

SE-LAB-SET3

i will give you question , explain in detail , simple words and in detail say what that function does

Perfect! Send me the question, and I'll break it down step by step, explaining clearly what each part and function does in simple words.

Q1. Software Requirement Specification (SRS)

Social Media Mini App (SMMA) is a lightweight web-based application designed for users to connect, share posts, and interact with each other. It includes basic modules such as user login/registration, posting feeds, liking/commenting, and profile management. The application demonstrates the working of a scalable, community-driven social platform in a simplified manner..

Based on this scenario, answer the following: [10 Marks]

1. Draft an Abstract for SMMA highlighting its scope – 4M
2. List five functional requirements for SMMA – 2M
3. List three non-functional requirements for SMMA (security, scalability, usability) – 2M
4. Identify and classify the users of SMMA (admin, registered user, guest) – 2M

Got it 👍 Let's go step by step and explain in **simple words + detailed meaning** for each sub-question.

Q1. Software Requirement Specification (SRS) - SMMA

1. Abstract (4 Marks)

An **abstract** is a short summary that tells what the project is, why it exists, and what it covers.

Answer:

The *Social Media Mini App (SMMA)* is a lightweight web-based platform created to help users connect and interact online in a simple way. It allows users to register, log in, create and share posts, like and comment on others' posts, and manage their profiles. The app focuses on demonstrating the basic features of a community-driven social media platform in a scalable and user-friendly manner. Its scope includes building a secure and interactive environment where users can engage with each other, while also ensuring smooth performance and ease of use.

2. Functional Requirements (2 Marks)

Functional requirements = features or services the system *must do*.

Answer (any 5):

1. User registration and login (sign-up, sign-in, logout).
2. Create, edit, and delete posts (text/images).
3. Like and comment on posts.
4. Manage user profiles (update name, bio, picture).
5. Search for other users or posts.

3. Non-Functional Requirements (2 Marks)

Non-functional requirements = quality attributes, not direct features.

Answer (any 3):

1. **Security:** User data and login credentials must be encrypted and protected.
 2. **Scalability:** The system should handle growth in users and posts without crashing.
 3. **Usability:** The interface must be simple, clean, and easy for anyone to use.
-

4. Users of SMMA (2 Marks)

Here we classify the types of users and their role.

Answer:

1. **Admin** – Manages the platform, monitors user activity, removes inappropriate content.
 2. **Registered User** – A person who signs up and can post, like, comment, and manage their profile.
 3. **Guest** – A visitor who can only view limited content but cannot post or interact until registered.
-

✅ That covers all **10 Marks** with clear, simple explanations.

Do you also want me to **explain what each part (abstract, functional, non-functional, users) actually means in SRS** so it's extra clear for exams?

QII: Maven Web Application Development

[30M]

1. You are required to configure and enhance the SMMA Maven project:
Clone the GitHub repo: <https://github.com/savram674/SMMA.git>
and perform a clean build. Document the Maven lifecycle phases involved. – 10M
2. Add dependency for JSP Standard Tag Library (JSTL 1.2) and explain its use in JSP pages. – 5M
3. A developer removes the <dependencies> section completely.
Will Maven still build the project? What issues might occur during testing? – 2 M
3. Inside your pom.xml, locate the <build> section. Then, add or modify the <directory> tag to specify the new output directory, which in this case is build_output/-. – 3M
4. Add dependency for MongoDB Driver and describe how MongoDB can be integrated for storing user posts. – 5M
5. Run mvn dependency:tree command and explain the output. -- 2M
6. Describe the steps to deploy the generated WAR file in Tomcat. – 3M

Perfect 🙌 Let's go through **QII: Maven Web Application Development** step by step, in **simple words + detailed explanation** for each part.

QII: Maven Web Application Development [30M]

1. Clone & Clean Build + Maven Lifecycle Phases (10M)

- First, clone the repo:

```
git clone https://github.com/savram674/SMMA.git
cd SMMA
mvn clean install
```

- When you run `mvn clean install`, Maven executes its **lifecycle phases**.

The main phases are:

1. **validate** → Checks if project structure and configuration (pom.xml) are correct.
2. **compile** → Compiles the Java source code (`src/main/java`).
3. **test** → Runs unit tests using JUnit/TestNG.
4. **package** → Packages compiled code into a distributable format (e.g., WAR for web app).
5. **verify** → Runs integration tests to check if WAR/JAR works properly.
6. **install** → Installs the package into local Maven repository (`.m2/repository`) so it can be reused.
7. **deploy** → (Optional) Uploads final artifact to remote repo (e.g., Nexus, Artifactory).

