**Simplilearn**

**Project: Developing a Backend Admin for Learner’s Academy.**

DESCRIPTION

Project objective:

As a Full Stack Developer, design and develop a backend administrative portal for the Learner’s Academy. Use the GitHub repository to manage the project artifacts.

Background of the problem statement:

Learner’s Academy is a school that has an online management system. The system keeps track of its classes, subjects, students, and teachers. It has a back-office application with a single administrator login.

The administrator can:

● Set up a master list of all the subjects for all the classes  
● Set up a master list of all the teachers  
● Set up a master list of all the classes  
● Assign classes for subjects from the master list  
● Assign teachers to a class for a subject (A teacher can be assigned to different classes for different subjects)  
● Get a master list of students (Each student must be assigned to a single class)

There will be an option to view a Class Report which will show all the information about the class, such as the list of students, subjects, and teachers  
       
The goal of the company is to deliver a high-end quality product as early as possible.

The flow and features of the application:

● Plan more than two sprints to complete the application  
● Document the flow of the application and prepare a flow chart   
● List the core concepts and algorithms being used to complete this application  
● Implement the appropriate concepts, such as exceptions, collections, and sorting techniques for source code optimization and increased performance

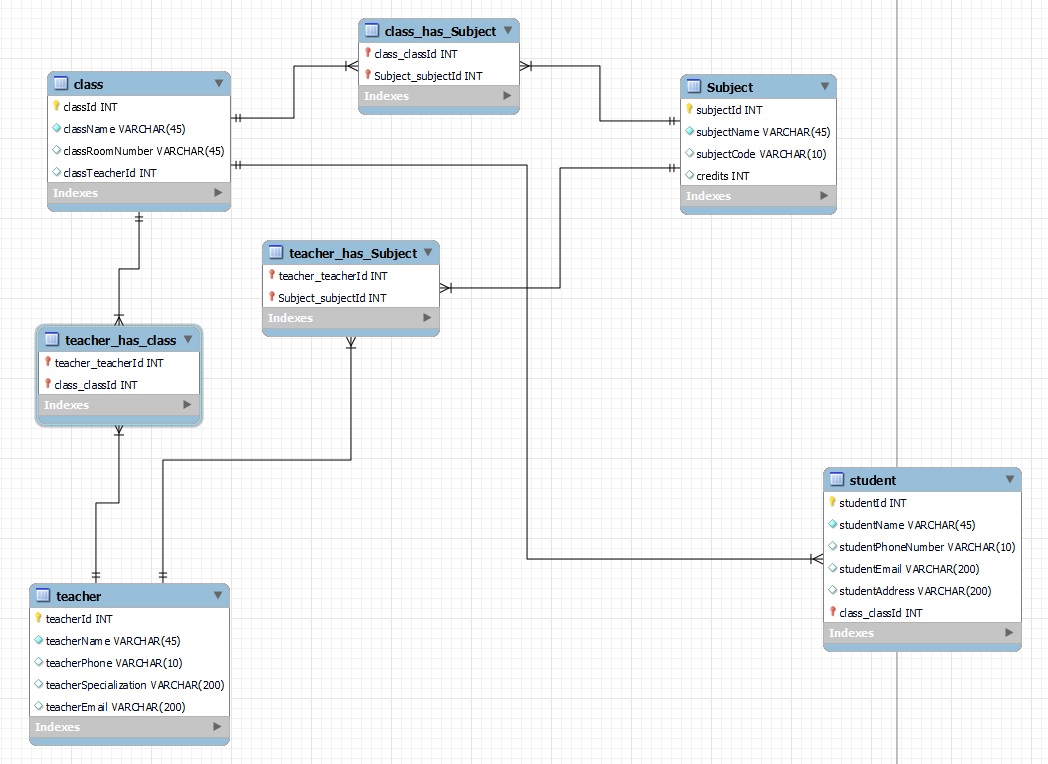
You must use the following:

● Eclipse/IntelliJ: An IDE to code for the application   
● Java: A programming language to develop the web pages, databases, and others  
● SQL: To create tables for admin, classes, students, and other specifics  
● Git: To connect and push files from the local system to GitHub   
● GitHub: To store the application code and track its versions   
● Scrum: An efficient agile framework to deliver the product incrementally   
● Search and Sort techniques: Data structures used for the project   
● Specification document: Any open-source document or Google Docs

