Exception Handling:

Identify Common Exceptions:

Start by identifying common exceptions that can occur in your travel approval application. These might include database connection errors, input validation errors, authorization issues, and more. Create a list of expected exceptions.

Use Try-Catch Blocks:

Surround the code that might generate exceptions with try-catch blocks. In each catch block, handle a specific type of exception and perform appropriate actions.

```
try {
   // Code that may throw exceptions
} catch (DatabaseConnectionException ex) {
   // Handle database connection error
} catch (ValidationException ex) {
   // Handle input validation error
} catch (AuthorizationException ex) {
   // Handle authorization issue
} catch (Exception ex) {
   // Handle any other unexpected exceptions
}
```

Custom Exception Classes:

Create custom exception classes to represent specific application-level exceptions. This can make your code more organized and maintainable. For instance, you might create a TravelRequestValidationException class to handle validation errors.

```
public class TravelRequestValidationException extends Exception {
public TravelRequestValidationException(String message) {
    super(message);
}
}
```

Logging:

Always log exceptions and relevant information, such as error messages, timestamps, and stack traces. This information is invaluable for diagnosing and debugging issues. Use a logging framework like Log4j or Logback.

```
catch (DatabaseConnectionException ex) {
   log.error("Database connection error: " + ex.getMessage());
}
```

Graceful Error Messages:

Provide clear and user-friendly error messages for different types of exceptions. This is especially important for user-facing errors. When displaying error messages to users, avoid exposing technical details that could be exploited.

Transaction Rollback:

In the case of database-related exceptions, consider rolling back any ongoing transactions to maintain data integrity.

Notification:

Depending on the severity of the exception, you may want to notify administrators or support teams about critical errors through email, SMS, or other alerting mechanisms.

Retries:

Implement retry mechanisms for certain exceptions. For example, if a network-related exception occurs, you might attempt the operation again a certain number of times before giving up.

Fallback Mechanisms:

For critical parts of the application, consider implementing fallback mechanisms or alternative ways of achieving the same task if an exception occurs. This can help ensure essential processes continue to function.

Testing:

Thoroughly test your exception handling by intentionally causing exceptions during testing to ensure your application responds as expected.

Documentation:

Document the exceptions your application can throw, the conditions under which they occur, and how they should be handled.