👉 **Document:** In this project, after cloning, `mvn clean install` will clean old files, compile code, run tests, create WAR, and install it locally.

2. Add JSTL Dependency + Explain Use (5M)

Inside `pom.xml`, add:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

Use in JSP pages:

- JSTL (JavaServer Pages Standard Tag Library) gives ready-made tags for common tasks.
- Instead of writing Java code in JSP, we use JSTL tags:
 - `<c:if>` → conditional statements.
 - `<c:forEach>` → loops.
 - `<c:out>` → safely display values.
- Example:

```
<c:forEach var="post" items="${posts}">
  <p>${post.content}</p>
</c:forEach>
```

👉 This makes JSP cleaner, easier to maintain, and avoids Java code inside HTML.

3. If `<dependencies>` Section Removed (2M)

- Yes, Maven will still try to build the project.
- But:
 - No external libraries (like Servlet API, JSTL, MongoDB driver) will be available.
 - Compilation may fail if classes from those libraries are used.
 - During testing, `ClassNotFoundException` or `NoClassDefFoundError` will occur.

👉 Basically, without `<dependencies>`, only core Java works, no extra features.

4. Modify ``<directory>`` in ``<build>`` (3M)

Inside ``pom.xml``:

```
<build>
  <directory>build_output</directory>
</build>
```

👉 This changes Maven's default output folder from ``target/`` to ``build_output/``. So compiled classes, WAR file, etc. will be placed inside ``build_output/``.

5. Add MongoDB Driver + Explain Integration (5M)

In ``pom.xml``:

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongodb-driver-sync</artifactId>
  <version>4.11.1</version>
</dependency>
```

Integration with SMMA:

- MongoDB can be used to store posts, likes, comments, and user data.
- Example code to insert a post:

```
MongoClient client = MongoClient.create("mongodb://localhost:27017");
MongoDatabase db = client.getDatabase("smmaDB");
MongoCollection<Document> posts = db.getCollection("posts");

Document post = new Document("user", "Anirudh")
    .append("content", "Hello World!")
    .append("likes", 0);
posts.insertOne(post);
```

👉 This allows storing and retrieving user posts dynamically.

6. mvn dependency:tree (2M)

Command:

```
mvn dependency:tree
```

What it does:

- Shows all dependencies of the project in a **tree structure**.
- Includes **direct dependencies** (declared in ``pom.xml``) and **transitive dependencies** (dependencies of dependencies).
- Helps to check for version conflicts.

Example Output (simplified):

```
[INFO] com.smma:smma:war:1.0
[INFO] |-- javax.servlet:jstl:1.2
[INFO] |-- org.mongodb:mongodb-driver-sync:4.11.1
[INFO] |__ junit:junit:4.13.2 (test)
```

7. Deploy WAR File in Tomcat (3M)

Steps:

1. After `mvn package`, WAR file is generated in `target/` (or `build_output/`).
2. Copy `smma.war` into `TOMCAT_HOME/webapps/` directory.
3. Start Tomcat server (`bin/startup.sh` or `startup.bat`).
4. Open in browser:

```
http://localhost:8080/smma
```

👉 Now the web app runs on Tomcat.

✅ That completes **30 Marks** with full explanations.

Do you want me to also **draw a neat table mapping Marks → Points (like what to write in exam)** so it's easier to present?

QIII. - Git & GitHub Integration with Maven Project [30M]

Using the Maven project created above, perform the following Git tasks:

1. Configure username and email in Git (show commands). - 2M
2. Stage modified files and undo staging without losing changes. - 2M
3. Commit with the message: "Added feed.jsp for user posts". - 2M
4. View full details of a commit (hash, author, date, and changes). - 2M
5. Create a new branch comment-feature and switch to it with one command. - 2M
6. Push the branch comment-feature into main. Demonstrate a fast-forward merge. - 3M
7. Simulate a merge conflict between feed-feature and main and resolve it. - 3M
8. You want to see the differences introduced by the last commit on your branch. - 2M
9. Demonstrate git pull with explanation of how it differs from git fetch. - 3M
10. Use git revert to undo the changes of a specific commit (without deleting history). - 3M
11. Modify the commit message of the last commit. - 2M
12. You want to see what changes are present between your local branch and main on GitHub. - 2M
13. A teammate wants to work on the SMMA. Which Git command will he use to copy the project from GitHub to his machine?.

Great 👍 Let's go step by step for **QIII: Git & GitHub Integration with Maven Project**.

I'll give **commands + simple explanations** so you can easily write them in exams.

QIII. Git & GitHub Integration with Maven Project [30M]

1. Configure username and email (2M)

