SE-LAB-SET1

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

Perfect! Go ahead and give me the question. I'll explain it thoroughly, in simple words, step by step, and clarify exactly what each function or concept does.

QI. Software Requirement Specification (SRS)

Smart Assessment Hub (SAH) is a technology-driven platform that streamlines examination and evaluation processes. It enhances learning outcomes through automated grading, real-time feedback, and data-driven insights. The platform ensures transparency, scalability, and efficiency in assessments. By integrating advanced analytics, it empowers educators and institutions to make informed decisions for continuous improvement.

Based on this scenario, answer the following:[10 Marks]
a. Abstract - 4M
b. Functional Requirements-2M
c. Non-Functional Requirements-2M
d. Identification of Users-2

Sure! Let's break this **SRS question** step by step. I'll explain each part clearly, in simple words, and provide examples based on the **Smart Assessment Hub (SAH)** scenario.

---

a. Abstract (4 Marks)

What is it?
The abstract is a **brief summary** of the system. It explains what the system is, its purpose, and what it aims to achieve. It should be concise but cover the essence.

Example for SAH:

> ***Abstract:***
> *Smart Assessment Hub (SAH) is a modern, technology-driven platform designed to simplify the process of examination and evaluation. It automates grading, provides real-time feedback to students, and generates insights using data analytics. The system ensures transparency, efficiency, and scalability in assessments. By offering detailed reports and performance analysis, SAH empowers educators and institutions to make informed decisions for continuous improvement in learning outcomes.*

Explanation:

- Mention **what the system is**: technology-driven platform.
- Mention **key features**: automated grading, real-time feedback, data analytics.
- Mention **benefits**: transparency, efficiency, informed decisions.

---

b. Functional Requirements (2 Marks)

What is it?
Functional requirements describe **what the system should do**—the tasks and functions it must perform.

Example for SAH:

1. **Automated Grading:** The system must grade exams automatically.
2. **Real-Time Feedback:** Students receive instant feedback on their performance.
3. **Report Generation:** Generate performance reports for students and educators.
4. **Question Bank Management:** Create, edit, and store questions for exams.
5. **Exam Scheduling:** Schedule and manage online assessments.

Explanation:

- These are **actions the system must perform**.

- Think of it as the “jobs” SAH does.

---

c. Non-Functional Requirements (2 Marks)

What is it?

Non-functional requirements describe **how the system should behave**—performance, reliability, usability, security, etc.

Example for SAH:

1. **Performance:** The system should handle 1000+ concurrent users without delays.
2. **Scalability:** It should support the addition of more users or exams easily.
3. **Security:** User data must be encrypted and secure from unauthorized access.
4. **Usability:** Interface should be user-friendly for both students and teachers.
5. **Reliability:** The system must have 99% uptime for exams and reporting.

Explanation:

- Focus on **quality attributes** rather than specific tasks.
- They define **how well** the system performs its functions.

---

d. Identification of Users (2 Marks)

What is it?

This identifies **who will use the system**—the stakeholders and end users.

Example for SAH:

1. **Students:** Take exams, receive feedback, view performance reports.
2. **Teachers / Educators:** Create exams, grade, analyze results, provide feedback.
3. **Administrators:** Manage users, maintain the system, schedule exams.
4. **Institution / School Management:** Access analytics to make decisions on learning outcomes.

Explanation:

- Users can be **primary (direct users)** or **secondary (indirect users)**.
- Understanding users helps **design the system effectively**.

✅ Summary Table (Optional for clarity)

Section	Description	Example (SAH)
Abstract	Brief summary of system	Platform automating exams, grading, feedback
Functional Requirements	Tasks system must perform	Grading, feedback, report generation
Non-Functional Requirements	System qualities	Performance, scalability, security
Users	Who uses the system	Students, teachers, admin, management

---

If you want, I can **write it in a ready-to-submit SRS format** exactly for 10 marks so you can use it in your answer sheet.

Do you want me to do that?

QII. Maven Web Application Development (30M)

You are hired as a Software Engineer at a tech startup. Your first task involves understanding and setting up a Maven-based Web application for the above project. The template is provided as per the following link.