The following requirements should be met:

● The source code should be pushed to your GitHub repository. You need to document the steps and write the algorithms in it.  
● The submission of your GitHub repository link is mandatory. In order to track your task, you need to share the link of the repository. You can add a section in your document.   
● Document the process step-by-step starting from sprint planning to the product release.   
● The application should not close, exit, or throw an exception if the user specifies an invalid input.  
● You need to submit the final specification document which will include:   
● Project and developer details   
● Sprints planned and the tasks achieved in them   
● Algorithms and flowcharts of the application   
● Core concepts used in the project   
● Links to the GitHub repository to verify the project completion

------------------------------------------------------------------------------------------------------------------------------------------------

Step 1: open MY SQL workbench: to create entity for STUDENT, TEACHER, CLASS & SUBJECT

File🡪 new model 🡪 add diagram

Teacher to Class: Many to Many  
Teacher to Subject: Many to Many  
Class to Student: One to Many  
Class to Subject: Many to Many

Step 2: Create new maven project in IDE.

Select WAR (Web ARchive) file

* A WAR file is used for packaging web applications. It contains all the necessary resources (HTML, JSP, CSS, JavaScript files) and server-side components (Servlets, Java classes) required for deploying a web application.
* WAR files are typically used in dynamic web projects that utilize Java web technologies (such as Servlets and JSP) and are deployed on web servers (such as Apache Tomcat, Jetty, or JBoss).

Step 3: update pom.xml

* The pom.xml file is an essential configuration file used in Maven projects. "POM" stands for Project Object Model. It is an XML file that resides in the root directory of the Maven project and defines the project's configuration, dependencies, build settings, plugins, project structure & project Lifecycle.
* XML (Extensible Markup Language) is a markup language used for storing and transmitting structured data. It provides a set of rules for encoding documents in a format that is both human-readable and machine-readable. XML is widely used in various domains, including web development, data interchange, configuration files, and more.

**Add maven war plugin**: from website: <https://maven.apache.org/plugins/maven-war-plugin/usage.html>

Add Maven War Plugin with a specific version (3.3.2) and customizes its behavior.

<build>: This is the root element for Maven build configuration.

<plugins>: It defines a list of plugins that are used during the build process.

<plugin>: Specifies a specific plugin and its configuration.

<groupId>: Identifies the group or organization that provides the plugin. In this case, it is org.apache.maven.plugins.

<artifactId>: Specifies the unique identifier of the plugin. Here, it is maven-war-plugin, which is used to build WAR files.

<version>: Indicates the version of the plugin to be used. In this example, it is set to 3.3.2.

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

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

<maven.compiler.source>1.8</maven.compiler.source> indicates that the source code in the project is compatible with Java version 1.8.

<maven.compiler.target>1.8</maven.compiler.target> specifies that the compiled bytecode should be compatible with Java version 1.8.

**Add Dependencies:**

Add hibernate core maven dependency: from : <https://mvnrepository.com/artifact/org.hibernate/hibernate-core> and select 5.6.14 version

<dependency>: This is the start of the dependency configuration.

<groupId>: It specifies the group or organization that provides the dependency. In this case, it is org.hibernate.

<artifactId>: Specifies the unique identifier of the dependency. Here, it is hibernate-core, which refers to the Hibernate Core library.

<version>: Indicates the desired version of the dependency. In this example, it is set to 5.6.14.Final.

Add MY SQL 8 connector for java maven dependency: form: <https://mvnrepository.com/artifact/mysql/mysql-connector-java> and select 8.0.33 version

<dependency>: This is the start of the dependency configuration.

<groupId>: It specifies the group or organization that provides the dependency. In this case, it is mysql.

<artifactId>: Specifies the unique identifier of the dependency. Here, it is mysql-connector-java, which refers to the MySQL Connector/J library.

<version>: Indicates the desired version of the dependency. In this example, it is set to 8.0.33.

Add servlet API maven dependency: from : <https://mvnrepository.com/artifact/javax.servlet/servlet-api> and select 2.5 version.

<dependency>: This is the start of the dependency configuration.

<groupId>: It specifies the group or organization that provides the dependency. In this case, it is javax.servlet.

<artifactId>: Specifies the unique identifier of the dependency. Here, it is servlet-api, which refers to the Java Servlet API.

<version>: Indicates the desired version of the dependency. In this example, it is set to 2.5.

<scope>: It specifies the scope of the dependency. The provided scope means that the Servlet API will be provided by the target runtime environment and will not be included in the final artifact. This is common when developing web applications that will be deployed on a Java Servlet container such as Apache Tomcat, which already provides the Servlet API implementation.

Add JSP API maven dependency: from: <https://mvnrepository.com/artifact/javax.servlet/jsp-api> and select 2.0 version.

<dependency>: This is the start of the dependency configuration.

<groupId>: It specifies the group or organization that provides the dependency. In this case, it is javax.servlet.

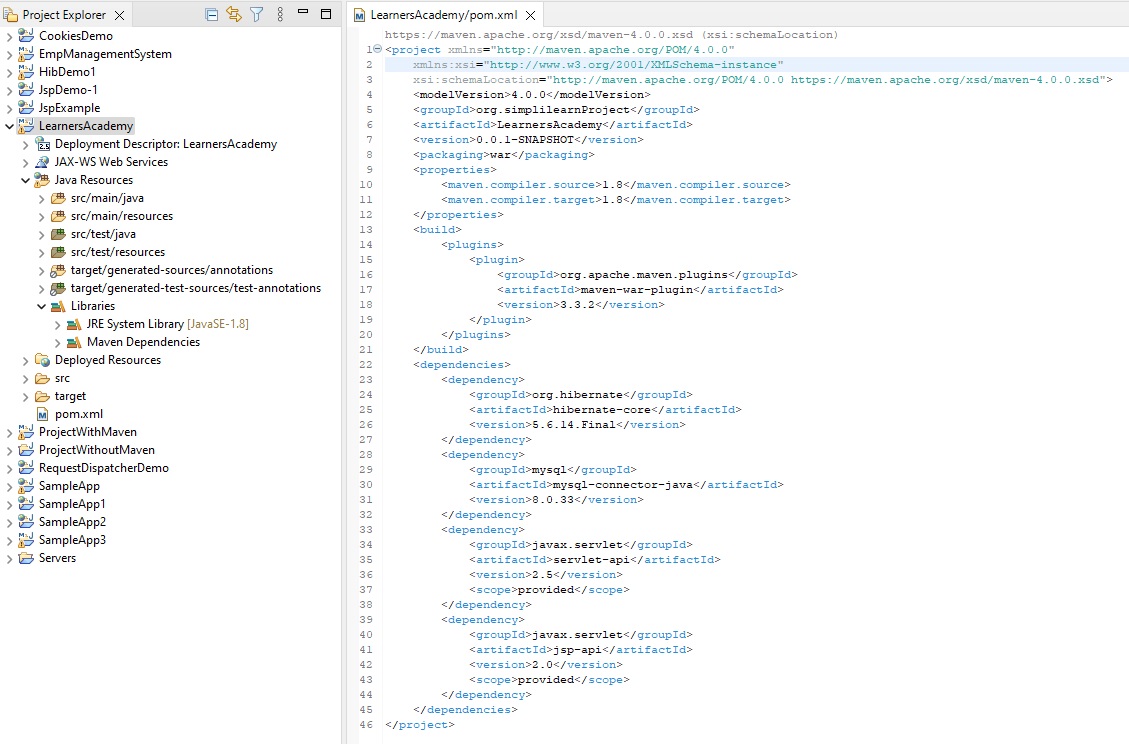
<artifactId>: Specifies the unique identifier of the dependency. Here, it is jsp-api, which refers to the JavaServer Pages (JSP) API.

<version>: Indicates the desired version of the dependency. In this example, it is set to 2.0.

<scope>: It specifies the scope of the dependency. The provided scope means that the JSP API will be provided by the target runtime environment and will not be included in the final artifact. This is common when developing web applications that will be deployed on a Java Servlet container, which already provides the JSP API implementation.

Note: Ctrl+shift+f for formatting

Update maven project



<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.simplilearnProject</groupId>

<artifactId>LearnersAcademy</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<properties>

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

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

</properties>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>3.3.2</version>

</plugin>

</plugins>

</build>

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.6.14.Final</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.33</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

</dependencies>

</project>

Step 4: Create java class for “course”.

Step 5: Create java class for “subject”.

Step 6: Create java class for “teacher”.

Step 7: Create java class for “student”.

**Annotation from javax.persistence package**

**@Entity**

The @Entity annotation is used in Java classes to indicate that the class is an entity in a relational database. It is typically used in object-relational mapping (ORM) frameworks like Hibernate, which map Java objects to database tables.

When you annotate a Java class with @Entity, you are essentially defining that the class represents a table in a database. Each instance of the class corresponds to a row in the table, and the class's fields represent the table columns.

**@Id**

When you annotate a field with @Id, you are specifying that the field represents the primary key of the entity in the database. The primary key uniquely identifies each record (row) in the corresponding database table.

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

The @GeneratedValue annotation is used in Java classes to specify the generation strategy for automatically generating values for a field annotated with @Id. It is commonly used in conjunction with the @Id annotation and is often used in ORM frameworks like Hibernate.

The @GeneratedValue annotation provides different strategies for generating unique values for primary keys. One of the commonly used strategies is GenerationType.IDENTITY.

**@ManyToMany**

The @ManyToMany annotation is typically applied to a collection field in an entity class to indicate that it is associated with a many-to-many relationship. The mappedBy attribute specifies the inverse side of the relationship, which means the field name in the associated entity that manages the relationship.

**inverseJoinColumns**

The inverseJoinColumns attribute is used in Java classes to define the mapping of the join columns in a Many-to-Many relationship using the @ManyToMany annotation. It is typically used in object-relational mapping (ORM) frameworks like Hibernate.

In a Many-to-Many relationship, two entities are associated with each other through a join table. The inverseJoinColumns attribute specifies the columns in the inverse (non-owning) side of the relationship that are mapped to the join table.

**@JoinColumn**

The @JoinColumn annotation is typically applied to a field in an entity class that represents a relationship with another entity. It allows you to customize the column name used for the join operation.

**Set**

Sets are commonly used when you need to store a collection of distinct values and perform operations like adding elements, removing elements, checking for membership, or performing set operations such as union, intersection, and difference.

In many programming languages, including Java, there is a built-in data structure called Set that represents a set. Some popular implementations of the Set interface in Java include HashSet, TreeSet, and LinkedHashSet.

1. HashSet: It stores elements using a hash table, providing constant-time performance for basic operations like add, remove, and contains. The elements are not ordered.

Set<obj\_Type> obj\_Name = new HashSet<>();

2. TreeSet: It stores elements in a sorted tree structure (usually a red-black tree). The elements are ordered based on their natural ordering or a custom comparator.

3. LinkedHashSet: It is similar to HashSet but maintains a doubly-linked list to preserve the order of insertion. The elements are ordered based on their insertion order.

**Cascade**

In object-relational mapping (ORM) frameworks like Hibernate, the cascade attribute is used to specify cascading behavior for operations performed on an entity with a relationship to other entities. Specifically, the CascadeType.ALL is an option that indicates that all operations (e.g., persist, remove, merge, refresh) should be cascaded from the parent entity to the associated entities.

When you specify cascade = CascadeType.ALL, it means that if an operation is performed on the parent entity, the same operation will be applied to the associated entities. For example, if you persist the parent entity, all associated entities will also be persisted. Similarly, if you remove the parent entity, all associated entities will be removed as well.

After creating All entity and mapping relationships between them

Step 8: create Hibernate based configuration class

Step 9 : create Dao interfaces

Step 10 : create Dao implementation for all class

Step 11: create homepage by index.jsp

Step 11: create by register.jsp

Step 12: create register controller class

Step 13: create UserModel class

Step 14: create UserService interface

Step 15: create UserServiceImpl java class

Pending: check weather same email registered or not then only do registration

Step 16: create database in mysql

Now work with login functionality so,

Step 17: create login.jsp and then create login model.java

Step 18: create logincontroller servlet

Step 19: create AddTeacher.jsp

Step 20: create TeacherController servlet

Step 21: create TeacherModel.java

Step 22: create TeacherService interface

Step 23: create TeacherServiceImpl.java class

Step 24: add student succ

Similary add all required java class and services