**Software Engineering Lab Internal - Answer Key (Sets 1, 2 & 3)**

**Set 1: Campus Event Management System**

**QI. Software Requirement Specification (SRS)**

**a. Abstract**

The **Campus Event Management System** is a centralized web-based platform designed to streamline the planning, promotion, and execution of events within a college campus. The system will provide an integrated solution for event organizers to create and manage events, for students and faculty to register and receive notifications, and for administrators to monitor and manage all campus activities. The primary goal is to enhance communication, improve event attendance, and increase overall efficiency by automating registration, scheduling, and attendance tracking.

**b. Functional Requirements**

1. **User Authentication:** Users (Students, Admins) must be able to register and log in to the system securely.
2. **Event Creation:** Administrators and designated event organizers must be able to create new events with details like title, description, date, time, and location.
3. **Event Registration:** Students and faculty must be able to browse a list of upcoming events and register for them.
4. **Notifications:** The system shall send automated email or in-app notifications to registered users about upcoming events and any changes to the schedule.
5. **Attendance Tracking:** Event organizers must be able to mark the attendance of participants for each event.

**c. Non-Functional Requirements**

1. **Performance:** The system should load any page within 3 seconds and handle at least 100 concurrent user sessions without a noticeable drop in performance.
2. **Security:** All user data, especially passwords, must be encrypted both in transit and at rest. The system should be protected against common web vulnerabilities like SQL injection and XSS.
3. **Usability:** The user interface must be intuitive and easy to navigate, requiring minimal training for a new user to register for an event.
4. **Availability:** The system must be available 99.5% of the time, excluding planned maintenance windows.

**d. Identification of Users**

1. **Students:** The primary users who will browse events, register for them, and view their schedule.
2. **Event Organizers / Faculty:** Users who have permissions to create, update, and manage specific events. They can also view registrant lists and track attendance.
3. **System Administrator:** A privileged user responsible for managing user accounts, overseeing all events, and maintaining the system's overall health.

**QII. Maven Java Application Development**

1. Download the repository and list files:

First, you clone the repository from GitHub. Then, you can list the files.

Bash

git clone https://github.com/kumbhambhargavi75/CampusMgmtSystem/

cd CampusMgmtSystem

ls -R

1. Resolve "Source option 1.7 is no longer supported" error:

This error occurs because the version of Java you are using to run Maven is newer than the version specified in the pom.xml. To fix it, you need to update the Java version property in your pom.xml.

Solution: Change the compiler source and target properties to a supported version like 1.8 (Java 8) or 11.

XML

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

1. Upgrade JUnit from 3.8.1:

To use modern Java features and testing annotations (@Test), you should upgrade JUnit.

Solution: Update the JUnit dependency in your pom.xml to a more recent version, like 4.13.2.

XML

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

1. What happens if the <version> tag is missing?

If the <version> tag is omitted from a dependency, Maven cannot determine which version of the library to download. This will cause the build to fail with an error like "'dependencies.dependency.version' for [dependency] is missing." Maven needs an explicit version to ensure a deterministic and reproducible build.

1. Misspelled <artifactId>cms</artifactId> in a dependency:

If you misspell an <artifactId>, Maven will not be able to find the specified library in the remote repositories (like Maven Central). This will result in a build failure with an error message indicating that the dependency could not be resolved or found.

To ensure a proper version is used, you must first correct the <artifactId> to its proper value and then specify the desired version explicitly in the <version> tag.

1. Error from <paking> tag:

The tag <paking> is a typo. The correct tag name is <packaging>.

Error Thrown: Maven will fail during the validation phase of the build lifecycle, throwing an error like "Unrecognised tag: 'paking'" or "Project descriptor is invalid."

Debugging: To debug such XML mistakes, you can carefully proofread the pom.xml file, use an IDE with Maven support, or run mvn validate.

1. Impact of default packaging (jar):

If you omit the <packaging> tag, Maven defaults to jar. The impact is that mvn package will produce a .jar file instead of a .war file. A .jar file cannot be deployed on a web server like Tomcat.

