Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment Title: Error Storages Identify error storage by considering various Applications

Aim/Objective

To identify error storage mechanisms by analyzing various applications.

Description:

It aims to identify and analyze different error storage mechanisms used across various applications. It focuses on how errors are logged, stored, and retrieved for debugging and maintenance. The outcome will help in understanding efficient error management practices in software systems.

Prerequisites:

- Basic understanding of software development and application architecture.
- Knowledge of error handling techniques in programming languages (e.g., try-catch, exception handling).
- Familiarity with logging tools and storage systems (e.g., log files, databases, cloud logging services).

Pre-Lab: Before starting the lab activity, students should:

- Review basic error handling concepts in at least one programming language (such as Python, Java, or JavaScript).
- Explore common logging mechanisms used in applications, such as file-based logs, database logs, and cloud-based logging services (e.g., Logstash, ELK Stack, AWS CloudWatch).
- Understand the architecture of a sample application to trace how and where errors are generated and stored.
- Set up a basic development environment with tools required for testing error logging (e.g., IDE, local server, logging libraries).
- Prepare a checklist or template for analyzing error storage (e.g., location, format, retrieval method, rotation policy).

In-Lab: During the lab session, students will:

- Select two or more applications (web, desktop, or mobile) for analysis of their error handling and storage mechanisms.
- Trigger intentional errors (e.g., invalid inputs, broken API calls) to observe how the application logs and stores them.
- Identify the error storage locations such as log files, system events, or cloud logging services.
- Analyze the format of stored errors (e.g., plain text, JSON, structured logs) and note how accessible and readable they are.
- Compare logging tools/libraries used (e.g., Python's logging, Java's Log4j, or external tools like Sentry or ELK Stack).

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- Document the findings with screenshots, sample error messages, and storage paths.
- Discuss efficiency of the error storage mechanism in terms of speed, scalability, and ease of debugging.

Procedure/Program:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:

Analysis and Inferences:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Sample viva voce questions:

1. **How do you handle runtime errors in a Java-based frontend application?**

2. **What is the purpose of using a logger like java.util.logging or Log4j?**

3. **How do you ensure user-friendly error messages while storing technical details for developers?**

4. **Where are log files stored in your Java application, and how are they formatted?**

5. **What are the best practices for logging in GUI-based applications like JavaFX?**

Evaluator Remark (if Any):	**Marks Secured:_____out of 50**
	Signature of the Evaluator with Date