You are given the following GitHub repository link:
https://github.com/savram674/Smart-Assessment-Hub.git
1. Download the given GitHub repository. There are few errors in POM.XML correct them and build the project. 10M
2. The project requires Servlet API dependency. Add the proper Maven dependency for

javax.servlet-api:4.0.1. – 5M
3. What happens if the <version> tag is omitted in a dependency? – 2M
4. Add a dependency for MySQL Connector/J 8.0.33 to enable database connectivity. – 5M
5. Package the project as a WAR file and explain how to deploy it on Tomcat. – 3M
6. Where do you find the added dependency of Servlet in the maven local repository – 2M
7. Draw the folder structure of your project. Where you find packaged file? – 3M

Great! Let's go **step by step** through this Maven Web Application question. I'll explain everything in **simple terms** and **detail what each part does**.

QII. Maven Web Application Development (30M)

---

1. Download GitHub repo, fix POM.XML, and build the project (10M)

Steps:

1. Clone the repo:

```
git clone https://github.com/savram674/Smart-Assessment-Hub.git
```

2. Open the project in an IDE like **IntelliJ** or **Eclipse**.
3. Locate `pom.xml` file. Errors usually happen because:
    - Wrong dependency syntax
    - Missing `<version>`
    - Incorrect `<groupId>` or `<artifactId>`
4. Fix the errors in `pom.xml`. Example syntax of a dependency:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

5. Build the project using Maven:

```
mvn clean install
```

Explanation:

- `mvn clean install` removes old compiled files and builds the project again.
- POM.XML (Project Object Model) is **the configuration file** for Maven. It tells Maven: which dependencies to include, build settings, plugins, etc.

---

2. Add Servlet API dependency (5M)

Dependency for `javax.servlet-api:4.0.1`:
Add this to `<dependencies>` section of `pom.xml`:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

Explanation:

- `groupId` → organization providing the library
- `artifactId` → library name
- `version` → version of the library

- **`` `scope` ``** → **`` `provided` ``** means the servlet API will be **provided by the server (Tomcat)** and not packaged in WAR.

---

3. What happens if `` `<version>` `` is omitted in a dependency? (2M)

- If you **omit** **`` `<version>` ``**, Maven will try to use a **default version**, but usually it causes a **build failure** unless your **parent POM or dependency management** defines it.
- Best practice: **always specify** **`` `<version>` ``** to avoid conflicts.

---

4. Add MySQL Connector/J 8.0.33 (5M)

Add this dependency to connect your project to MySQL:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
</dependency>
```

Explanation:

- This library enables **Java to connect to MySQL database**.
- Maven will automatically download it from **Maven Central** and add it to your project.

---

5. Package as WAR file & deploy on Tomcat (3M)

Step 1: Package as WAR

```
mvn clean package
```

- After this, **`` `target/Smart-Assessment-Hub.war` ``** file is created.

Step 2: Deploy WAR on Tomcat

1. Copy WAR file to Tomcat **`` `webapps` ``** folder:

   ```
   <TOMCAT_HOME>/webapps/
   ```

2. Start Tomcat using **`` `bin/startup.sh` ``** (Linux) or **`` `bin/startup.bat` ``** (Windows).
3. Access your application at:

   ```
   http://localhost:8080/Smart-Assessment-Hub
   ```

Explanation:

- WAR (Web Archive) is a **packaged web application** containing JSPs, Servlets, libraries, and resources.
- Tomcat automatically extracts WAR and deploys it as a web application.

---

6. Where is Servlet dependency in Maven local repository? (2M)

- Maven stores downloaded dependencies in the **local repository**:

```
C:\Users\<username>\.m2\repository\javax\servlet\javax.servlet-api\4.0.1\
```

- You will find the **jar file** here (**`` `javax.servlet-api-4.0.1.jar` ``**).

Explanation:

- Maven downloads dependencies from remote repo (Maven Central) and caches them locally to avoid re-downloading.

---

7. Folder structure & packaged file (3M)

Typical Maven Web App folder structure:

```
Smart-Assessment-Hub/
│
├── src/
│   ├── main/
│   │   ├── java/           # Java source files (Servlets, etc.)
│   │   ├── resources/      # Config files, properties
│   │   └── webapp/         # JSP, HTML, CSS
│   │       └── WEB-INF/    # web.xml and lib folder
│   └── test/
│       └── java/           # Test classes
│
├── pom.xml                 # Maven config file
└── target/
    └── Smart-Assessment-Hub.war  # Packaged WAR file
```

Explanation:

- `src/main/java` → source code
- `src/main/webapp` → web resources
- `pom.xml` → Maven build config
- `target/` → output folder where WAR file is created

If you want, I can **draw a neat diagram of the folder structure** and **highlight where the WAR file is located** for your notes.

Do you want me to do that?

QIII. Git & GitHub Integration with Maven Project (30M)

Using the Maven project created above, perform the following Git tasks:
1. Introduce yourself to git - Configure -- 2 M
2. Initialize the Smart Assessment Hub project in Git. – 2M
3. Commit with message “Added automated grading module”. – 2M
4. Check the status of modified and staged files. – 2M
5. View the commit history with author details. – 2M
6. Create a new branch feedback-module. – 2M
7. Merge feedback-module into main without conflicts [resolve the intentionally created conflict]. – 3M
8. Resolve a merge conflict when report-module branch conflicts with main. – 3M
9. Create a patch file for bug fixing. Write step to Apply a .patch file. – 3M
10. Push the branch analytics-dashboard to GitHub. – 2M
11. Use a fork-and-pull request workflow to contribute a new feature for the git repo https://github.com/octocat/Spoon-Knife . – 5M
12. Check the differences between your local branch and remote main branch. – 2M

Perfect! Let’s go **step by step** and explain each Git task for your **Maven project**, in simple words and with commands.

QIII. Git & GitHub Integration with Maven Project (30M)

---

1. Introduce yourself to Git – Configure (2M)

What it does:
Git needs your **name and email** to record who makes commits.

Commands:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

Explanation:

- `--global` → sets config for all repos on your system

- These details appear in commit history.

---

2. Initialize the project in Git (2M)

What it does:

Creates a `` `.git` `` folder to track changes in your project.

Command:

```
cd Smart-Assessment-Hub
git init
```

Explanation:

- Converts project into a Git repository
- You can now track changes, commit, and push to GitHub.

---

3. Commit with message “Added automated grading module” (2M)

Steps:

1. Add files to staging area:

```
git add .
```

2. Commit changes:

```
git commit -m "Added automated grading module"
```

Explanation:

- `` `git add .` `` → stages all changes
- `` `git commit -m` `` → saves snapshot with a message

---

4. Check status of modified and staged files (2M)

Command:

```
git status
```

Explanation:

- Shows **modified, staged, and untracked files**
- Helps you know what is ready to commit

---

5. View commit history with author details (2M)

Command:

```
git log --pretty=format:"%h - %an, %ar : %s"
```

Explanation:

- `` `%h` `` → short commit hash
- `` `%an` `` → author name
- `` `%ar` `` → relative date
- `` `%s` `` → commit message

---

6. Create a new branch feedback-module (2M)

Command:

```
git checkout -b feedback-module
```

Explanation:

- Creates **new branch** and switches to it
- Branch allows **working independently** without affecting main

---

7. Merge feedback-module into main without conflicts (3M)

Steps:

1. Switch to main:

```
git checkout main
```

2. Merge branch:

```
git merge feedback-module
```

3. If there are **conflicts**, edit the files to resolve and then:

```
git add <resolved-file>
git commit
```

Explanation:

- Merge combines changes from another branch
- Conflicts occur when same lines are changed in both branches

---

8. Resolve merge conflict with report-module (3M)

Steps:

1. Merge branch:

```
git checkout main
git merge report-module
```

2. Git will mark conflicts:

```
<<<<<<< HEAD
main version
=======
report-module version
>>>>>>> report-module
```

3. Edit file to **choose correct content**
4. Stage and commit:

```
git add <file>
git commit -m "Resolved merge conflict with report-module"
```

Explanation:

- Conflicts must be **manually resolved**
- Git won’t allow merge until resolved

---

9. Create a patch file for bug fixing & apply it (3M)

Create patch:

```
git diff > bugfix.patch
```

Apply patch:

```
git apply bugfix.patch
```

Explanation:

- Patch file contains **differences between files**

- Can be sent to others or applied on another branch

---

10. Push branch analytics-dashboard to GitHub (2M)

Command:

```
git checkout analytics-dashboard
git push origin analytics-dashboard
```

Explanation:

- Uploads local branch to **remote GitHub repository**

---

11. Fork-and-pull request workflow (5M)

Steps to contribute to another repo:

1. **Fork** the repo `https://github.com/octocat/Spoon-Knife` on GitHub (creates your copy).
2. **Clone your fork locally:**

```
git clone https://github.com/<your-username>/Spoon-Knife.git
```

3. **Create a new branch** for your feature:

```
git checkout -b new-feature
```

4. **Make changes**, commit, and push:

```
git add .
git commit -m "Added new feature"
git push origin new-feature
```

5. **Create Pull Request** on GitHub to merge your branch into original repo

Explanation:

- Fork → copy repo
- Pull request → request to merge your changes into original

---

12. Check differences between local branch and remote main branch (2M)

Command:

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

Explanation:

- `git fetch` → updates info from remote
- `git diff` → shows changes between your local branch and remote main

---

If you want, I can **draw a workflow diagram** showing **branches, merge, conflict resolution, and pull request steps** which makes it very easy to remember for exams.

Do you want me to do that?

QIV. Docker containerization for Maven Web Application - 20M
Task: Containerize your Maven project using Docker.
1. Check Docker version and list all images. – 2M
2. Pull and run the hello-world image. – 2M
3. Run python image in an interactive mode. – 2M
4. Write a Dockerfile to build and run the Smart Assessment Hub application using Tomcat base image.
3M
5. Build an image named smarthub-image and run it as a container. – 2M

6. List all containers (running and stopped). – 2M
7. Start and stop a running container. – 2M
8. Push your custom image to Docker Hub. – 3M
9. Inspect the logs of a Docker container to troubleshoot errors. – 2M

Sure! Let's go **step by step** for Docker containerization of your Maven Web Application. I'll explain each command and its purpose in **simple terms**.

QIV. Docker Containerization for Maven Web Application – 20M

---

1. Check Docker version & list all images (2M)

Commands:

```
docker --version
docker image ls
```

Explanation:

- `docker --version` → shows installed Docker version
- `docker image ls` → lists all Docker images on your system

---

2. Pull and run the hello-world image (2M)

Commands:

```
docker pull hello-world
docker run hello-world
```

Explanation:

- `docker pull hello-world` → downloads test image from Docker Hub
- `docker run hello-world` → runs a container from that image
- This verifies Docker installation is working correctly

---

3. Run Python image in interactive mode (2M)

Command:

```
docker run -it python:3.12
```

Explanation:

- `-it` → interactive terminal mode
- Opens a **Python shell inside the container**
- You can run Python commands without installing Python on your host

---

4. Write a Dockerfile for Smart Assessment Hub (3M)

Dockerfile Example:

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

# Remove default webapps
RUN rm -rf /usr/local/tomcat/webapps/*

# Copy WAR file to Tomcat webapps folder
```

🚀 Powered by ChatGPT Exporter

```
COPY target/Smart-Assessment-Hub.war /usr/local/tomcat/webapps/ROOT.war

# Expose default Tomcat port
EXPOSE 8080

# Start Tomcat server
CMD ["catalina.sh", "run"]
```

Explanation:

- **`FROM`** → base image (Tomcat)
- **`COPY`** → copy your WAR file into the container
- **`EXPOSE`** → tells Docker which port the app will use
- **`CMD`** → command to run when container starts

---

5. Build image & run container (2M)

Commands:

```
docker build -t smarthub-image .
docker run -d -p 8080:8080 smarthub-image
```

Explanation:

- **`docker build -t smarthub-image .`** → builds an image with name **`smarthub-image`**
- **`docker run -d -p 8080:8080`** → runs container in **detached mode** and maps port 8080 to host

---

6. List all containers (2M)

Command:

```
docker ps -a
```

Explanation:

- Shows all **running and stopped containers**
- **`docker ps`** → running only
- **`docker ps -a`** → running + stopped

---

7. Start and stop a running container (2M)

Commands:

```
docker stop <container_id>
docker start <container_id>
```

Explanation:

- **`stop`** → stops a container
- **`start`** → restarts it without rebuilding

---

8. Push custom image to Docker Hub (3M)

Steps:

1. Log in to Docker Hub:

```
docker login
```

2. Tag image with your Docker Hub username:

```
docker tag smarthub-image <dockerhub-username>/smarthub-image:latest
```

3. Push to Docker Hub:

```
docker push <dockerhub-username>/smarthub-image:latest
```

Explanation:

- Uploads your custom image to Docker Hub so it can be shared

---

9. Inspect logs of a container (2M)

Command:

```
docker logs <container_id>
```

Explanation:

- Shows container output
- Useful for **troubleshooting errors** or checking if app started properly

---

If you want, I can **draw a neat diagram showing the full Docker workflow** for this Maven web app: from Dockerfile → build → run → push → logs. This is very helpful for exams.

Do you want me to do that?

QV. DOCKER COMPOSE -10M
Your have successfully built Smart Assessment Hub using Maven and pushed the application image to Docker Hub. The next step is to run the application along with a Tomcat and run the containers. Also include a database for future use (MYSQL)
1. create a multi folder docker-compose.yml for your Smart Assessment Hub Project with Tomcat Server and include a database mysql to be used at port 3306. Add volumes for persistency. Show both the containers running using docker compose up command
-- 5M
2. Container 1 – Access Tomcat at port 8080 -- 2M
3. Container 2 – Execute your database to add a new table -- 3M

Perfect! Let's go **step by step** for **Docker Compose** setup for your Smart Assessment Hub (SAH) with Tomcat and MySQL. I'll explain clearly and in simple terms.

QV. Docker Compose – 10M

---

1. Create docker-compose.yml with Tomcat and MySQL (5M)

docker-compose.yml Example:

```
version: "3.9"

services:
  tomcat-app:
    image: <your-dockerhub-username>/smarthub-image:latest
    container_name: tomcat-container
    ports:
      - "8080:8080"
    depends_on:
      - db
    restart: always

  db:
    image: mysql:8.0
    container_name: mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: root123
      MYSQL_DATABASE: sah_db
      MYSQL_USER: sah_user
      MYSQL_PASSWORD: sah_pass
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
    restart: always

volumes:
  mysql-data:
```

Explanation:

- `tomcat-app`: Runs **Smart Assessment Hub** on Tomcat at port 8080
- `depends_on`: Ensures Tomcat starts **after MySQL**
- `db`: MySQL container, port 3306
- `volumes`: `mysql-data` persists database data even if container is removed

Run the containers:

```
docker compose up -d
```

Check running containers:

```
docker ps
```

- You should see **Tomcat and MySQL containers running**

---

2. Access Tomcat at port 8080 (2M)

- Open browser:

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

- You will see the **Smart Assessment Hub web application** running in Tomcat

Explanation:

- `ports: "8080:8080"` maps **host port 8080** to **container port 8080**

---

3. Access MySQL container and create a table (3M)

Steps:

1. Enter MySQL container:

```
docker exec -it mysql-container mysql -u sah_user -p
```

2. Enter password: `sah_pass`
3. Switch to your database:

```
USE sah_db;
```

4. Create a new table:

```
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50)
);
```

5. Verify table:

```
SHOW TABLES;
```

Explanation:

- `docker exec -it` → open **interactive shell inside container**

- `` `mysql -u <user> -p` `` → login to MySQL
- Volumes ensure **data persists** even if MySQL container restarts

---

✅ Summary:

Step	Action	Command/URL
1	Start Tomcat & MySQL via Docker Compose	`` `docker compose up -d` ``
2	Access web app	`` `http://localhost:8080` ``
3	Add table to database	`` `docker exec -it mysql-container mysql -u sah_user -p` ``

I can also **draw a diagram showing both containers, ports, volumes, and flow** for easy understanding and exam reference.

Do you want me to do that?