```
git config --global user.name "YourName"
git config --global user.email "your@email.com"
```

👉 Sets your identity for commits (name + email).

2. Stage modified files & undo staging (2M)

- Stage:

```
git add filename
```

- Undo staging (keep changes in working directory):

```
git reset filename
```

👉 `git add` = prepare for commit, `git reset` = remove from staging but keep edits.

3. Commit with message (2M)

```
git commit -m "Added feed.jsp for user posts"
```

👉 Saves staged changes with a meaningful message.

4. View full commit details (2M)

```
git show commit_hash
```

👉 Shows commit hash, author, date, and all code changes.

5. Create branch + switch in one command (2M)

```
git checkout -b comment-feature
```

👉 Creates branch **comment-feature** and moves to it immediately.

6. Push branch + fast-forward merge (3M)

- Push branch to remote:

```
git push origin comment-feature
```

- Merge into main (fast-forward if no conflict):

```
git checkout main  
git merge comment-feature
```

👉 Fast-forward = main simply moves ahead to the same commit as branch.

7. Simulate merge conflict & resolve (3M)

- Create conflict: both `feed-feature` and `main` edit same line.
- Merge attempt:

```
git checkout main  
git merge feed-feature
```

👉 Git marks conflict in file with `<<<<<<`, `=====`, `>>>>>>`.

- Resolve by editing file manually → keep correct changes.
- Then:

```
git add conflicted_file  
git commit
```

👉 Conflict resolved.

8. See differences of last commit (2M)

```
git show HEAD
```

👉 Displays only changes introduced in the most recent commit.

9. git pull vs git fetch (3M)

- Command:

```
git pull origin main
```

👉 ``git pull = git fetch + git merge``.

- **git fetch** → downloads new commits from GitHub but doesn't merge.
- **git pull** → fetches and merges automatically into your branch.

10. Revert a commit (3M)

```
git revert commit_hash
```

👉 Creates a *new commit* that undoes changes from an old commit.

👉 Unlike ``reset``, it doesn't delete history.

11. Modify last commit message (2M)

```
git commit --amend -m "New commit message"
```

👉 Edits the message of the latest commit.

12. Compare local branch vs main on GitHub (2M)

```
git fetch origin
git diff branch_name origin/main
```

👉 Shows differences between your branch and remote main.

13. Teammate copies project (2M)

```
git clone https://github.com/savram674/SMMA.git
```

👉 ``git clone`` = downloads entire repo with history to teammate's machine.

✅ That's **all 30 Marks** with full commands + explanations.

Do you want me to also prepare a **ready-to-write exam table (Question | Command | Explanation)** so you can directly reproduce it?

QIV. Docker containerization for Maven Application-
Containerize the SMMA application using Docker:

[20M]

1. Write a Dockerfile to:

- Use Tomcat base image
 - Copy WAR file into webapps/
 - Expose port 8080 – 6M
2. Build Docker image with name smma-web and run it. – 3M
 3. Execute a running container to check deployed WAR file path. – 3M
 4. List out all the docker images and containers. – 2M
 5. Pull a Ubuntu image and run it in interactive mode. Create a file inside that and commit so as the retain the changes even while exiting the container. -- 3M
 6. Demonstrate how to pull an image from Docker Hub and tag it locally. – 3M

Perfect 👍 Let's solve **QIV: Docker containerization for Maven Application [20M]** step by step, with **Dockerfile + commands + simple explanations.**

QIV. Docker Containerization for Maven Application [20M]

1. Dockerfile for SMMA (6M)

A **Dockerfile** is a script with instructions to build an image.

```
# Use Tomcat base image
FROM tomcat:9.0

# Copy WAR file generated by Maven into Tomcat webapps directory
COPY target/smma.war /usr/local/tomcat/webapps/

# Expose port 8080 for external access
EXPOSE 8080
```

👉 Explanation:

- **FROM tomcat:9.0** → uses official Tomcat image.
- **COPY** → puts WAR file into Tomcat's `webapps/` folder.
- **EXPOSE 8080** → allows outside world to access app on port 8080.

2. Build & Run Docker Image (3M)

- Build image with name `smma-web`:

```
docker build -t smma-web .
```

- Run container:

```
docker run -d -p 8080:8080 --name smma-container smma-web
```

👉 `-d` runs in background, `-p` maps container port to local port, `--name` names the container.

3. Execute into Running Container (3M)

Check deployed WAR file path:

```
docker exec -it smma-container bash
ls /usr/local/tomcat/webapps/
```

👉 Inside container, you should see `smma.war` and extracted folder.

4. List Docker Images & Containers (2M)

- List images:

```
docker images
```

- List running containers:

```
docker ps
```

- List all containers (including stopped):

```
docker ps -a
```

5. Ubuntu Interactive + Commit Changes (3M)

- Pull and run Ubuntu in interactive mode:

```
docker pull ubuntu
docker run -it ubuntu
```

- Inside container, create file:

```
echo "Hello Docker" > hello.txt
exit
```

- Find container ID:

```
docker ps -a
```

- Commit changes into new image:

```
docker commit <container_id> ubuntu-custom
```

👉 Now `hello.txt` will be retained in `ubuntu-custom`.

6. Pull Image from Docker Hub & Tag Locally (3M)

- Pull image:

```
docker pull nginx
```

- Tag it locally:

```
docker tag nginx mynginx:1.0
```

👉 Now you can use `mynginx:1.0` as a local alias for the pulled image.

✅ That covers **20 Marks** with clear commands + explanations.

Do you want me to also **draw a neat marks-distribution table** (Question | Command | Explanation | Marks) so it's exam-ready?

QV. DOCKER COMPOSE

[10M]

Create a multi-container setup for SMMA using Docker Compose:

1. Write a docker-compose.yml to run two services: – 5M

□ Tomcat with SMMA app (port 8080)

□ MongoDB database (port 27017 with volume for persistence)

2. Use Docker Compose commands to:

- Start both containers – 2M
- Insert a sample post into MongoDB and retrieve it -- 3 M

Perfect 🍌 Let's solve **QV: Docker Compose for SMMA [10M]** step by step, with **docker-compose.yml + commands + explanations**.

QV. Docker Compose - Multi-Container Setup [10M]

1. docker-compose.yml (5M)

```
version: '3.8'

services:
  smma-app:
    image: smma-web          # Docker image of SMMA created earlier
    container_name: smma-app
    ports:
      - "8080:8080"          # Expose Tomcat port
    depends_on:
      - mongo
    networks:
      - smma-network

  mongo:
    image: mongo:6.0         # Official MongoDB image
    container_name: smma-mongo
    ports:
      - "27017:27017"        # Expose MongoDB port
    volumes:
      - mongo-data:/data/db  # Persistent storage
    networks:
      - smma-network

volumes:
  mongo-data:

networks:
  smma-network:
```

Explanation:

- **smma-app** → runs Tomcat with SMMA WAR. Depends on ``mongo``.
- **mongo** → runs MongoDB with a persistent volume ``mongo-data``.
- **networks** → allows containers to communicate.

2. Start Both Containers (2M)

```
docker compose up -d
```

- ``-d`` → runs containers in background.
- To check:

```
docker compose ps
```

✅ Both ``smma-app`` and ``mongo`` should be running.

3. Insert & Retrieve Sample Post in MongoDB (3M)

1. Connect to MongoDB container:

```
docker exec -it smma-mongo mongo
```

2. Switch to database:

```
use smmaDB
```

3. Insert a sample post:

```
db.posts.insertOne({user: "Anirudh", content: "Hello SMMA!"})
```

4. Retrieve posts:

```
db.posts.find().pretty()
```

Explanation:

- `insertOne()` → adds a document (sample post).
- `find().pretty()` → lists documents in a readable format.

✅ That completes **10 Marks** with **docker-compose.yml + commands + explanation**.

I can also prepare a **single-page cheat-sheet for all Docker + Docker Compose commands for SMMA** so you can quickly revise for exams. Do you want me to do that?

```
$ docker run -d -p 8080:8080 --name smma-container smma-web
docker: Error response from daemon: Conflict. The container name "/smma-container" is
already in use by container
"4dbe7ea4240829b355db06fb5dd4a02592481cce6e2c20d62c3737ed9e376160". You
have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
```

Ah! That error is very clear: Docker says **you already have a container named** `smma-container`, so you can't create another with the same name. You have a few options:

Option 1: Remove the old container

If you don't need it anymore:

```
docker rm smma-container
```

- Then rerun your command:

```
docker run -d -p 8080:8080 --name smma-container smma-web
```

⚠ *If the container is still running, you need to stop it first:*

```
docker stop smma-container
docker rm smma-container
```

Option 2: Rename the new container

Just give a different name:

```
docker run -d -p 8080:8080 --name smma-container-2 smma-web
```

Option 3: Force remove old container and run new

```
docker rm -f smma-container  
docker run -d -p 8080:8080 --name smma-container smma-web
```

💡 **Tip:** Always check running/stopped containers before reusing names:

```
docker ps -a
```

If you want, I can **give a one-line command to safely replace the old container with the new one without manual stopping/removing**. Do you want me to do that?