1. **Add MySQL dependency:**

XML

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

1. How can I add the JQuery dependency?

You can manage it using Maven via WebJars.

XML

<dependency>

<groupId>org.webjars</groupId>

<artifactId>jquery</artifactId>

<version>3.6.4</version>

</dependency>

1. Correcting <artifactId>:

An incorrect <artifactId> will cause the generated JAR/WAR file to have the wrong name. This can cause issues in deployment scripts that expect a specific filename. To fix it, you simply correct the spelling inside the <artifactId>...</artifactId> tags in the pom.xml.

1. & 12. Change build output directory:

To change the default build output directory from target/ to build\_output/, you add a <directory> tag inside the <build> section of your pom.xml.

XML

<build>

<finalName>CampusMgmtSystem</finalName>

<directory>${project.basedir}/build\_output</directory>

</build>

1. **Add JUnit 4.13-beta-2 dependency:**

XML

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13-beta-2</version>

<scope>test</scope>

</dependency>

1. **Save, Run, and Push updated pom.xml:**

Bash

mvn clean install

git add pom.xml

git commit -m "feat: Update pom.xml with correct dependencies and settings"

git push origin main

**QIII. Git & GitHub Integration with Maven Project**

1. **Start tracking the project:**

Bash

git init

git add .

git commit -m "Initial commit"

1. **Create a snapshot with a message:**

Bash

git add .

git commit -m "Added event registration feature"

1. **View status of modified/staged files:** git status
2. **View commit history:** git log
3. **See all branches (local and remote):** git branch -a
4. **Create a new branch event-scheduler:** git branch event-scheduler
5. **Apply a patch file:** git apply name-of-the-patch-file.patch
6. **Create, commit on, and merge a branch:**

Bash

git checkout -b feedback

# ... make changes and commits ...

git checkout main

git merge feedback

1. Resolve a merge conflict:

When git merge reports a conflict:

* 1. Open the conflicted file(s) and look for conflict markers (<<<<<<<, =======, >>>>>>>).
  2. Edit the file to remove the markers and keep the correct code.
  3. Stage the resolved file: git add <conflicted-file-name>.
  4. Commit the merge: git commit.

1. **Collaborate using Fork-and-Pull-Request workflow:**
   1. **Fork:** Click the "Fork" button on the original GitHub repository.
   2. **Clone:** Clone your forked repository to your local machine.
   3. **Branch:** Create a new branch for your feature: git checkout -b my-new-feature.
   4. **Commit & Push:** Make changes, commit them, and push the branch to your fork: git push origin my-new-feature.
   5. **Pull Request (PR):** Go to your fork on GitHub and create a Pull Request to the original repository.
2. **See changes between local and remote main branch:**

Bash

git fetch origin

git diff origin/main main

1. **Test a teammate's new branch locally:**

Bash

git fetch origin

git checkout event-feedback

1. **See differences from the last commit:** git show

**QIV. Docker containerization for Maven JAVA Application**

**Dockerfile for the Maven Project**

Create a file named Dockerfile in the root of your project:

Dockerfile

# Stage 1: Build the application using Maven

FROM maven:3.8.5-openjdk-11 AS build

WORKDIR /app

COPY pom.xml .

COPY src ./src

RUN mvn clean package -DskipTests

# Stage 2: Create the final production image using Tomcat

FROM tomcat:9.0-jdk11-openjdk-slim

RUN rm -rf /usr/local/tomcat/webapps/\*

COPY --from=build /app/target/CampusMgmtSystem.war /usr/local/tomcat/webapps/ROOT.war

EXPOSE 8080

CMD ["catalina.sh", "run"]

**Docker Commands**

* **Check Docker version and list images:**

Bash

docker --version

docker images

1. **Pull and run hello-world:**

Bash

docker pull hello-world

docker run hello-world

1. **Run ubuntu in interactive mode:** docker run -it ubuntu bash
2. **Build the csmage image:** docker build -t csmage:latest .
3. **Pull an official image, run it, and list running containers:**

Bash

docker pull nginx

docker run --name my-nginx-server -d -p 8080:80 nginx

docker ps

1. **Start and stop a running container:**

Bash

docker stop my-nginx-server

docker start my-nginx-server

1. **Check a list of all containers (including stopped ones):** docker ps -a
2. **Stop a running container:** docker stop <container\_name\_or\_id>
3. **Push your custom image to Docker Hub:**

Bash

docker login

docker tag csmage:latest your-dockerhub-username/csmage:latest

docker push your-dockerhub-username/csmage:latest

1. Check if a container exited cleanly or crashed:

You can inspect the exit code of a stopped container. 0 means a clean exit; any non-zero code indicates an error.

