**PROJECT DEPLOYMENT PHASE**

**EXCEPTION HANDLING**

|  |  |
| --- | --- |
| **Date** | **03 NOVEMBER 2023** |
| **Team ID** | **NM2023TMID04681** |
| **Project name** | **BUILD AN EVENT MANAGEMENT SYSTEM** |

Exception handling in Salesforce development is crucial to ensure that your event management system can gracefully handle errors and unexpected situations. Salesforce provides features and best practices for robust exception handling. Here's how you can develop code for exception handling in your Salesforce event management system:

**1. Custom Exceptions:**

You can create custom exceptions to handle specific errors or issues that might occur within your code. Here's an example of a custom exception class in Apex:

public class EventManagementException extends Exception {

public EventManagementException(String message) {

super(message);

}

}

**2. Try-Catch Blocks:**

Use try-catch blocks to catch and handle exceptions in your Apex code. Here's an example of how to use try-catch:

try {

// Code that may generate an exception

// ...

} catch (Exception e) {

// Handle the exception

System.debug('An exception occurred: ' + e.getMessage());

throw new EventManagementException('An error occurred while processing the event.');

}

In this example, we catch any exceptions that may occur, log the exception details, and throw a custom EventManagementException with a more user-friendly error message.

**3. Logging:**

Utilize Salesforce's built-in logging mechanisms, such as System.debug(), to log error details and the stack trace when exceptions occur. You can view these logs in the Developer Console.

**4. Handling Specific Errors:**

You can catch and handle specific types of exceptions, such as DML exceptions when inserting or updating records. Here's an example of handling a DML exception:

try {

// Code that may generate a DML exception

// ...

} catch (DmlException ex) {

// Handle DML exception

for (Integer i = 0; i < ex.getNumDml(); i++) {

// Process individual DML errors

System.debug('DML Error: ' + ex.getDmlMessage(i));

}

} catch (Exception e) {

// Handle other exceptions

System.debug('An exception occurred: ' + e.getMessage());

throw new EventManagementException('An error occurred while processing the event.');

}

**5. Bulk Processing:**

When working with large datasets, consider using bulk processing and handling governor limits to avoid exceptions related to limits in Salesforce.

**6. Testing Exception Handling:**

To test your exception handling, write unit tests that intentionally cause exceptions and verify that your code handles them correctly. Use assertions in your unit tests to validate the expected behavior.

Here's a simple example of a unit test:

@isTest

public class EventManagementTest {

@isTest

static void testExceptionHandling() {

try {

// Simulate an exception-generating code

// ...

System.assert(false, 'Exception should be thrown');

} catch (EventManagementException ex) {

System.assertEquals('An error occurred while processing the event.', ex.getMessage());

}

}

}

**7. Logging and Monitoring:**

Set up monitoring and logging tools to track exceptions in your production environment, allowing you to identify and resolve issues quickly.