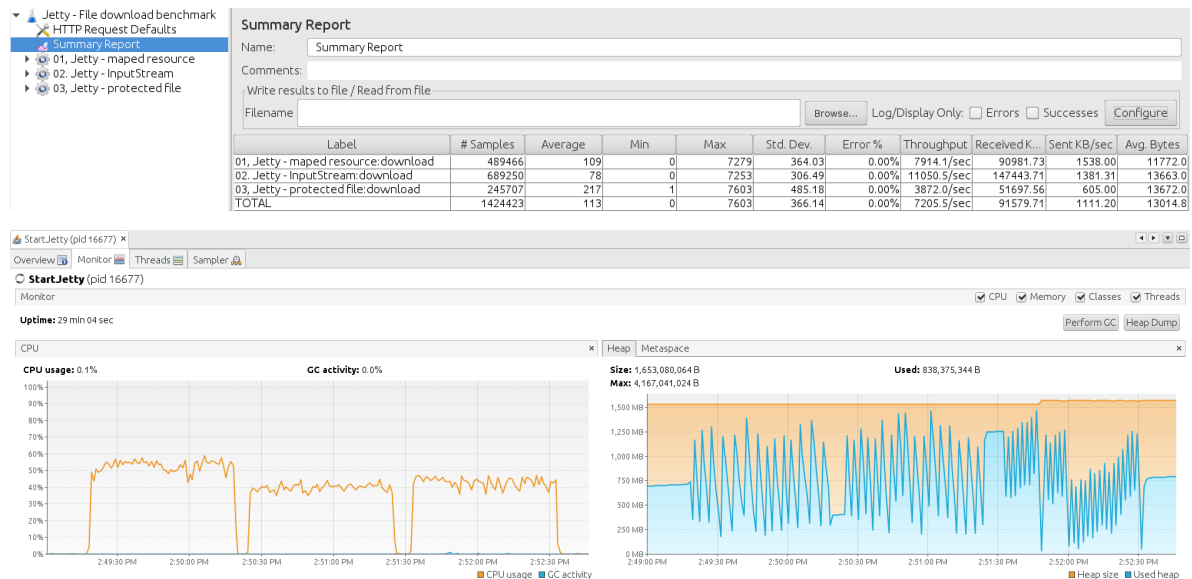
# Download benchmark (public vs static files)

## Test on Jetty

We need to choose how to serve static files. Let's consider two solutions: serving by servlet and serving by our app as `InputStream`.

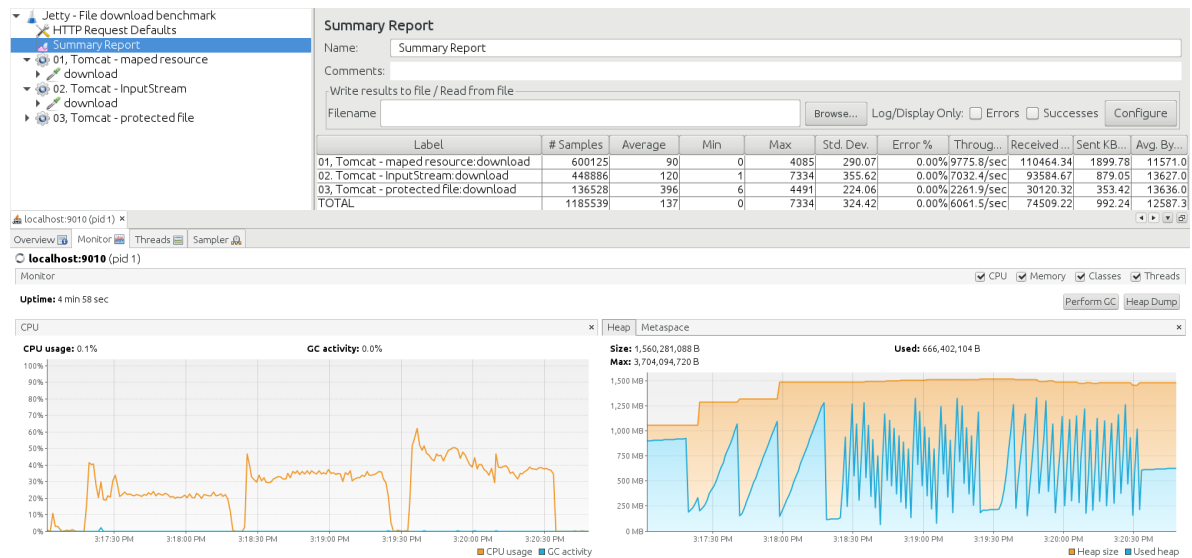
We perform three tests:

1. access file from servlet (we will use Jetty)
2. access file from our app
3. access protected file from app (we will need to make database query to get the file)



As we can see we have similar resource usage, and response time for both solutions. Even Jetty shows little worse performance than our app. So let's test it on Tomcat.

## Test on Tomcat



OK. On Tomcat we got expected results. But why access to protected file took two times as long as access on Jetty? I don't know.

As conclusion we can say that both solutions are very slow and resource hungry. Java servlets are not suitable for file serving, we should use proper tools like Nginx for that.

# Deployment

## Java in Docker - the right way (almost)

### Dockerfile

```
1 FROM maven:3.6.3-jdk-8 AS builder
2 WORKDIR /usr/src/app
3
4 COPY pom.xml .
5 COPY dao/pom.xml dao/pom.xml
6 COPY service/pom.xml service/pom.xml
7 COPY web/pom.xml web/pom.xml
8 COPY web/src/main/webapp/WEB-INF/web.xml web/src/main/webapp/WEB-INF/web.xml
9 RUN mvn package && mvn clean
10
11 COPY ./ .
12 RUN mvn package -DskipTests
13
14
15
16 FROM tomcat:9-jre8-alpine
17 WORKDIR ${CATALINA_HOME}
18
19 RUN rm -rf webapps/*
20 COPY config/tomcat/server.xml config/tomcat/context.xml conf/
21 COPY --from=builder /usr/src/app/web/target/${APP_NAME}.war webapps/ROOT.war
22
23 RUN addgroup www-data && \
24     adduser -D -H -u 1000 -s /bin/bash www-data -G www-data && \
25     chown -R www-data:www-data webapps temp
26
27 USER www-data
28 CMD ["catalina.sh", "run"]
```

.war size 47Mb, final docker image size 200Mb and `builder` image size is bigger then 800Mb.

NB In modern world we should use [Embedded Tomcat](#).