Bash

docker inspect <container\_id> | grep ExitCode

**QV. DOCKER COMPOSE**

Create a file named docker-compose.yml:

YAML

version: '3.8'

services:

# Container 1: A plain Tomcat server

tomcatservice:

image: tomcat:9.0-jdk11-openjdk-slim

container\_name: plain-tomcat

ports:

- "8086:8080"

# Container 2: Your Campus Management System application

campusapp:

image: your-dockerhub-username/csmage:latest

container\_name: campus-management-app

ports:

- "7007:8080"

To run this, navigate to the directory and run docker-compose up -d.

**Set 2: Food Ordering System**

**QI. Software Requirement Specification (SRS)**

**a. Abstract**

The **Cafeteria Food Ordering System** is a modern, web-based application designed to digitize and streamline the food ordering process for students and faculty on campus. The system provides an intuitive interface for browsing the cafeteria menu, placing orders, and making secure digital payments. It offers real-time order status updates for customers and a comprehensive dashboard for administrators to manage menu items, pricing, and orders. The primary objective is to enhance user convenience, reduce wait times, and improve the operational efficiency of the cafeteria.

**b. Functional Requirements**

1. **User Authentication:** Users (Students, Staff, Admin) must be able to create an account, log in, and log out.
2. **Menu Management:** Administrators must be able to add, update, delete, and categorize food items on the menu.
3. **Order Placement:** Customers must be able to browse the menu, add items to a cart, and place an order.
4. **Payment Gateway:** The system must integrate with a digital payment gateway to process online payments.
5. **Order Tracking:** Customers must be able to view the real-time status of their order.

**c. Non-Functional Requirements**

1. **Performance:** The system must load the menu and process an order within 3 seconds.
2. **Security:** All payment transactions must be encrypted using SSL/TLS.
3. **Usability:** The interface should be simple and self-explanatory.
4. **Reliability:** The system should have an uptime of 99.8%.

**d. Identification of Users**

1. **Customer (Student/Faculty):** The primary user who browses the menu and places orders.
2. **Administrator (Cafeteria Staff):** A privileged user who manages the menu and views incoming orders.

**QII. Maven Web Application Development**

1. **Artifact name with <finalName>Food-System</finalName>:**
   * **Generated Artifact:** Food-System.war.
   * **Deployment Issues:** Tomcat will use the filename as the application's **context path**, so the URL would be http://localhost:8080/Food-System/. This can cause "404 Not Found" errors if developers expect a different URL.
2. **Download the repository and list files:**

Bash

git clone https://github.com/kumbhambhargavi75/FoodSystem/

cd FoodSystem

ls -R

1. **JUnit version is left out:**
   * **What Maven will do:** The **build will fail** because Maven requires an explicit version for every dependency.
   * **How to ensure a proper version:** You must explicitly add the <version> tag inside the <dependency> block.
2. **Misspelled <artifactId>:**
   * **Error:** The **build will fail** with a "Could not resolve dependencies" error because Maven can't find the library.
   * **Debugging:** Check the pom.xml for typos or run mvn dependency:tree.
3. Impact of omitting <packaging> tag:

Maven will default to jar packaging, creating a .jar file instead of a .war. This .jar file cannot be deployed on a Tomcat server.

1. Correcting the MySQL dependency:

The provided XML snippet has misspelled tags: <groupdId> should be <groupId> and <artifactd> should be <artifactId>.

