**1. What artifact will be generated?**

If in your **pom.xml** (Maven build section) you have:

<build>

<finalName>Food-System</finalName>

</build>

👉 Then, after you run mvn package, the **artifact name** in the target/ folder will be:

* **WAR project** → Food-System.war
* **JAR project** → Food-System.jar

**2. Why might this cause a deployment issue in Tomcat?**

Tomcat uses the **WAR file name** to determine the **context path** of the application:

* Food-System.war → deployed as http://localhost:8080/Food-System/
* myapp.war → deployed as http://localhost:8080/myapp/
* ROOT.war → deployed as http://localhost:8080/ (default context root)

**The problem**

* The artifact name has a **hyphen (-)**.
* Some configurations, links, or hardcoded paths in your code (like /FoodSystem or /foodsystem) may not match the actual Tomcat context (/Food-System).
* This mismatch leads to **404 errors** or deployment failures.

**✅ Best practice / Fix**

1. **If you want the app accessible at root context:**
2. <finalName>ROOT</finalName>

→ Deploys as http://localhost:8080/

1. **If you want a clean context path:**
2. <finalName>FoodSystem</finalName>

→ Deploys as http://localhost:8080/FoodSystem/

👉 Would you like me to also show you the **exact steps in Eclipse** to fix this so Tomcat deploys your app correctly (without 404 issues)?

**You said:**

download the given repo and show the list of files

**ChatGPT said:**

I don’t have direct access to download repos from the internet on my own. But if you share the **repository link** (GitHub/GitLab/Bitbucket/etc.), I can guide you to:

* Fetch the repo contents.
* Show you the **list of files/folders** (like a tree structure).
* Even help analyze pom.xml, src/ files, or configs.  
    
    
    
  Perfect 👍 I can help you solve the **Git + GitHub integration with Maven project** SQB tasks from the image. Let’s go one by one:
* **1. Initialize Git repository**
* git init
* **2. Set Git username & email globally**
* git config --global user.name "YourName"
* git config --global user.email "your@email.com"
* **3. Connect local repo to GitHub**
* git remote add origin https://github.com/username/Food-Ordering-System.git
* **4. Commit staged changes with message**
* git commit -m "Added Order Service for Food Ordering System"
* **5. Create and switch to a new branch feature/payment**
* git checkout -b feature/payment
* **6. Remove wrongly staged file (temp.txt) but keep it locally**
* git reset HEAD temp.txt
* **7. Verify list of remote repositories**
* git remote -v
* **8. Temporarily save & restore uncommitted changes**
* Save changes:
* git stash
* Restore changes:
* git stash pop
* **9. Merge branch feature/Reviews into main**
* git checkout main
* git merge feature/Reviews
* **10. Clone project from GitHub**
* git clone https://github.com/username/Food-Ordering-System.git
* **11. Push new branch (feature/real-time-status)**
* git push origin feature/real-time-status
* **12. Use SSH instead of HTTPS**
* Add remote using SSH:
* git remote set-url origin git@github.com:username/Food-Ordering-System.git
* Or when cloning:
* git clone git@github.com:username/Food-Ordering-System.git
* ✅ With these commands, you can:
* Initialize → configure → branch → merge → stash → push → switch to SSH.

**1. Pull official nginx:latest image**

docker pull nginx:latest

**2. Run nginx as container mynginx exposing port 8090 → 80**

docker run -d --name mynginx -p 8090:80 nginx:latest

**3. Check which host port is mapped to container port 8080**

docker ps

(Shows mapping like 0.0.0.0:32768->8080/tcp).

**4. Stop a container consuming high CPU**

docker stop <container\_id\_or\_name>

**5. Run web\_app container exposing internal port 5000 → host 8080**

docker run -d --name web\_app -p 8080:5000 web\_app

**6. Pull official python image, run it, and list containers**

docker pull python

docker run -it --name mypython python

docker ps -a

**7. Verify if web-app is accessible on port 8080 (check conflict)**

* First check if port 8080 is already in use:

docker ps

* If conflict exists, stop the container using that port:

docker stop <container\_id>

**8. If app doesn’t run on 8080, rebuild to run on 9090**

docker run -d --name web\_app -p 9090:5000 web\_app

**9. Stop and start a running container**

docker stop <container\_id\_or\_name>

docker start <container\_id\_or\_name>

**10. Check container status**

docker ps -a  
  
  
  
**1. docker-compose.yml file (Food Ordering System + PostgreSQL)**

version: '3.8'

services:

food-ordering-app:

image: your-dockerhub-username/food-ordering-system:latest

container\_name: food\_ordering\_app

ports:

- "7078:8080" # Host:Container mapping

depends\_on:

- postgres-db

environment:

- SPRING\_DATASOURCE\_URL=jdbc:postgresql://postgres-db:5432/fooddb

- SPRING\_DATASOURCE\_USERNAME=fooduser

- SPRING\_DATASOURCE\_PASSWORD=foodpass

postgres-db:

image: postgres:15

container\_name: postgres\_food\_db

restart: always

environment:

POSTGRES\_DB: fooddb

POSTGRES\_USER: fooduser

POSTGRES\_PASSWORD: foodpass

ports:

- "5432:5432"

volumes:

- postgres\_data:/var/lib/postgresql/data

volumes:

postgres\_data:

**2. Pull and run your Maven-built image**

Docker Compose will automatically pull the image from **Docker Hub** if not available locally:

docker-compose up -d

This runs your app on [**http://localhost:7078**](http://localhost:7078).

**3. Run PostgreSQL container & insert data**

Once containers are up, access the DB:

docker exec -it postgres\_food\_db psql -U fooduser -d fooddb

Then insert sample data:

INSERT INTO users (id, name, role) VALUES (1, 'Admin', 'Manager');

INSERT INTO menu\_items (id, name, price) VALUES (1, 'Burger', 150);

✅ With this setup:

* **Food Ordering System** runs inside a container, exposed on port **7078**.
* **Postgres DB** runs inside another container, exposed on port **5432**.
* Both are connected via Docker Compose networking.