Error Handling

1️⃣When is it Good to Crash?

✔ Irrecoverable System State: Crashing when the system can't recover (e.g., memory corruption).
✔ Security Violations: When continuing would expose vulnerabilities (e.g., buffer overflow).
✔ Data Corruption Risk: If continuing may corrupt data (e.g., database corruption).
✔ Invariant Violations: When the program reaches an impossible state.
✔ Development Failures: In development, crashing helps catch issues early.

2️⃣When is it Good to Surface Errors to Users?

✔ User Input Errors: Invalid form entries, passwords, or missing fields.
✔ Connectivity Issues: When users can fix their internet or network.
✔ Permission Denied: If a user tries to access something they don’t have rights for.
✔ File Not Found: When a user-specified file is missing.
✔ Payment Failures: Inform users of payment or order failures with reasons.

3️⃣When is it Good to Hide Errors from Users?

✔ Internal System Errors: Low-level system issues users can’t fix.
✔ Performance Issues: Temporary slowdowns that don’t affect functionality.
✔ Background Errors: Errors handled automatically, like retries for email failures.
✔ Security-Sensitive Errors: Avoid exposing details on authentication failures.
✔ Minor Recoverable Failures: Errors that are automatically handled (e.g., retrying API calls).

4️⃣Error Handling Strategies

✔ Crashing: For critical issues that require termination.
✔ Failure Responses: Use error codes, exceptions, or status classes to return error information.
✔ Recovering from Errors: Requires good design and testing for automatic recovery.
✔ Recording Information: Use logging to track errors for debugging.

📌 Error handling requires deciding when to crash, surface, or hide errors based on the severity. Use strategies like error codes, exceptions, and logging for better stability and usability.