* + **Correct Coordinates:**

XML

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.28</version>

</dependency>

1. Missing information in tomcat7-maven-plugin:

The plugin configuration is likely missing a specific <version> for the plugin itself and a <configuration> block with a defined <path> for the application's context URL.

1. **The <url> tag:**
   * **Mistake:** The snippet url>...</url> is missing the opening angle bracket (<).
   * **Purpose:** The <url> element in a pom.xml is **purely informational**. It provides a link to the project's website and has **no effect on the build process**.
2. Removing <contextPath> from the Tomcat plugin:

Without <contextPath>, the application's context path will default to the project's <artifactId>, which might not be the desired URL.

1. **Deployment URL .../FoodSystem-0.0.1-SNAPSHOT:**
   * **Output:** The homepage of your web application.
   * **Why:** Tomcat uses the name of the WAR file (<artifactId>-<version>.war) as the default context path.
2. **Effect of <finalName>FoodOrder</finalName>:**
   * **WAR Name:** FoodOrder.war.
   * **Deployment URL:** http://localhost:8080/FoodOrder/.
3. **Meaning of SNAPSHOT version:**
   * **Meaning:** A SNAPSHOT version indicates an unstable version currently in **active development**.
   * **Impact:** Maven will always check for the latest SNAPSHOT build from the remote repository, which can lead to non-reproducible builds.
4. **Applying a patch file with Git:**
   * **Git Command:** git apply path/to/the/css-fix.patch
   * **Ensuring it becomes part of commits:** After applying and testing, you must stage and commit the changes:

Bash

git add .

git commit -m "fix: Apply CSS fix from teammate"

**QIII. Git & GitHub Integration**

1. **Initialize a Git repository:** git init
2. **Set global user name and email:**

Bash

git config --global user.name "Your Name"

git config --global user.email "youremail@example.com"

1. **Stage, commit, and connect to GitHub:**

Bash

git add .

git commit -m "Initial commit"

git remote add origin <your-github-repository-url>

git push -u origin main

1. **Stage a specific file and a folder:** git add pom.xml src/main/java
2. **git reset vs. git rm --cached:**
   * **git reset HEAD <file>:** **Unstages** a file, leaving it in your working directory as a modified file.
   * **git rm --cached <file>:** **Unstages** the file AND tells Git to **stop tracking** it completely.
3. **Save changes without committing:** git stash
4. **Remove a wrongly staged file (temp.txt) from staging:** git reset HEAD temp.txt
5. **Stash changes with a descriptive message:** git stash save "WIP: Implementing user login"
6. **Merge a feature branch into main:**

Bash

git checkout main

git pull origin main

git merge feature/reviews

1. **Copy a project to a teammate's local machine:** git clone <repository\_url>
2. **Push a new local branch to GitHub:** git push -u origin feature/real-time-status
3. **Connect to GitHub using SSH:**
   1. **Generate an SSH key pair:** ssh-keygen -t ed25519 -C "your\_email@example.com"
   2. **Add the public key to GitHub:** Copy the contents of your public key file (e.g., ~/.ssh/id\_ed25519.pub) and add it to your GitHub account settings.

**QIV. Docker containerization**

1. **Pull nginx:latest from Docker Hub:** docker pull nginx:latest
2. **Run nginx in a detached container with port mapping:** docker run -d --name web-nginx -p 8090:80 nginx:latest
3. **Stop a container consuming high CPU:** docker stop <container\_name\_or\_id>
4. **Stop and remove a running container:** docker rm -f <container\_name\_or\_id>
5. **Run a Redis container with port mapping:** docker run -d -p 8080:6379 redis
6. **List running containers:** docker ps
7. **Pull and run python and list running containers:**

Bash

docker pull python

docker run -dit python

docker ps

1. **Handle a port conflict on 8080:** The error "port is already allocated" means another process is using it. Solve it by stopping the other process or running your container on a different host port (e.g., -p 8081:8080).
2. **Update configuration and rebuild/rerun:**
   1. Stop and remove the old container: docker rm -f <old\_container\_name>.
   2. Update your source code.
   3. Rebuild the Docker image: docker build -t your-app:v2 .
   4. Rerun the container with the new image and port: docker run -d -p 8090:8080 your-app:v2.
3. **Stop and start a running container:** docker stop <container> and docker start <container>
4. **Check container status:** docker ps -a (to see status), docker logs <container> (to see application output).

