**Experiment Report - 23 - test1\_demoCode**

1. **Summary Table of Errors Found**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Error ID | Line Number | Error Type | CSCR? | Self-Detected? | LLM? |
| E01 | line 11 | Syntax | √ |  | √ |
| E02 | line 28 | Semantic | × |  | × |
| E03 | line 36 | Semantic | √ |  | × |
| E04 | line 47 | Semantic | √ |  | √ |
| E05 | line 5 (52) | Logic |  | √ | × |
| E06 | line 20 (67) | Semantic |  | √ | √ |
| E07 | line 37 (84) | Semantic |  | × | × |
| E08 | line 50 (97) | Syntax |  | √ | √ |

Additional Errors Found by Self: 0

CSCR Rate: 75%

Self-Review Detection Rate: 75%

LLM Rate: 50%

1. **Source Code**
2. *#include <string>*
3. *#include <fstream>*
4. void CameraController::setPoliceInfo(int id, std::string name, std::string department){
5. PolicemanInfo info(id, name, department);
6. this->policemanInfo = info;
7. }
8. void CameraController::configureVideoSettings() {
9. videoConfig.setFrameRate(30fps);                 *// 30 fps*
10. videoConfig.setResolution("1920x1080");       *// Full HD*
11. videoConfig.setEncodingStandard("H264");      *// H.264 encoding*
12. videoConfig.setBitrate(2500);                 *// 2500 kbps*
13. }
14. void CameraController::startRecording() {
15. configureVideoSettings();
16. videoRecorder = std::make\_unique<VideoRecorder>(timeManager, gpsModule, videoConfig);
17. if (videoRecorder) {
18. std::cout << "Starting video recording..." << std::endl;
19. videoRecorder->startRecording();
20. std::cout << "Recording started successfully." << std::endl;
21. } else {
22. std::cout << "Error: VideoRecorder instance could not be created." << std::endl;
23. }
24. }
25. void CameraController::stopRecording() {
26. if (videoRecorder && videoRecorder->getCurrentVideoFilePath().empty() == false) {
27. std::cout << "Stopping video recording..." << std::endl;
28. videoRecorder->stopRecording();
29. std::cout << "Recording stopped. File saved: " << videoRecorder->getCurrentVideoFilePath() << std::endl;
30. } else {
31. std::cerr << "Error: No active recording to stop." << std::endl;
32. }
33. }
34. void CameraController::encryptAndStoreVideo(const std::string& filePath, std::string& videoData){
35. std::string encryptedData = this->encryptionModule.encrypt(videoData);
36. std::string encryptedFilePath = this->storageManager.generateEncryptedFileName(filePath);
37. this->storageManager.write(videoData);
38. }
39. VideoRecorder::VideoRecorder(TimeManager& timeManager, GPSModule& gpsModule, const VideoConfig& config)
40. : timeManager(timeManager), gpsModule(gpsModule), videoConfig(config), recording(true) {}
41. VideoRecorder::~VideoRecorder() {
42. if (recording) {
43. stopRecording();
44. }
45. }
46. void VideoRecorder::startRecording() {
47. if (recording) {
48. std::cerr << "Error: Recording is already in progress." << std::endl;
49. return;
50. }
51. std::string timestamp = timeManager.getCurrentTimestamp();
52. currentVideoFilePath = "video\_" + timestamp + "mp4";
53. videoFileStream.open(currentVideoFilePath, std::ios::binary);
54. if (!videoFileStream.is\_open()) {
55. std::cerr << "Error: Failed to open video file for recording." << std::endl;
56. return;
57. }
58. writeMetadataHeader();
59. recording = true;
60. std::cout << "Recording started. Saving to: " << currentVideoFilePath << std::endl;
61. for (int i = 0; i < 150; ++i) {
62. if (!recording) break;
63. embedTimestampAndGPS();
64. videoFileStream << "VideoFrameData";
65. std::this\_thread::sleep\_for(std::chrono::milliseconds(30)); *// Simulate 30 fps (1000/30 �� 33 ms)*
66. }
67. }
68. void VideoRecorder::stopRecording() {
69. if (!recording) {
70. std::cerr << "Error: No ongoing recording to stop." << std::endl;
71. return;
72. }
73. videoFileStream.close();
74. recording = false;
75. std::cout << "Recording stopped. File saved at: " << currentVideoFilePath << std:endl;
76. }
77. bool VideoRecorder::isRecording() const {
78. return recording;
79. }
80. std::string VideoRecorder::getCurrentVideoFilePath() const {
81. return recording ? currentVideoFilePath : "";
82. }