**QV. DOCKER COMPOSE**

Create a docker-compose.yml file to run the Food Ordering System with a PostgreSQL database.

YAML

version: '3.8'

services:

database:

image: postgres:13

container\_name: food-db

environment:

POSTGRES\_DB: food\_ordering\_db

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

volumes:

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

app:

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

container\_name: food-ordering-app

ports:

- "7078:8080"

environment:

SPRING\_DATASOURCE\_URL: jdbc:postgresql://database:5432/food\_ordering\_db

SPRING\_DATASOURCE\_USERNAME: user

SPRING\_DATASOURCE\_PASSWORD: password

depends\_on:

- database

volumes:

postgres\_data:

**Set 3: Hospital Management System**

**QI. Software Requirement Specification (SRS)**

**a. Abstract**

The **Hospital Management System (HMS)** is a comprehensive, integrated information system designed to manage all aspects of a hospital's operations. This system aims to replace manual processes with a centralized, secure, and efficient digital solution. It will handle patient registration, appointment scheduling, electronic medical records (EMR), and billing. The primary goal is to improve the quality of patient care, reduce operational costs, and ensure compliance with healthcare regulations.

**b. Functional Requirements**

1. **Patient Registration:** Staff must be able to register new patients and create a unique patient ID.
2. **Appointment Scheduling:** Receptionists and patients must be able to book, view, and cancel appointments.
3. **Electronic Medical Records (EMR):** Doctors and nurses must be able to view and update a patient's medical history, diagnoses, and prescriptions.
4. **Billing and Invoicing:** The system must automatically generate bills for consultations and treatments.
5. **User Role Management:** An administrator must be able to create and manage user accounts with different roles (e.g., Doctor, Nurse, Admin).

**c. Non-Functional Requirements**

1. **Security:** The system must be HIPAA compliant, with strong encryption for all patient data.
2. **Performance:** The system must retrieve any patient record in under 2 seconds.
3. **Availability:** The system must be available 24/7 with a 99.9% uptime.
4. **Scalability:** The system must handle a 50% increase in patient data over two years.

**d. Identification of Users**

1. **Administrator:** Manages the entire system and user accounts.
2. **Doctor:** Accesses patient records and prescribes medication.
3. **Nurse:** Updates patient vitals and care notes.
4. **Receptionist:** Manages patient registration and appointments.
5. **Patient:** Accesses a personal portal to view history and book appointments.

**QII. Maven Java Application Development**

1. **Download repository and list files:**

Bash

git clone https://github.com/kumbhambhargavi75/HospitalMgmtSystem/

cd HospitalMgmtSystem

ls -R

1. **Phases of mvn clean install:**
   * **clean lifecycle:** Runs phases up to clean (deletes the target directory).
   * **default lifecycle:** Runs phases up to install (compiles, tests, packages, and installs the artifact to the local .m2 repository).
2. **Add servlet-api-2.5 and potential errors:** Adding such an old version can cause compilation errors if your code uses features from newer Servlet APIs (3.x or higher).
3. Fix compilation failure with Java 11:

Set the compiler source and target properties in your pom.xml:

XML

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

1. **Missing <version> tag in a dependency:** The build will **fail**. Maven requires an explicit version for every dependency. Solve it by adding the <version> tag.
2. Fixing the generated file name:

To change the output file name to HospitalMgmtSystem.war, add the <finalName> tag within the <build> section of your pom.xml.

XML

<build>

<finalName>HospitalMgmtSystem</finalName>

</build>

1. **Add Java.ServletAPI dependency:**

XML

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

1. **Removing the <dependencies> section:** The project will **fail to build** at the compile phase if the code imports any external libraries.
2. **Meaning of <scope>provided</scope>:** It tells Maven that the dependency is needed for compilation but **should not be bundled** into the final WAR file because it will be provided by the runtime environment (e.g., Tomcat).
3. **"Invalid target release: 11" error on a JRE:** The error means you are trying to compile code with a **Java Runtime Environment (JRE)**, which lacks a compiler. You must run Maven with a full **Java Development Kit (JDK)**.
4. **packaging is jar for a Tomcat project:** This is **incorrect**. A web application must be packaged as a **war**. Fix it by changing <packaging>jar</packaging> to <packaging>war</packaging>.
5. **The <url> tag:** The <url> element is for documentation, providing a link to the project's website. It has **no effect on the build**.
6. **Check pom.xml and push to your account:**

Bash

git add pom.xml

git commit -m "feat: Update and correct pom.xml"

git push origin main

**QIII. Git & GitHub Integration**

1. **Discard local (unstaged) changes:** git restore .
2. **View commit history to find a mistake:** git log. To fix it, use git revert <commit-hash> (safe for shared branches) or git reset --hard <commit-hash> (for local branches only).
3. **View commit history in a readable format:** git log --oneline --graph --decorate
4. **Create and switch to a new branch:** git checkout -b feature/patient
5. **Upload local commits to the remote:** git push
6. **See all local and remote branches:** git branch -a
7. **Pull latest changes and merge them:** git pull
8. **Push a new branch for the first time and set tracking:** git push -u origin <branch-name>
9. **Configure a cloned repo to trust the remote:** This is done automatically by git clone.
10. **Avoid sending large files to the remote:** Add file patterns to the .gitignore file.
11. **Update a local branch without losing stashed changes:** git pull then git stash pop.
12. **Delete a remote branch:** git push origin --delete <branch-name>
13. **Apply a .patch file:** git apply name-of-the-patch-file.patch

**QIV. Docker containerization**

**Dockerfile for Hospital Management System**

Dockerfile

# Stage 1: Build the Java application

FROM maven:3.8-openjdk-11 AS build

WORKDIR /app

COPY pom.xml .

COPY src ./src

RUN mvn clean package -DskipTests

# Stage 2: Create the final image with Tomcat

FROM tomcat:9.0-jdk11-slim

RUN rm -rf /usr/local/tomcat/webapps/\*

COPY --from=build /app/target/HospitalMgmtSystem.war /usr/local/tomcat/webapps/ROOT.war

EXPOSE 8080

CMD ["catalina.sh", "run"]

**Docker CLI Commands**

1. **Build the Docker image:** docker build -t hospital-mgmt .
2. **List all available Docker images:** docker images
3. **Run the container with port mapping:** docker run -d --name hms-app -p 9090:8080 hospital-mgmt
4. **Pull and run redis:** docker pull redis, docker run -d --name my-redis -p 8080:6379 redis, docker ps
5. **Tag and push the image to Docker Hub:**

Bash

docker tag hospital-mgmt yourusername/hospital-mgmt:latest

docker push yourusername/hospital-mgmt:latest

1. **Inspect logs to debug:** docker logs hms-app
2. **Which command to inspect logs?** docker logs <container\_name>
3. **Make the image publicly available:** Push it to a public Docker Hub repository.
4. **Open a shell inside a container:** docker exec -it hms-app /bin/bash
5. **List running containers:** docker ps
6. **Common container management commands:** docker stop, docker start, docker restart, docker rm.

**QV. DOCKER COMPOSE**

Create a docker-compose.yml file to run the Hospital Management System with a MongoDB database.

YAML

version: '3.8'

services:

mongodb:

image: mongo:latest

container\_name: hospital-db

environment:

MONGO\_INITDB\_ROOT\_USERNAME: user

MONGO\_INITDB\_ROOT\_PASSWORD: password

MONGO\_INITDB\_DATABASE: hospital\_db

volumes:

- mongo\_data:/data/db

app:

image: yourusername/hospital-mgmt:latest

container\_name: hospital-management-app

ports:

- "7079:8080"

environment:

MONGO\_URI: mongodb://user:password@mongodb:27017/hospital\_db

depends\_on:

- mongodb

volumes:

mongo\_data: